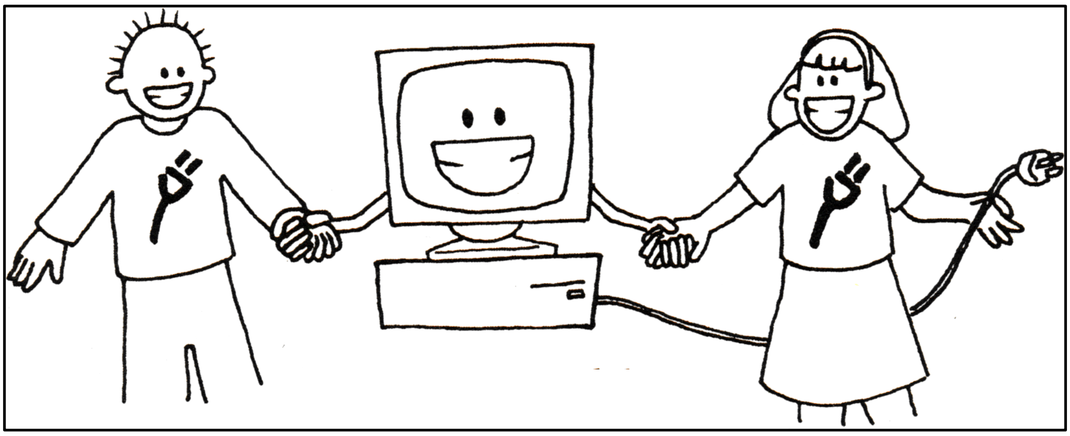
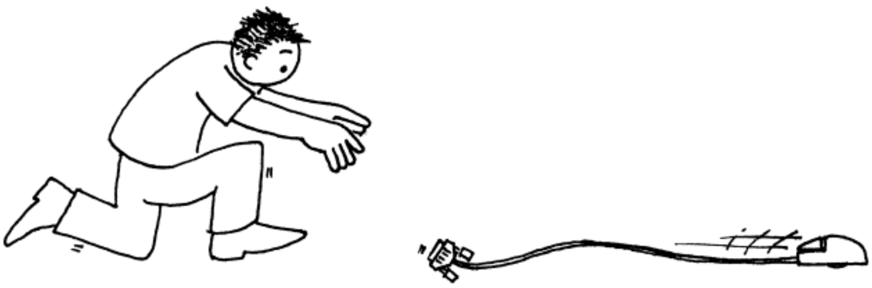
CS UNPLUGGED



An enrichment and extension programme for primary-aged students



Created by

Tim Bell, Ian H. Witten and Mike Fellows

Adapted for classroom use by Robyn Adams and Jane McKenzie

Illustrations by Matt Powell

2015 Revision by Sam Jarman

Introduction

Computers are everywhere. We all need to learn how to use them, and many of us use them every day. But how do they work? How do they think? And how can people write software that is fast and easy to use? Computer science is a fascinating subject that explores these very questions. The easy and fun activities in this book, designed for studentren of all ages, introduce you to some of the building blocks of how computers work—without using a computer at all!

This book can be effectively used in enrichment and extension programmes, or even in the regular classroom. You don't have to be a computer expert to enjoy learning these principles with your students. The book contains a range of activities, with background information explained simply. Answers to all problems are provided, and each activity ends with a 'what's it all about?' section that explains the relevance of the activities.

Many of the activities are mathematically based, e.g. exploring binary numbers, mapping and graphs, patterns and sorting problems, and cryptography. Others link in well with the technology curriculum, and the knowledge and understanding of how computers work. The studentren are actively involved in communication, problem solving, creativity, and thinking skills in a meaningful context. The activities also provide a very engaging way to explore "computational thinking", which is gaining traction in school curricula.

In addition to this book, the "Unplugged" project has as lot of free, online resources including videos, pictures and extra material at csunplugged.org. As part of the 2015 revision of this book, we have also released a brand new website, with more resources, better access to the open source material, and stronger curriculum links to match the appearance of computer science and computational thinking in school curricula.

This book was written by three computer science lecturers and two school teachers, and is based on our experience in classrooms as well as feedback from hundreds of educators over two decades. We have found that many important concepts can be taught without using a computer—in fact, sometimes the computer is just a distraction from learning. Often computer science is taught using programming first, but not every student finds this motivating, and it can be a significant barrier to getting into the really interesting ideas in computer science. So unplug your computer, and get ready to learn what computer science is really about!

This book has been translated into many languages. Please check the web site for information about the availability of translations.

Acknowledgements

Many children and teachers have helped us to refine our ideas. The children and teachers at South Park School (Victoria, BC), Shirley Primary School, Ilam Primary School and Westburn Primary School (Christchurch, New Zealand) were guinea pigs for many activities. We are particularly grateful to Linda Picciotto, Karen Able, Bryon Porteous, Paul Cathro, Tracy Harrold, Simone Tanoa, Lorraine Woodfield, and Lynn Atkinson for welcoming us into their classrooms and making helpful suggestions for refinements to the activities. Gwenda Bensemann has trialed several of the activities for us and suggested modifications. Richard Lynders and Sumant Murugesh have helped with classroom trials. Parts of the cryptography activities were developed by Ken Noblitz. Some of the activities were run under the umbrella of the Victoria "Mathmania" group, with help from Kathy Beveridge. Earlier versions of the illustrations were done by Malcolm Robinson and Gail Williams, and we have also benefited from advice from Hans Knutson. Matt Powell has also provided valuable assistance during the development of the "Unplugged" project. We are grateful to the Brian Mason Scientific and Technical Trust for generous sponsorship in the early stages of the development of this book.

Special thanks go to Paul and Ruth Ellen Howard, who tested many of the activities and provided a number of helpful suggestions. Peter Henderson, Bruce McKenzie, Joan Mitchell, Nancy Walker-Mitchell, Gwen Stark, Tony Smith, Tim A. H. Bell[1], Mike Hallett, and Harold Thimbleby also provided numerous helpful comments.

We owe a huge debt to our families: Bruce, Fran, Grant, Judith, and Pam for their support, and Andrew, Anna, Hannah, Max, Michael, and Nikki who inspired much of this work,[2] and were often the first children to test an activity.

We are particularly grateful to Google Inc. for sponsoring the Unplugged project, and enabling us to make this edition available as a free download.

We welcome comments and suggestions about the activities. The authors can be contacted via csunplugged.org.

[1] No relation to the first author.
[2] In fact, the text compression activity was invented by Michael.

Contents

Part I

Data: the raw material—*Representing information*

Data: The Raw Material

How can we store information in computers?

The word computer comes from the Latin *computare*, which means to calculate or add together, but computers today are more than just giant calculators. They can be a library, help us to write, find information for us, play music and even show movies. So how do they store all this information? Believe it or not, the computer uses only two things: zero and one!

What is the difference between data and information?

Data is the raw material, the numbers that computers work with. A computer converts its data into information (words, numbers and pictures) that you and I can understand.

How can numbers, letters, words and pictures be converted into zeros and ones?

In this section we will learn about binary numbers, how computers draw pictures, how fax machines work, what is the most efficient way to store lots of data, how we can prevent errors from happening and how we measure the amount of information we are trying to store.

Activity 1

Count the Dots—*Binary Numbers*

Summary

Data in computers is stored and transmitted as a series of zeros and ones. How can we represent words and numbers using just these two symbols?

Curriculum Links

- ✓ Mathematics: Number – Exploring numbers in other bases. Representing numbers in base two.
- ✓ Mathematics: Algebra – Continue a sequential pattern, and describe a rule for this pattern. Patterns and relationships in powers of two.

Skills

- ✓ Counting
- ✓ Matching
- ✓ Sequencing

Ages

- ✓ 6 and up

Materials

- ✓ You will need to make a set of five binary cards (see page 7) for the demonstration.
  A4 cards with smiley face sticker dots work well.

Each student will need:

- ✓ A set of five cards.
  Copy Photocopy Master: Binary numbers (page 7) onto card and cut out.
- ✓ Worksheet Activity: Binary numbers (page 6)

There are optional extension activities, for which each student will need:

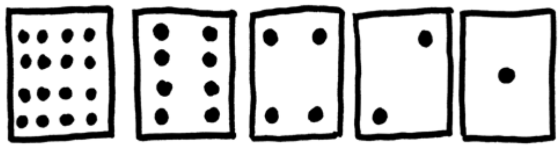
- ✓ Worksheet Activity: Working with binary (page 8)
- ✓ Worksheet Activity: Sending secret messages (page 9)
- ✓ Worksheet Activity: Email and modems (page 10)
- ✓ Worksheet Activity: Counting higher than 31 (page 11)
- ✓ Worksheet Activity: More on binary numbers (page 12)

Binary Numbers

Introduction

Before giving out the worksheet on page 6, it can be helpful to demonstrate the principles to the whole group.

For this activity, you will need a set of five cards, as shown below, with dots on one side and nothing on the other. Choose five students to hold the demonstration cards at the front of the class. The cards should be in the following order:



Discussion

As you give out the cards (from right to left), see if the students can guess how many dots are on the next card. What do you notice about the number of dots on the cards? (Each card has twice as many as the card to its right.)

How many dots would the next card have if we carried on to the left? (32) The next…? (64)

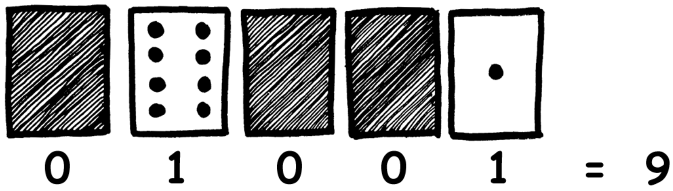
We can use these cards to make numbers by turning some of them face down and adding up the dots that are showing. Ask the students to show 6 dots (4-dot and 2-dot cards), then 15 (8-, 4-, 2- and 1-dot cards), then 21 (16, 4 and 1)… The only rule is that a card has to be completely visible, or completely hidden.

What is the smallest number of dots possible? (They may answer one, but it's zero).

Now try counting from zero onwards.

The rest of the class needs to look closely at how the cards change to see if they can see a pattern in how the cards flip (each card flips half as often as the one to its right). You may like to try this with more than one group.

When a binary number card is **not** showing, it is represented by a zero. When it **is** showing, it is represented by a one. This is the binary number system.



Ask the students to make 01001. What number is this in decimal? (9) What would 17 be in binary? (10001)

Try a few more until they understand the concept.

There are five optional follow-up extension activities, to be used for reinforcement. The students should do as many of them as they can.

Worksheet Activity: Binary Numbers

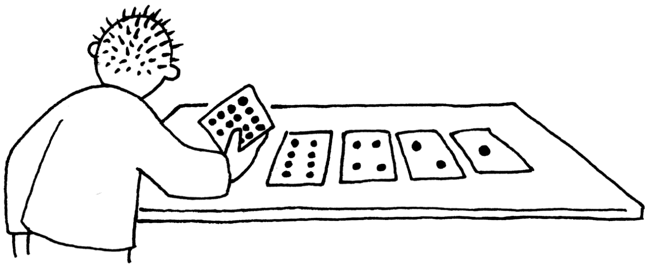
Learning how to count

So, you thought you knew how to count? Well, here is a new way to do it!

Did you know that computers use only zero and one? Everything that you see or hear on the computer—words, pictures, numbers, movies and even sound is stored using just those two numbers! These activities will teach you how to send secret messages to your friends using exactly the same method as a computer.

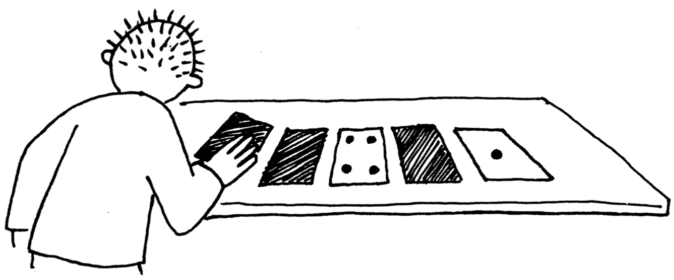
Instructions

Cut out the cards on your sheet and lay them out with the 16-dot card on the left as shown here:



Make sure the cards are placed in exactly the same order.

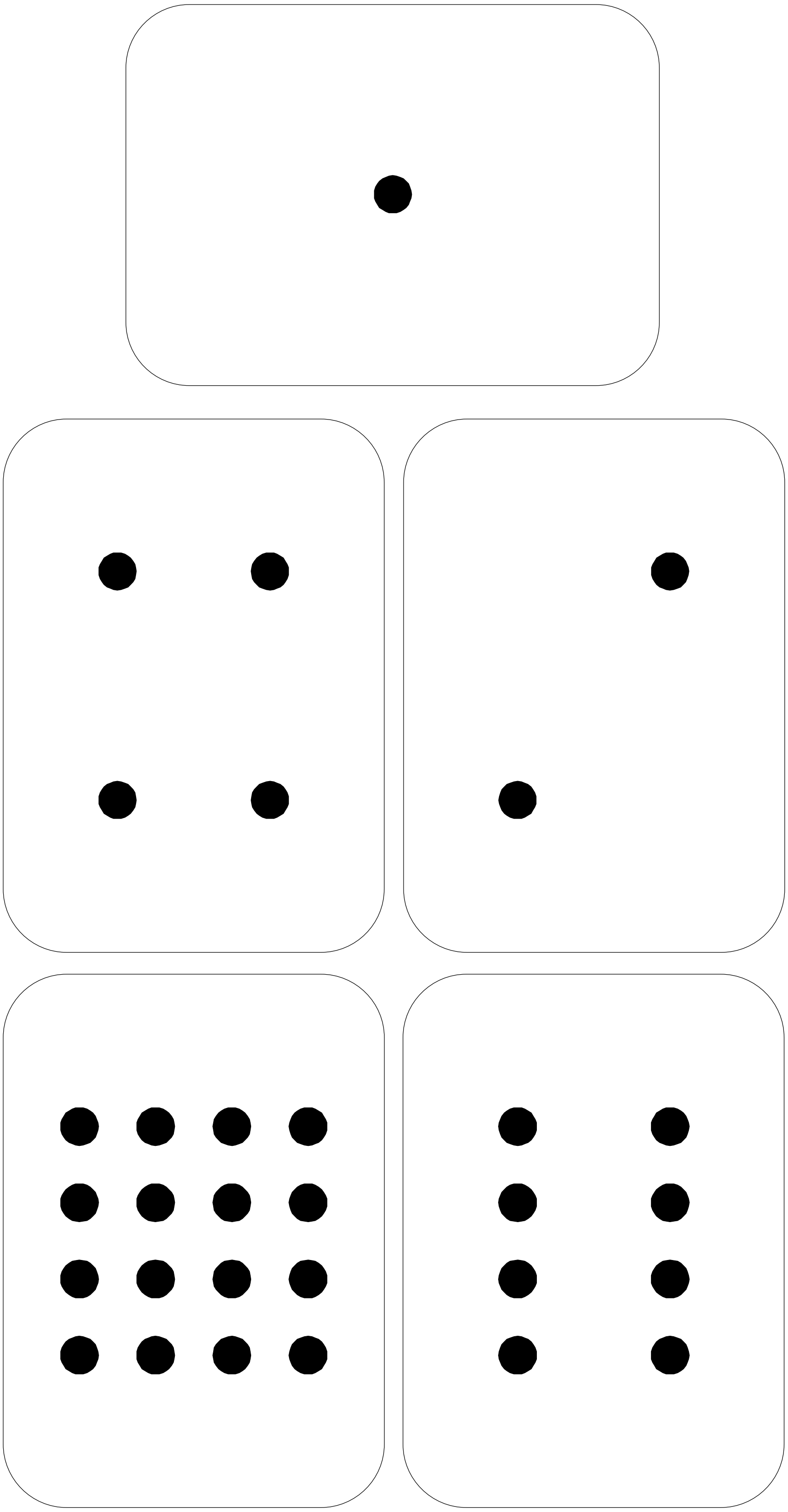
Now flip the cards so exactly 5 dots show—keep your cards in the same order!



Find out how to get 3, 12, 19. Is there more than one way to get any number? What is the biggest number you can make? What is the smallest? Is there any number you can't make between the smallest and biggest numbers?

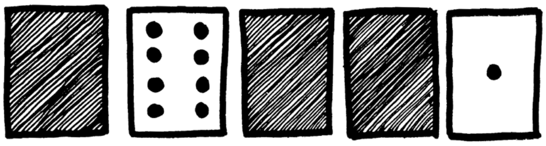
Extra for Experts: Try making the numbers 1, 2, 3, 4 in order. Can you work out a logical and reliable method of flipping the cards to increase any number by one?

Photocopy Master: Binary Numbers



Worksheet Activity: Working With Binary

The binary system uses **zero** and **one** to represent whether a card is face up or not. **0** shows that a card is hidden, and **1** means that you can see the dots. For example:



0 1 0 0 1 = 9

Can you work out what **10101** is? What about **11111**?

What day of the month were you born? Write it in binary. Find out what your friend's birthdays are in binary.

Try to work out these coded numbers:

☒☑☒☒☑ =
(☑=1, ☒=0)

👍👎👍👎 =
(👍=1, 👎=0)

⇧⇩⇧ =
(⇧=1, ⇩=0)

✚✚✕✚ =
(✚=1, ✕=0)

○○○○○ =
(⊙=1, ○=0)

↻↺↺↺↻ =
(↻=1, ↺=0)

📫📪 =
(📫=1, 📪=0)

▲▼▲▼▼ =
(▲=1, ▼=0)

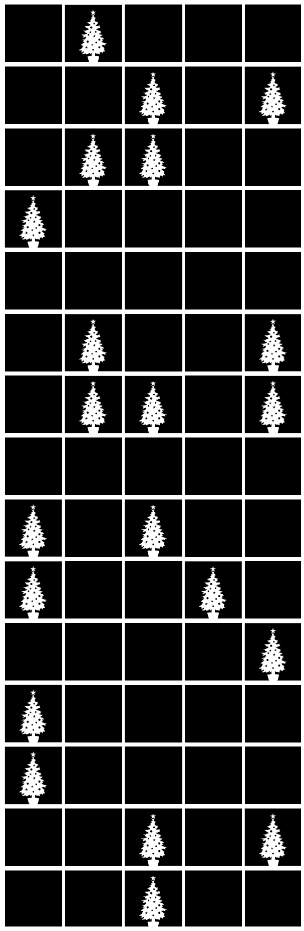
☹ =
(☺=1, ☹=0)

♠♠♠♠♠ =
(♠=1, ♣=0)

Extra for Experts: Using a set of rods of length 1, 2, 4, 8 and 16 units show how you can make any length up to 31 units. Or you could surprise an adult and show them how they only need a balance scale and a few weights to be able to weigh those heavy things like suitcases or boxes!

Worksheet Activity: Sending Secret Messages

Tom is trapped on the top floor of a department store. It's just before Christmas and he wants to get home with his presents. What can he do? He has tried calling, even yelling, but there is no one around. Across the street he can see some computer person still working away late into the night. How could he attract her attention? Tom looks around to see what he could use. Then he has a brilliant idea—he can use the Christmas tree lights to send her a message! He finds all the lights and plugs them in so he can turn them on and off. He uses a simple binary code, which he knows the woman across the street is sure to understand. Can you work it out?



1	2	3	4	5	6	7	8	9	10	11	12	13
a	b	c	d	e	f	g	h	i	j	k	l	m
14	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
n	o	p	q	r	s	t	u	v	w	x	y	z

Worksheet Activity: E-mail and Modems

Computers connected to the internet through a modem also use the binary system to send messages. The only difference is that they use beeps. A high-pitched beep can be used for a one and a low-pitched beep for a zero. These tones go very fast—so fast, in fact, that all we can hear is a horrible continuous screeching sound. If you have never heard it, listen to a modem connecting to the Internet, or try calling a fax machine—fax machines also use modems to send information.

Using the same code that Tom used in the department store, try sending an e-mail message to your friend. Make it easy for yourself and your friend though—you don't have to be as fast as a real modem!

Worksheet Activity: Counting higher than 31

Look at the binary cards again. If you were going to make the next card in the sequence, how many dots would it have? What about the next card after that? What is the rule that you are following to make your new cards? As you can see, only a few cards are needed to count up to very big numbers.

If you look at the sequence carefully, you can find a very interesting relationship:

1, 2, 4, 8, 16…

Try adding: 1 + 2 + 4 = ?   What does it come to?

Now try 1 + 2 + 4 + 8 = ?

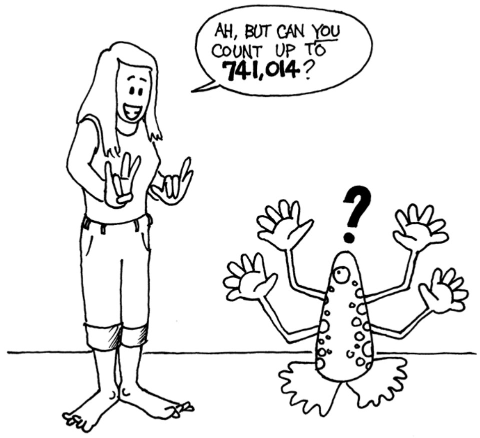
What happens if you add all the numbers up from the beginning?

Have you ever heard of "let your fingers do the walking"? Well now you can let your fingers do the counting, but you can get much higher than ten—no, you don't have to be an alien! If you use the binary system and let each finger on one hand represent one of the cards with dots you can count from 0–31. That's 32 numbers. (Don't forget that zero is a number too!)

Try counting in order using your fingers. If a finger is up it is a one, and if it is down it is a zero.

You can actually get from 0–1023 if you use both hands! That's 1024 numbers!

If you had really bendy toes (now you would have to be an alien) you could get even higher. If one hand can be used to count 32 numbers, and two hands can count to 32 × 32 = 1024 numbers, what is the biggest number Miss Flexi-Toes can reach?



Worksheet Activity: More on Binary Numbers

1. Another interesting property of binary numbers is what happens when a zero is put on the right hand side of the number. If we are working in base 10 (decimal), when you put a zero on the right hand side of the number, it is multiplied by 10. For example, 9 becomes 90, 30 becomes 300.

   But what happens when you put a 0 on the right of a binary number? Try this:

   1001 → **10010**

   (9) (?)

   Make up some others to test your hypothesis. What is the rule? Why do you think this happens?

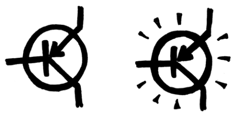
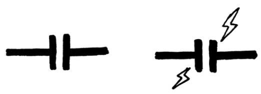
2. Each of the cards we have used so far represents a 'bit' on the computer ('bit' is short for '**b**inary dig**it**'). So our alphabet code we have used so far can be represented using just five cards, or 'bits'. However a computer has to know whether letters are capitals or not, and also recognise digits, punctuation and special symbols such as **$** or **~**.

   Go and look at a keyboard and work out how many characters a computer has to represent. So how many bits does a computer need to store all the characters?

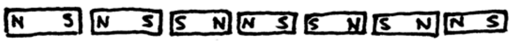
   Most computers today use a representation called ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange), which is based on using this number of bits per character, but some non-English speaking countries have to use longer codes.

What’s it all about?

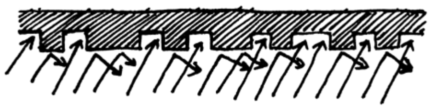
Computers today use the binary system to represent information. It is called binary because only two different digits are used. It is also known as base two (humans normally use base 10). Each zero or one is called a *bit* (**b**inary dig**it**). A bit is usually represented in a computer’s main memory by a transistor that is switched on or off, or a capacitor that is charged or discharged.

When data must be transmitted over a telephone line or radio link, high and low-pitched tones are used for the ones and zeros. On magnetic disks (hard disks and floppy disks) and tapes, bits are represented by the direction of a magnetic field on a coated surface, either North-South or South-North.



Audio CDs, CD-ROMs and DVDs store bits optically—the part of the surface corresponding to a bit either does or does not reflect light.

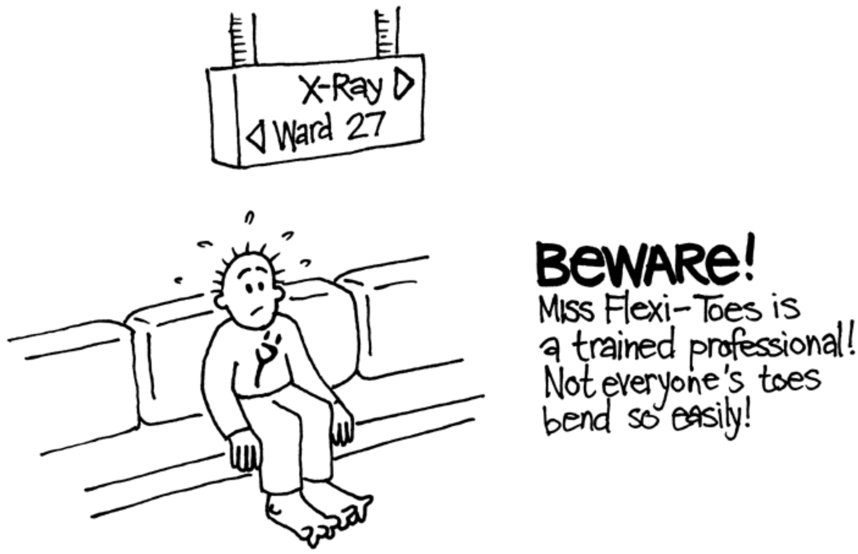


The reason that computers only use two different values is that it’s much easier to build devices that do it this way. We could have had CDs that have 10 levels of reflection so that we could represent all the digits from 0 to 9, but you have to build very expensive and precise devices to make it work. The other thing you may have noticed is that although we say that computers only store zeroes and ones, the actually don’t have zeroes and ones inside them – just high and low voltages, or north/south magnetism, and so on. But it’s quicker to write “0” and “1” than things like “shiny” and “not shiny”. *Everything* on computers is represented using these bits – documents, pictures, songs, videos, numbers, and even the programs and apps that we use are just a whole lot of binary digits.

One bit on its own can’t represent much, so they are usually grouped together in groups of eight, which can represent numbers from 0 to 255. A group of eight bits is called a byte.

The speed of a computer depends on the number of bits it can process at once. For example, a 32-bit computer can process 32-bit numbers in one operation, while a 16-bit computer must break 32-bit numbers down into smaller pieces, making it slower (but cheaper!)

In some of the later activities we will see how other kinds of information can be represented on a computer using binary digits.



Solutions and hints

Binary Numbers (page 6)

3 requires cards 2 and 1
12 requires cards 8 and 4
19 requires cards 16, 2 and 1

There is only one way to make any number.

The biggest number you can make is 31. The smallest is 0. You can make every number in between, and each has a unique representation.

Experts: To increase any number by one, flip all the cards from right to left until you turn one face up.

Working with binary (page 8)

10101 = 21, 11111 = 31

Sending Secret Messages (page 9)

Coded message: HELP IM TRAPPED

Counting higher than 31 (page 11)

If you add the numbers up from the beginning the sum will always be one less than the next number in the sequence.

Miss Flexi-toes can count 1024 × 1024 = 1,048,576 numbers—from 0 to 1,048,575!

More on Binary Numbers (page 12)

When you put a zero on the right hand side of a binary number the number doubles.

All of the places containing a one are now worth twice their previous value, and so the total number doubles. (In base 10 adding a zero to the right multiplies it by 10.)

A computer needs 7 bits to store all the characters. This allows for up to 128 characters. Usually the 7 bits are stored in an 8-bit byte, with one bit wasted.

Activity 2

Colour by Numbers—*Image Representation*

Summary

Computers store drawings, photographs and other pictures using only numbers. The following activity demonstrates how they can do this.

Curriculum Links

- ✓ Mathematics: Geometry – Shapes and Spaces
- ✓ Technology: using whole numbers to represent other kinds of data
- ✓ Technology: reducing the space used by repetitive data

Skills

- ✓ Counting
- ✓ Graphing

Ages

- ✓ 7 and up

Materials

- ✓ Slide for presenting: Colour by numbers (page 19)

Each student will need:

- ✓ Worksheet Activity: Kid Fax (page 20)
- ✓ Worksheet Activity: Make your own picture (page 21)

Colour by Numbers

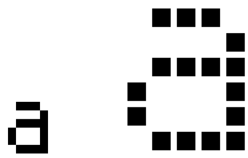
Introduction

Discussion Questions

1. What do facsimile (fax) machines do?
2. In what situations would computers need to store pictures? (A drawing program, a game with graphics, or a multi-media system.)
3. How can computers store pictures when they can only use numbers?

(You may like to arrange for the students to send and/or receive faxes as a preparation for this activity)

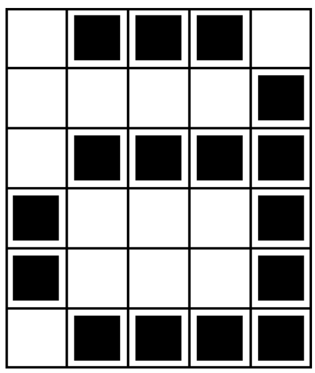
Demonstration using projection



Computer screens are divided up into a grid of small dots called *pixels* (**pic**ture **el**ements).

In a black and white picture, each pixel is either black or white.

The letter “a” has been magnified above to show the pixels. When a computer stores a picture, all that it needs to store is which dots are black and which are white.



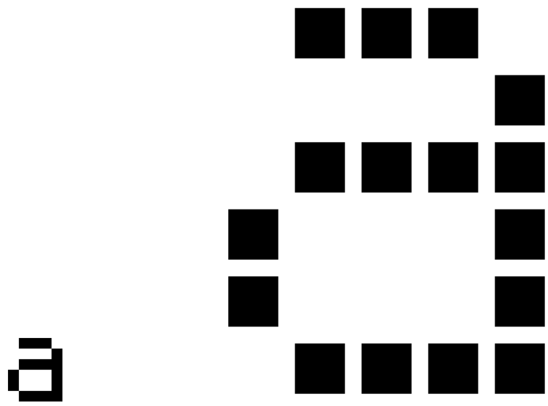
1, 3, 1
4, 1
1, 4
0, 1, 3, 1
0, 1, 3, 1
1, 4

The picture above shows us how a picture can be represented by numbers. The first line consists of one white pixel, then three black, then one white. Thus the first line is represented as 1, 3, 1.

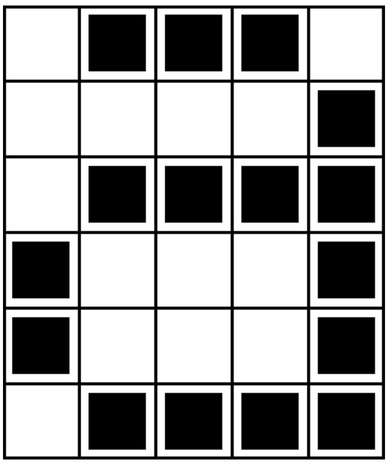
The first number always relates to the number of white pixels. If the first pixel is black the line will begin with a zero.

The worksheet on page 20 gives some pictures that the students can decode using the method just demonstrated.

Colour by numbers

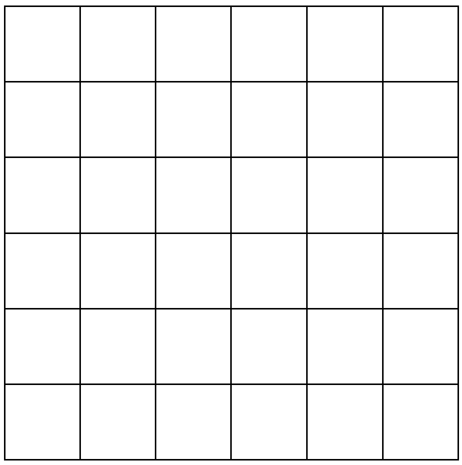


▲ A letter “a” from a computer screen and a magnified view showing the pixels that make up the image



1, 3, 1
4, 1
1, 4
0, 1, 3, 1
0, 1, 3, 1
1, 4

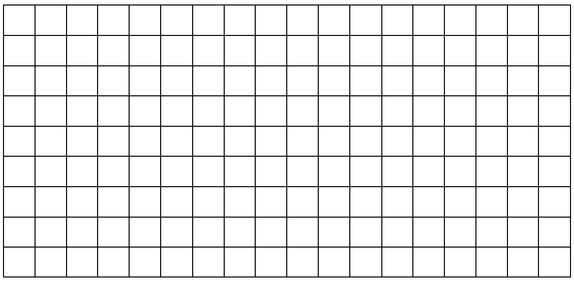
▲ The same image coded using numbers



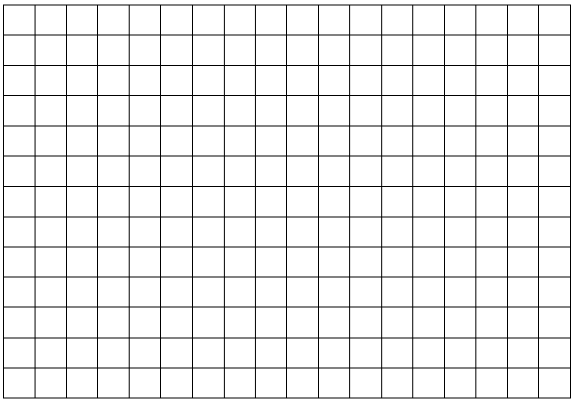
▲ Blank grid (for teaching purposes)

Worksheet Activity: Kid Fax

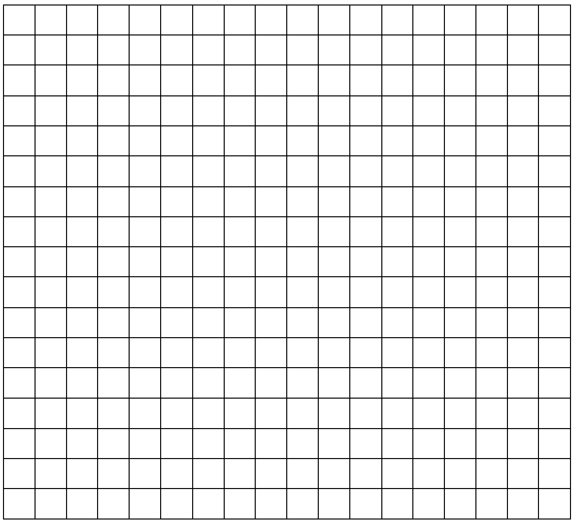
The first picture is the easiest and the last one is the most complex. It is easy to make mistakes and therefore a good idea to use a pencil to colour with and have a rubber handy!



4, 11
4, 9, 2, 1
4, 9, 2, 1
4, 11
4, 9
4, 9
5, 7
0, 17
1, 15



6, 5, 2, 3
4, 2, 5, 2, 3, 1
3, 1, 9, 1, 2, 1
3, 1, 9, 1, 1, 1
2, 1, 11, 1
2, 1, 10, 2
2, 1, 9, 1, 1, 1
2, 1, 8, 1, 2, 1
2, 1, 7, 1, 3, 1
1, 1, 1, 1, 4, 2, 3, 1
0, 1, 2, 1, 2, 2, 5, 1
0, 1, 3, 2, 5, 2
1, 3, 2, 5



6, 2, 2, 2
5, 1, 2, 2, 2, 1
6, 6
4, 2, 6, 2
3, 1, 10, 1
2, 1, 12, 1
2, 1, 3, 1, 4, 1, 3, 1
1, 2, 12, 2
0, 1, 16, 1
0, 1, 6, 1, 2, 1, 6, 1
0, 1, 7, 2, 7, 1
1, 1, 14, 1
2, 1, 12, 1
2, 1, 5, 2, 5, 1
3, 1, 10, 1
4, 2, 6, 2
6, 6

Worksheet Activity: Make Your Own Picture

Now that you know how numbers can represent pictures, why not try making your own coded picture for a friend? Draw your picture on the top grid, and when you've finished, write the code numbers beside the bottom grid. Cut along the dotted line and give the bottom grid to a friend to colour in. (Note: you don't have to use the whole grid if you don't want to—just leave some blank lines at the bottom if your picture doesn't take up the whole grid.)

Worksheet Activity: Make Your Own Picture

Extra for Experts: If you want to produce coloured images you can use a number to represent the colour (e.g. 0 is black, 1 is red, 2 is green etc.) Two numbers are now used to represent a run of pixels: the first gives the length of the run as before, and the second specifies the colour. Try making a coloured picture for a friend. Don’t forget to let your friend know which number stands for which colour!

Variations and Extensions

1. Try drawing with a sheet of tracing paper on top of the grid, so that the final image can be viewed without the grid. The image will be clearer.

2. Instead of colouring the grid the students could use squares of sticky paper, or place objects, on a larger grid.

Discussion Point

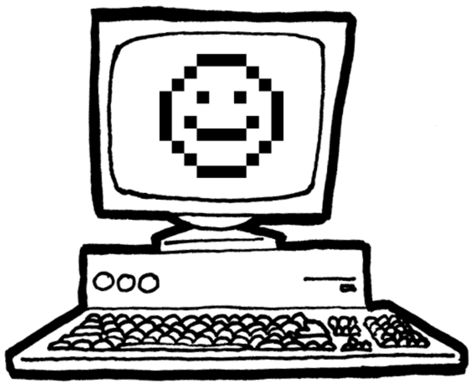
There is usually a limit to the length of a run of pixels because the length is being represented as a binary number. How would you represent a run of twelve black pixels if you could only use numbers up to seven? (A good way is to code a run of seven black pixels, followed by a run of zero white, then a run of five black.)

What's it all about?

A fax machine is really just a simple computer that scans a black and white page into about 1000 × 2000 pixels, which are sent using a modem to another fax machine, which prints the pixels out on a page. Often fax images have large blocks of white (e.g. margins) or black pixels (e.g. a horizontal line). Colour pictures also have a lot of repetition in them. To save on the amount of storage space needed to keep such images programmers can use a variety of compression techniques. The method used in this activity is called 'run-length coding', and is an effective way to compress images. If we didn't compress images it would take much longer to transmit pictures and require much more storage space. This would make it infeasible to send faxes or put photos on a web page. For example, fax images are generally compressed to about a seventh of their original size. Without compression they would take seven times as long to transmit!

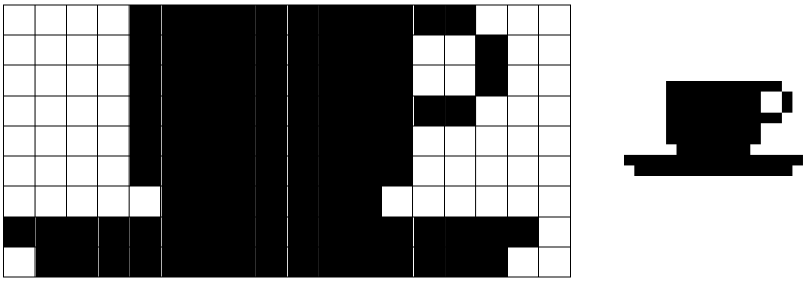
Photographs and pictures are often compressed to a tenth or even a hundredth of their original size (using a related techniques such as JPEG, GIF and PNG). This allows many more images to be stored on a disk, and it means that viewing them over the web will take a fraction of the time.

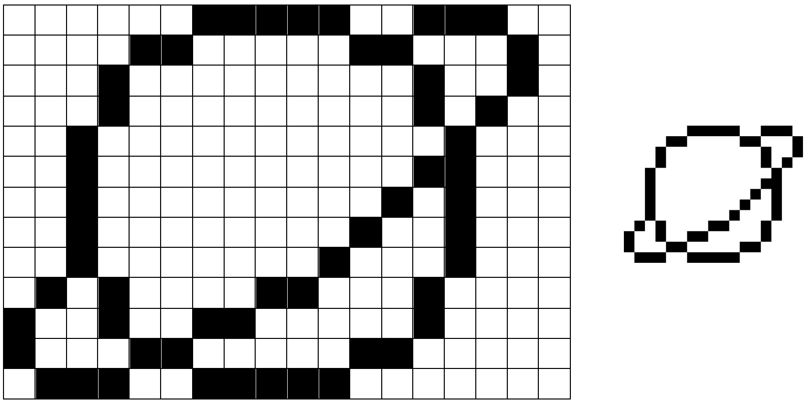
A programmer can choose which compression technique best suits the images he or she is transmitting.

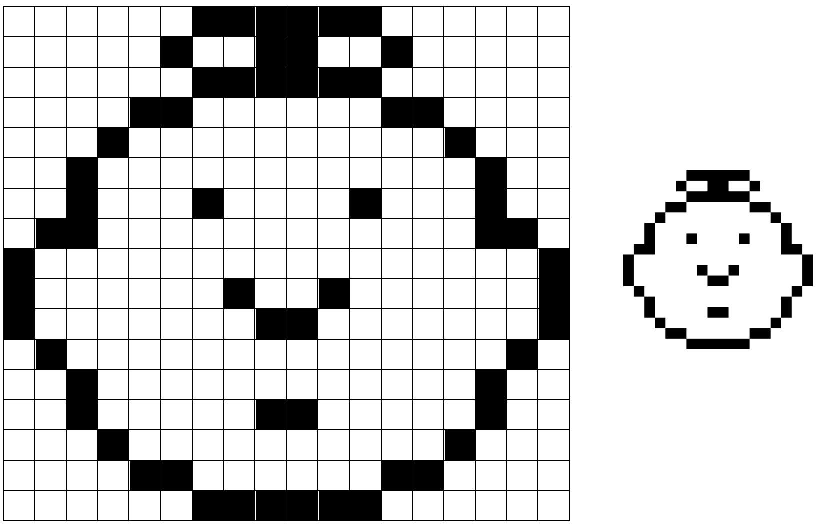


Solutions and hints

Answers to Kid Fax Worksheet







Activity 3

You Can Say That Again! —*Text Compression*

Summary

Since computers only have a limited amount of space to hold information, they need to represent information as efficiently as possible. This is called compression. By coding data before it is stored, and decoding it when it is retrieved, the computer can store more data, or send it faster through the Internet.

Curriculum Links

- ✓ English: Recognising patterns in words and text.
- ✓ Technology: reducing the space used by repetitive data

Skills

- ✓ Copying written text

Ages

- ✓ 9 and up

Materials

- ✓ Presentation Slide: You can say that again! (page 28)

Each student will need:

- ✓ Worksheet Activity: You can say that again! (page 29)
- ✓ Worksheet Activity: Extras for experts (page 30)
- ✓ Worksheet Activity: Short and sweet (page 31)
- ✓ Worksheet Activity: Extras for *real* experts (page 33)

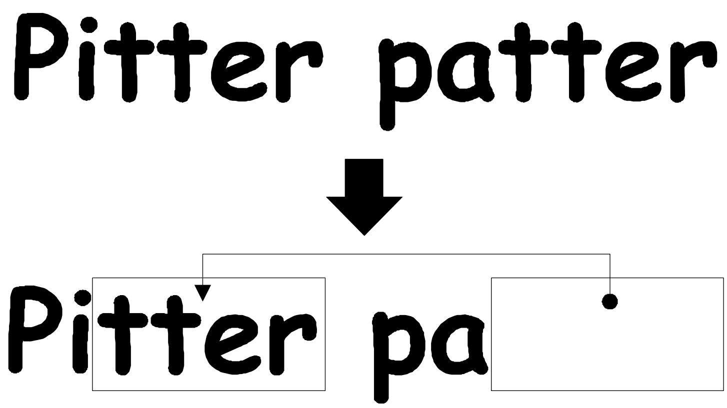
You can say that again!

Introduction

Computers have to store and transmit a lot of data. So that they don't have to use up too much storage space, or take too long to send information through a network connection, they compress the text a bit like this.

Demonstration and Discussion

Show "The Rain" slide (page 28). Look for the patterns of letters in this poem. Can you find groups of 2 or more letters that are repeated, or even whole words or phrases? (Replace these with boxes as shown in the diagram below.)



The Rain

Pitter patter

Pitter patter

Listen to the rain

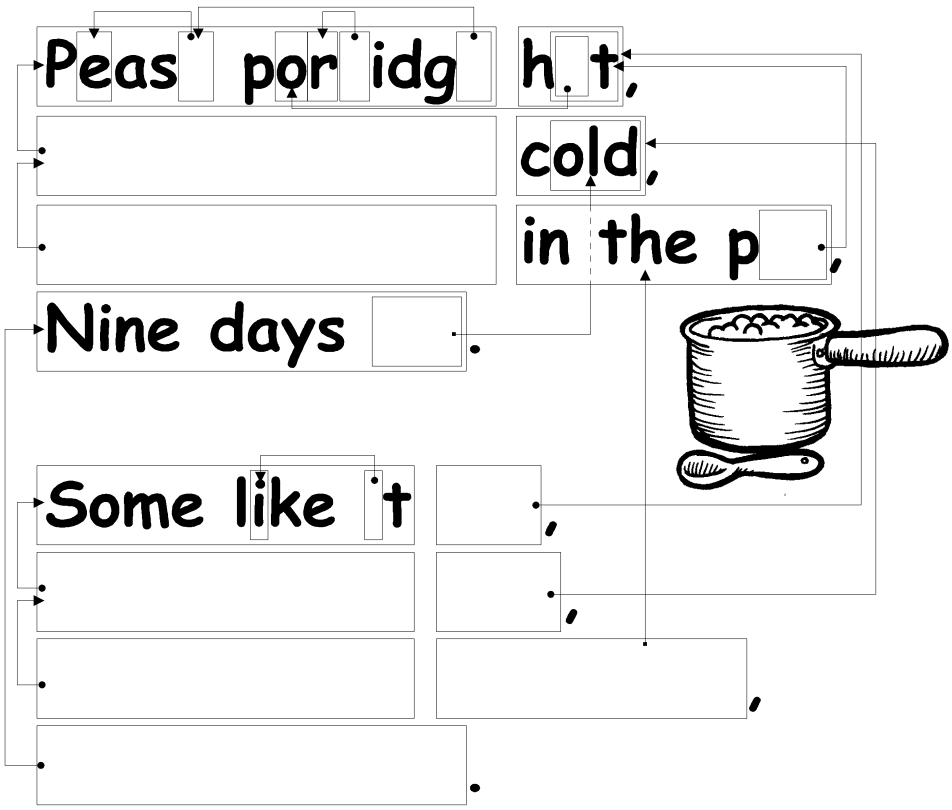
Pitter patter

Pitter patter

On the window pane

Worksheet Activity: You can say that again!

Many of the words and letters are missing in this poem. Can you fill in the missing letters and words to complete it correctly? You will find these in the box that the arrow is pointing to.



Now choose a simple poem or nursery rhyme and design your own puzzle. Make sure your arrows always point to an earlier part of the text. Your poem should be able to be decoded from left to right and from top to bottom in the same way we read.

Challenge: See how few of the original words you need to keep!

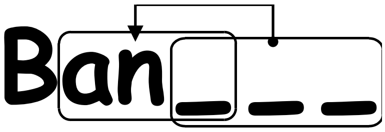
Here are some suggestions: Three Blind Mice, Mary Mary Quite Contrary, Hickory Dickory Dock—or try some Dr Seuss books!

Hint: Try to avoid overcrowding of arrows. Leave a lot of space around letters and words as you write them so that you have room for the boxes within boxes and the arrows pointing to them.

It is easier to design the puzzle if you write out the poem first and then decide where the boxes need to be.

Worksheet Activity: Extra for Experts

How would you solve this puzzle?



Sometimes missing text points to part of itself. In this case it can be decoded correctly if the letters are copied from left to right. Then each letter is available to be copied before it is needed. This is useful in computers if there is a long run of a particular character or pattern.

Try drawing some of your own.

On computers the boxes and arrows are represented by numbers. For example,

Banana

can be written as **Ban(2,3)**. "**2**" means count back two characters to find the starting point for copying,

Ban___

and "**3**" means copy three consecutive characters:

Bana__

Banan_

Banana

As two numbers are used to code these words, usually only groups of two or more letters are worth compressing, otherwise there is no saving of space. In fact the size of the file could go up if two numbers are used to code one letter.

Make up some words of your own written in the way a computer would if they were compressed. Can your friends decode them?

Worksheet Activity: Short and Sweet

How many words do you really need here?

Pretend you are a computer trying to fit as much into your disk as possible. Cross out all the groups of two or more letters that have already occurred. These are no longer needed as they could be replaced by a pointer. Your goal is to get as many letters crossed out as possible.

I know an old lady who swallowed a bird

How absurd! She swallowed a bird!

She swallowed the bird to catch the spider

That wriggled and jiggled

and tickled inside her

She swallowed the spider to catch the fly

I don’t know why she swallowed a fly

Perhaps she’ll die…

Worksheet Activity: Extra for *Real* Experts

Ready for some *really* tough compression?

The following story was run through a computer program, which found that there are at least 1,633 letters that can be crossed out. How many can you find? Remember, only groups of two or more repeated characters can be eliminated. Good luck!

Once upon a time, long, long ago, three little pigs set out to make their fortunes. The first little pig wasn't very clever, and decided to build his house out of straw, because it was cheap. The second little pig wasn't very clever either, and decided to build his house out of sticks, for the "natural" look that was so very much in fashion, even in those days. The third little pig was much smarter than his two brothers, and bought a load of bricks in a nearby town, with which to construct a sturdy but comfortable country home.

Not long after his housewarming party, the first little pig was curled up in a chair reading a book, when there came a knock at the door. It was the big bad wolf, naturally.

"Little pig, little pig, let me come in!" cried the wolf.

"Not by the hair on my chinny-chin-chin!" squealed the first little pig.

"Then I'll huff, and I'll puff, and I'll blow your house down!" roared the wolf, and he *did* huff, and he *did* puff, and the house soon collapsed. The first little pig ran as fast as he could to the house of sticks, and was soon safe inside. But it wasn't long before the wolf came calling again.

"Little pig, little pig, let me come in!" cried the wolf.

"Not by the hair on my chinny-chin-chin!" squealed the second little pig.

"Then I'll huff, and I'll puff, and I'll blow your house down!" roared the wolf, and he *did* huff, and he *did* puff, and the house was soon so much firewood. The two terrified little pigs ran all the way to their brother's brick house, but the wolf was hot on their heels, and soon he was on the doorstep.

"Little pig, little pig, let me come in!" cried the wolf.

"Not by the hair on my chinny-chin-chin!" squealed the third little pig.

"Then I'll huff, and I'll puff, and I'll blow your house down!" roared the wolf, and he huffed, and he puffed, and he huffed some more, but of course, the house was built of brick, and the wolf was soon out of breath. Then he had an idea. The chimney! He clambered up a handy oak tree onto the roof, only to find that there *was* no chimney, because the third little pig, being conscious of the environment, had installed electric heating. In his frustration, the wolf slipped and fell off the roof, breaking his left leg, and severely injuring his pride. As he limped away, the pigs laughed, and remarked how much more sensible it was to live in the city, where the only wolves were in the zoo. And so that is what they did, and of course they all lived happily ever after.

What’s it all about?

The storage capacity of computers is growing at an unbelievable rate—in the last 25 years, the amount of storage provided on a typical computer has grown about a millionfold—but we still find more to put into our computers. Computers can store whole books or even libraries, and now music and movies too, if only they have the room. Large files are also a problem on the Internet, because they take a long time to download. We also try to make computers smaller—even a cellphone or wristwatch can be expected to store lots of information!

There is a solution to this problem, however. Instead of buying more storage space, or a faster network connection, we can *compress* the data so that it takes up less space. This process of compressing and decompressing the data is normally done automatically by the computer. All we might notice is that the disk holds more, or that web pages display faster, but the computer is actually doing more processing.

Many methods of compression have been invented. The method used in this activity, with the principle of pointing to earlier occurrences of chunks of text, is often referred to as ‘Ziv-Lempel coding,’ or ‘LZ coding’, invented by two Israeli professors in the 1970s. It can be used for any language and can easily halve the size of the data being compressed. It is sometimes referred to as ‘zip’ on personal computers, and is also used for ‘GIF’ and ‘PNG’ images, and has been used in high-speed modems. In the case of modems, it reduces the amount of data that needs to be transmitted over the phone line, so it goes much faster.

Some other methods are based on the idea that letters that are used more often should have shorter codes than the others. Morse code used this idea.

Solutions and hints

You can say that again! (page 29)

Pease porridge hot,
Pease porridge cold,
Pease porridge in the pot,
Nine days old.

Some like it hot,
Some like it cold,
Some like it in the pot,
Nine days old.

Activity 4

Card Flip Magic—*Error Detection & Correction*

Summary

When data is stored on a disk or transmitted from one computer to another, we usually assume that it doesn't get changed in the process. But sometimes things go wrong and the data is changed accidentally. This activity uses a magic trick to show how to detect when data has been corrupted, and to correct it.

Curriculum Links

- ✓ Mathematics: Number – Exploring computation and estimation.
- ✓ Mathematics: Algebra – Exploring patterns and relationships, solving for a missing value.
- ✓ Mathematics: Rows and columns, coordinates
- ✓ Technology: Validating data

Skills

- ✓ Counting
- ✓ Recognition of odd and even numbers

Ages

- ✓ 7 years and up

Materials

- ✓ A set of 36 "fridge magnet" cards, coloured on one side only
- ✓ A metal board (a whiteboard works well) for the demonstration.

Each pair of students will need:

- ✓ 36 identical cards, coloured on one side only.

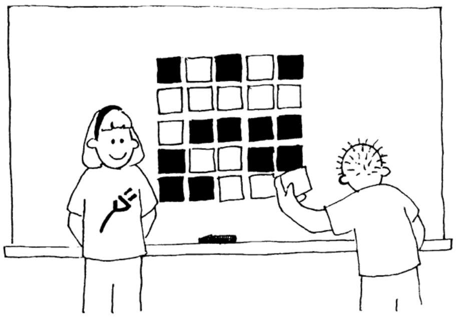
The “Magic Trick”

Demonstration

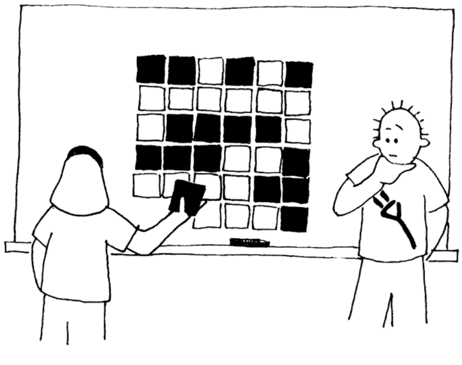
Here’s your chance to be a magician!

You will need a pile of identical, two-sided cards. (To make your own cut up a large sheet of card that is coloured on one side only). For the demonstration it is easiest to use flat magnetic cards that have a different colour on each side—fridge magnets are ideal, but make sure they are magnetic on both sides (many are single sided, in which case you can glue them face to face, and put a white dot on one side).

1. Choose a student to lay out the cards in a 5 × 5 square, with a random mixture of sides showing.



Casually add another row and column, “just to make it a bit harder”.



These cards are the key to the trick. You must choose the extra cards to ensure that there is an even number of coloured cards in each row and column.

2. Get a student to flip over one card only while you cover your eyes. The row and column containing the changed card will now have an odd number of coloured cards, and this will identify the changed card. Can the students guess how the trick is done?

Teach the trick to the students:

1. Working in pairs, the students lay out their cards 5 × 5.

2. How many coloured cards are there in each row and column? Is it an odd or even number? Remember, 0 is an even number.

3. Now add a sixth card to each row, making sure the number of coloured cards is always even. This extra card is called a "parity" card.

4. Add a sixth row of cards along the bottom, to make the number of cards in each column an even number.

5. Now flip a card. What do you notice about the row and column? (They will have an odd number of coloured cards.) Parity cards are used to show you when a mistake has been made.

6. Now take turns to perform the 'trick'.

Extension Activities:

1. Try using other objects. Anything that has two 'states' is suitable. For example, you could use playing cards, coins (heads or tails) or cards with 0 or 1 printed on them (to relate to the binary system).

2. What happens if two, or more, cards are flipped? (It is not always possible to know exactly which two cards were flipped, although it is possible to tell that something has been changed. You can usually narrow it down to one of two pairs of cards. With 4 flips it is possible that all the parity bits will be correct afterwards, and so the error could go undetected.)

3. Try this with a much larger layout e.g. 9 × 9 cards, with the extra row and column expanding it to 10 × 10. (It will work for any size layout, and doesn't have to be square).

4. Another interesting exercise is to consider the lower right-hand card. If you choose it to be the correct one for the column above, then will it be correct for the row to its left? (The answer is yes, always, if you use even parity.)

5. In this card exercise we have used even parity—using an even number of coloured cards. Can we do it with odd parity? (This is possible, but the lower right-hand card only works out the same for its row and column if the numbers of rows and columns are both even or both odd. For example, a 5 × 9 layout will work fine, or a 4 × 6, but a 3 × 4 layout won't.)

A Real-Life Example for Experts!

This same checking technique is used with book codes and bar codes. Published books have a ten- or 13-digit code usually found on the back cover. The last digit is a check digit, just like the parity bits in the exercise.

This means that if you order a book using its ISBN (International Standard Book Number), the website can check that you haven't made a mistake. They simply look at the checksum. That way you don't end up waiting for the wrong book!

Here's how to work out the checksum for a 10-digit book code:

Multiply the first digit by ten, the second by nine, the third by eight, and so on, down to the ninth digit multiplied by two. Each of these values is then added together.

For example, the ISBN 0-13-911991-4 gives a value

$$\begin{aligned} & (0 \times 10) + (1 \times 9) + (3 \times 8) + (9 \times 7) + (1 \times 6) \\ + \quad & (1 \times 5) + (9 \times 4) + (9 \times 3) + (1 \times 2) \\ = \quad & 172 \end{aligned}$$

Then divide your answer by eleven. What is the remainder?

$$172 \div 11 = \quad 15 \text{ remainder } 7$$

If the remainder is zero, then the checksum is zero, otherwise subtract the remainder from 11 to get the checksum.

$$11 - 7 = \quad 4$$

Look back. Is this the last digit of the ISBN? Yes!

If the last digit of the ISBN wasn't a four, then we would know that a mistake had been made.

It is possible to come up with a checksum of the value of 10, which would require more than one digit. When this happens, the character X is used.

9 414942 010910

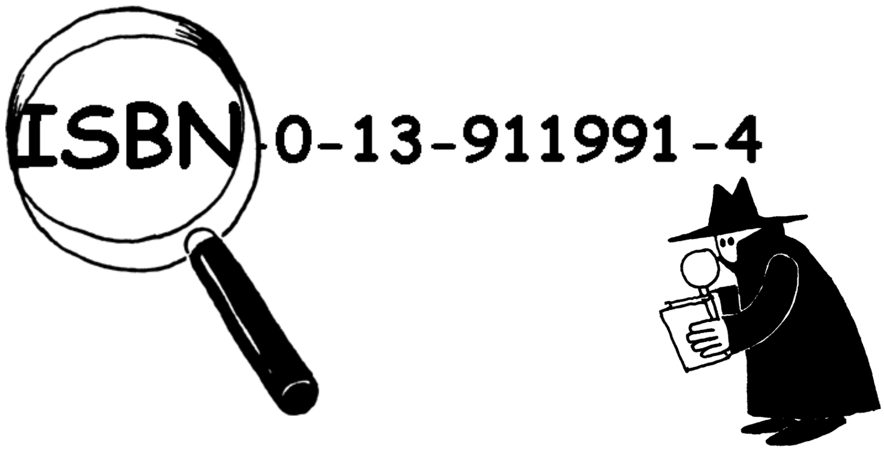
▲ A barcode (UPC) from a box of Weet-Bix™

Another example of the use of a check digit is the bar codes on grocery items. This uses a different formula (the same formula is used for 13-digit book codes). If a bar code is misread the final digit should be different from its calculated value. When this happens the scanner beeps and the checkout operator re-scans the code. Check digits are also used for bank account numbers, social security numbers, tax numbers, numbers on trains and rolling stock, and many other applications where people are copying a number and need some assurance that it has been typed in correctly.

Check that book!

Detective Blockbuster

Book Tracking Service, Inc.



We find and check ISBN checksums for a small fee.

Join our agency—look in your classroom or library for real ISBN codes.

Are their checksums correct?

Sometimes errors are made.

Some of the common errors are:

- a digit has its value changed;
- two adjacent digits are swapped with each other;
- a digit is inserted in the number; and
- a digit is removed from the number

Can you find a book with the letter X for a checksum of 10? It shouldn't be too hard to find—one in every 11 should have it.

What sort of errors might occur that wouldn't be detected? Can you change a digit and still get the correct checksum? What if two digits are swapped (a common typing error)?

What's it all about?

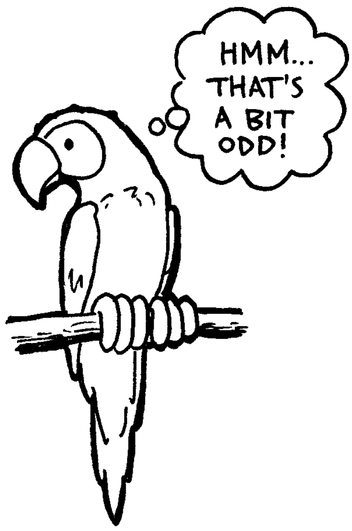
Imagine you are depositing $10 cash into your bank account. The teller types in the amount of the deposit, and it is sent to a central computer. But suppose some interference occurs on the line while the amount is being sent, and the code for $10 is changed to $1,000. No problem if you are the customer, but clearly a problem for the bank!

It is important to detect errors in transmitted data. So a receiving computer needs to check that the data coming to it has not been corrupted by some sort of electrical interference on the line. Sometimes the original data can be sent again when an error has been transmitted, but there are some situations when this is not feasible, for example if a disk has been corrupted by exposure to magnetic or electrical radiation, by heat or by physical damage. If data is received from a deep space probe, it would be very tedious to wait for retransmission if an error had occurred! (It takes just over half an hour to get a radio signal from Jupiter when it is at its closest to Earth!)

We need to be able to recognize when the data has been corrupted *(error detection)* and to be able to reconstruct the original data *(error correction)*.

The same technique as was used in the "card flip" game is used on computers. By putting the bits into imaginary rows and columns, and adding parity bits to each row and column, we can not only detect if an error has occurred, but *where* it has occurred. The offending bit is changed back, and so we have performed error correction.

Of course computers often use more complex error control systems that are able to detect and correct multiple errors. The hard disk in a computer has a large amount of its space allocated to correcting errors so that it will work reliably even if parts of the disk fail. The systems used for this are closely related to the parity scheme.



And to finish, a joke that is better appreciated after doing this activity:

Q: What do you call this: "Pieces of nine, pieces of nine"?

A: A parroty error.

Solutions and hints

Errors that would not be detected by an ISBN-10 checksum are those where one digit increases and another decreases to compensate. Then the sum might still be the same. However, because of the way the calculation is done, this is unlikely to happen. In other systems (such as ISBN-13) there are other types of errors that might not be detected, such as three consecutive digits being reversed, but most of the common errors (typing one digit incorrectly, or swapping two adjacent digits) will be detected.

Activity 5

Twenty Guesses—*Information Theory*

Summary

How much information is there in a 1000-page book? Is there more information in a 1000-page telephone book, or in a ream of 1000 sheets of blank paper, or in Tolkien's *Lord of the Rings*? If we can measure this, we can estimate how much space is needed to store the information. For example, can you still read the following sentence?

Ths sntnc hs th vwls mssng.

You probably can, because there is not much 'information' in the vowels. This activity introduces a way of measuring information content.

Curriculum links

- ✓ Mathematics: Number – Exploring number: Greater than, less than, ranges.
- ✓ Mathematics: Algebra – Patterns and sequences
- ✓ English: spelling, recognising elements of text

Skills

- ✓ Comparing numbers and working with ranges of numbers
- ✓ Deduction
- ✓ Asking questions

Ages

- ✓ 10 and up

Materials

- ✓ No materials are required for the first activity

There is an extension activity, for which each student will need:

- ✓ Worksheet Activity: Decision trees (page 47)

Twenty Guesses

Discussion

1. Discuss with the students what they think information is.

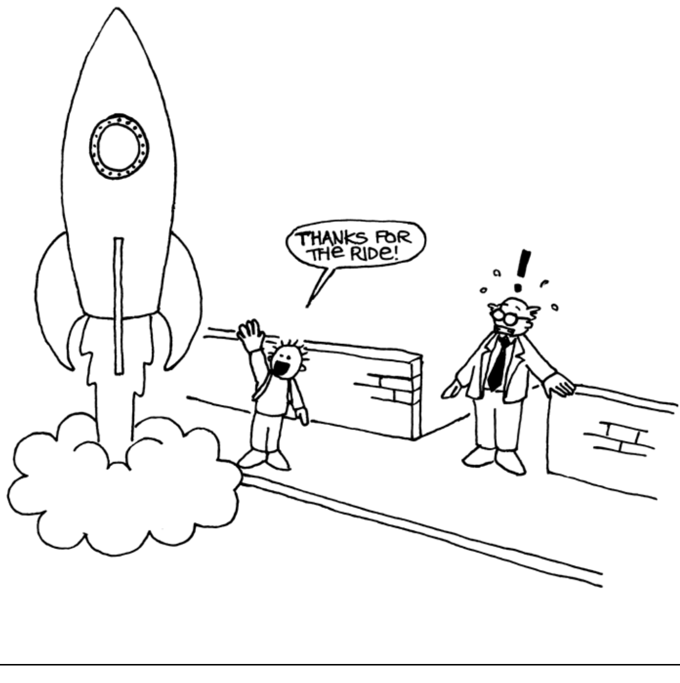
2. How could we measure how much information there would be in a book? Is the number of pages or number of words important? Can one book have more information than another? What if it is a very boring book, or a particularly interesting one? Would 400 pages of a book containing the phrase “blah, blah, blah” have more or less information than, say, the telephone directory?

3. Explain that computer scientists measure information by how surprising a message (or book!) is. Telling you something that you know already—for example, when a friend who always walks to school says “I walked to school today”—doesn’t give you any information, because it isn’t surprising. If your friend said instead, “I got a ride to school in a helicopter today,” that *would* be surprising, and would therefore tell us a lot of information.

4. How can the surprise value of a message be measured?

5. One way is to see how hard it is to guess the information. If your friend says, “Guess how I got to school today,” and they had walked, you would probably guess right first time. It might take a few more guesses before you got to a helicopter, and even more if they had travelled by spaceship.

6. The amount of information that messages contain is measured by how easy or hard they are to guess. The following game gives us some idea of this.



Twenty Questions Activity

This is an adapted game of 20 questions. Students may ask questions of a chosen student, who may only answer yes or no until the answer has been guessed. Any question may be asked, provided that the answer is strictly 'yes' or 'no'.

Suggestions:

I am thinking of:

- ✓ a number between 1 and 100
- ✓ a number between 1 and 1000
- ✓ a number between 1 and 1,000,000.
- ✓ any whole number
- ✓ a sequence of 6 numbers in a pattern (appropriate to the group). Guess in order from first to last. (e.g. 2, 4, 6, 8, 10)

Count the number of questions that were asked. This is a measure of the value of the "information".

Follow-up Discussion

What strategies did you use? Which were the best ones?

Point out that it takes just 7 guesses to find a number between 1 and 100 if you halve the range each time. For example:

Is it less than 50?	Yes.
Is it less than 25?	No.
Is it less than 37?	No.
Is it less than 43?	Yes.
Is it less than 40?	No.
Is it less than 41?	No.
It must be 42!	Yes!

Interestingly if the range is increased to 1000 it doesn't take 10 times the effort—just three more questions are needed. Every time the range doubles you just need one more question to find the answer.

A good follow up would be to let the students play Mastermind.

Extension: How much information is there in a message?

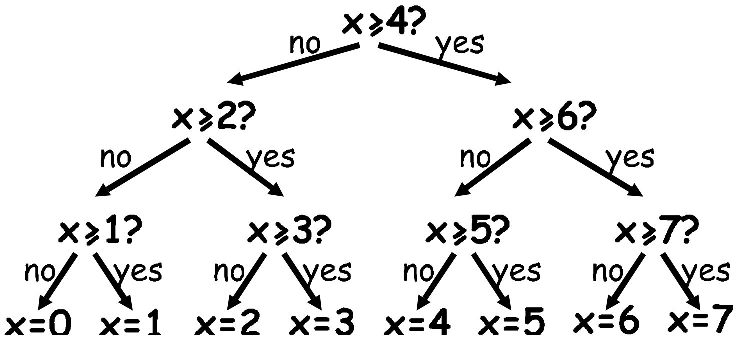
Computer scientists don't just use guessing with numbers—they can also guess which letter is more likely to be next in a word or sentence.

Try the guessing game with a short sentence of 4–6 words. The letters must be guessed in the correct order, from first to last. Get someone to write down the letters as they are found and keep a record of how many guesses it takes to find each letter. Any questions with a yes/no answer can be used. Examples would be, “It it a *t*?” “Is it a vowel?” “Does it come before *m* in the alphabet?” A space between words also counts as a “letter” and must be guessed. Take turns and see if you can discover which parts of messages are easiest to find out.

Worksheet Activity: Decision Trees

If you already know the strategy for asking the questions, you can transmit a message without having to ask anything.

Here is a chart called a 'decision tree' for guessing a number between 0 and 7:



What are the yes/no decisions needed to 'guess' the number 5?

How many yes/no decisions do you need to make to work out any number?

Now look at something very fascinating. Underneath the numbers 0, 1, 2, 3… in the final row of the tree write the number in binary (see Activity 1).

Look closely at the tree. If no=0 and yes=1, what do you see?

In the number guessing game we try to choose questions so that the sequence of answers works out to represent the number in exactly this way.

Design your own decision tree for guessing numbers between 0 and 15.

Extra for experts: What kind of tree would you use to guess someone's age? What about a tree to guess which letter is next in a sentence?

What's it all about?

A celebrated American mathematician (and juggler, and unicyclist) called Claude Shannon did a lot of experiments with this game. He measured the amount of information in bits—each yes/no answer is equivalent to a 1/0 bit. He found that the amount of "information" contained in a message depends on what you already know. Sometimes we can ask a question that eliminates the need to ask a lot of other questions. In this case the information content of the message is low. For example, the information in a single toss of a coin is normally one bit: heads or tails. But if the coin happens to be a biased one that turns up heads nine times out of ten, then the information is no longer one bit—believe it or not, it's less. How can you find out what a coin toss was with less than one yes/no question? Simple—just use questions like "are the next *two* coin tosses both heads?" For a sequence of tosses with the biased coin, the answer to this will be "yes" about 80%, of the time. On the 20% of occasions where the answer is "no," you will have to ask two further questions. But on average you will be asking less than one question per coin toss!

Shannon called the information content of a message "entropy". Entropy depends not only on the *number* of possible outcomes—in the case of a coin toss, two—but also on the *probability* of it happening. Improbable events, or surprising information, need a lot more questions to guess the message because they tell us more information we didn't already know—just like the situation of taking a helicopter to school.

The entropy of a message is very important to computer scientists. You cannot compress a message to occupy less space than its entropy, and the best compression systems are equivalent to a guessing game. Since a computer program is making the 'guesses', the list of questions can be reproduced later, so as long as the answers (bits) are stored, we can reconstruct the information! The best compression systems can reduce text files to about a quarter of their original size—a big saving on storage space!

The guessing method can also be used to build a computer interface that predicts what the user is going to type next! This can be very useful for physically disabled people who find it difficult to type. The computer suggests what it thinks they are likely to type next, and they just indicate what they want. A good system needs an average of only two yes/no answers per character, and can be of great assistance to someone who has difficulty making the fine movements needed to control a mouse or keyboard. This sort of system is also used in a different form to ‘type’ text on some cellphones.

Solutions and hints

The answer to a single yes/no question corresponds to exactly one bit of information—whether it is a simple question like “Is it more than 50?” or a more complex one like “Is it between 20 and 60?”

In the number-guessing game, if the questions are chosen in a certain way, the sequence of answers is just the binary representation of the number. Three is 011 in binary and is represented by the answers “No, yes, yes” in the decision tree, which is the same if we write no for 0 and yes for 1.

A tree you would use for someone’s age might be biased towards smaller numbers.

The decision about the letters in a sentence might depend upon what the previous letter was.

Part II

Putting Computers to Work—*Algorithms*

Putting Computers to Work

Computers operate by following a list of instructions set for them. These instructions enable them to sort, find and send information. To do these things as quickly as possible, you need good methods for finding things in large collections of data, and for sending information through networks.

An *algorithm* is a set of instructions for completing a task. The idea of an algorithm is central to computer science. Algorithms are how we get computers to solve problems. Some algorithms are faster than others, and many of the algorithms that have been discovered have made it possible to solve problems that previously took an infeasible length of time—for example, finding millions of digits in pi, or all pages that contain your name on the World-Wide Web, or finding out the best way to pack parcels into a container, or finding out whether or not very large (100-digit) numbers are prime.

The word "algorithm" is derived from the name of Mohammed ibn Musa Al-Khowarizmi—Mohammed, son of Moses, from Khowarizm—who joined an academic centre known as the House of Wisdom in Baghdad around 800AD. His works transmitted the Hindu art of reckoning to the Arabs, and thence to Europe. When they were translated into Latin in 1120AD, the first words were "Dixit Algorismi"—"thus said Algorismi".

Activity 6

Battleships—*Searching Algorithms*

Summary

Computers are often required to find information in large collections of data. They need to develop quick and efficient ways of doing this. This activity demonstrates three different search methods: linear searching, binary searching and hashing.

Curriculum Links

- ✓ Mathematics: Number – Exploring numbers: Greater than, less than and equal to
- ✓ Mathematics: Geometry – Exploring shape and space: Co-ordinates
- ✓ Computing: Algorithms

Skills

- ✓ Logical reasoning

Ages

- ✓ 9 years and up

Materials

Each student will need:

- ✓ Copy of battleships games
  - 1A, 1B for game 1
  - 2A, 2B for game 2
  - 3A, 3B for game 3

- ✓ You may also need a few copies of the supplementary game sheets, 1A', 1B', 2A', 2B', 3A', 3B'.

Battleships

Introductory Activity

1. Choose about 15 students to line up at the front of the classroom. Give each student a card with a number on it (in random order). Keep the numbers hidden from the rest of the class.

2. Give another student a container with four or five sweets in it. Their job is to find a given number. They can "pay" to look at a particular card. If they find the correct number before using all their sweets, they get to keep the rest.

3. Repeat if you wish to.

4. Now shuffle the cards and give them out again. This time, have the students sort themselves into ascending order. The searching process is repeated.

   If the numbers are sorted, a sensible strategy is to use just one "payment" to eliminate half the students by having the middle student reveal their card. By repeating this process they should be able to find the number using only three sweets. The increased efficiency will be obvious.

Activity

The students can get a feel for how a computer searches by playing the battleship game. As they play the game, get them to think about the strategies they are using to locate the ships.

Battleships—A *Linear* Searching Game

Read the following instructions to the students

1. Organise yourselves into pairs. One of you has sheet 1A, the other sheet 1B. Don't show your sheet to your partner!

2. Both of you circle one battleship on the top line of your game sheet and tell your partner its number.

3. Now take turns to guess where your partner's ship is. (You say the letter name of a ship and your partner tells you the number of the ship at that letter.)

4. How many shots does it take to locate your partner's ship? This is your score for the game.

   (Sheets 1A' and 1B' are extras provided for students who would like to play more games or who "inadvertently" see their partner's sheet. Sheets 2A', 2B' and 3A', 3B' are for the later games.)

Follow Up Discussion

1. What were the scores?

2. What would be the minimum and maximum scores possible? (They are 1 and 26 respectively, assuming that the students don't shoot at the same ship twice. This method is called 'linear search', because it involves going through all the positions, one by one.)

Battleships—A *Binary* Searching Game

Instructions

The instructions for this version of the game are the same as for the previous game but the numbers on the ships are now in ascending order. Explain this to the students before they start.

1. Organise yourselves into pairs. One of you has sheet 2A, the other sheet 2B. **Don't** show your sheet to your partner!

2. Both of you circle one battleship on the top line of your game sheet and tell your partner its number.

3. Now take turns to guess where your partner's ship is. (You say the letter name of a ship and your partner tells you the number of the ship at that letter.)

4. How many shots does it take to locate your partner's ship? This is your score for the game.

Follow Up Discussion

1. What were the scores?

2. What strategy did the low scorers use?

3. Which ship should you choose first? (The one in the middle tells you which half of the line the chosen ship must be in.) Which location would you choose next? (Again the best strategy is always to choose the middle ship of the section that must have the selected ship.)

4. If this strategy is applied how many shots will it take to find a ship? (Five at most).

This method is called 'binary search', because it divides the problem into two parts.

Battleships—A *Hashing* Search Game

Instructions

1. Each take a sheet as in the previous games and tell your partner the number of your chosen ship.

2. In this game you can find out which column (0 to 9) the ship is in. You simply add together the digits of the ship's number. The last digit of the sum is the column the ship is in. For example, to locate a ship numbered 2345, add the digits 2+3+4+5, giving 14. The last digit of the sum is 4, so that ship must be in column 4. Once you know the column you need to guess which of the ships in that column is the desired one. This technique is called 'hashing', because the digits are being squashed up ("hashed") together.

3. Now play the game using this new searching strategy. You may like to play more than one game using the same sheet—just choose from different columns.

   (Note that, unlike the other games, the spare sheets 3A' and 3B' must be used as a pair, because the pattern of ships in columns must correspond.)

Follow Up Discussion

1. Collect and discuss scores as before.

2. Which ships are very quick to find? (The ones that are alone in their column.) Which ships may be harder to find? (The ones whose columns contain lots of other ships.)

3. Which of the three searching processes is fastest? Why?

   What are the advantages of each of the three different ways of searching? (The second strategy is faster than the first, but the first one doesn't require the ships to be sorted into order. The third strategy is usually faster than the other two, but it is possible, by chance, for it to be very slow. In the worst case, if all the ships end up in the same column, it is just as slow as the first strategy.)

Extension Activities

1. Have the students make up their own games using the three formats. For the second game they must put the numbers in ascending order. Ask how they might make the Hashing Game very hard. (The hardest game is when all the ships are in the same column.) How could you make it as easy as possible? (You should try to get the same number of ships into each column.)

2. What would happen if the ship being sought wasn't there? (In the Linear Search game it would take 26 shots to show this. In the Binary Search game you would need five shots to prove this. When using the Hash System it would depend on how many ships appeared in the relevant column.)

3. Using the Binary Search strategy how many shots would be required if there were a hundred locations (about six shots), a thousand locations (about nine), or a million (about nineteen)? (Notice that the number of shots increases very slowly compared to the number of ships. One extra shot is required each time the size doubles, so it is proportional to the logarithm of the number of ships.)

My Ships

Number of Shots Used:

9058	7169	3214	5891	4917	2767	4715	674	8088	1790	8949	13	3014
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
8311	7621	3542	9264	450	8562	4191	4932	9462	8423	5063	6221	2244
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

Your Ships

Number of Shots Used:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

1A

My Ships

Number of Shots Used:

1630	9263	4127	405	4429	7113	3176	4015	7976	88	3465	1571	8625
A	B	C	D	E	F	G	H	I	J	K	L	M
2587	7187	5258	8020	1919	141	4414	3056	9118	717	7021	3076	3336
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Your Ships

Number of Shots Used:

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

1B

My Ships

Number of Shots Used:

163	445	622	1410	1704	2169	2680	2713	2734	3972	4208	4871	5031
A	B	C	D	E	F	G	H	I	J	K	L	M
5283	5704	6025	6801	7440	7542	7956	8094	8672	9137	9224	9508	9663
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Your Ships

Number of Shots Used:

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

2A

My Ships

Number of Shots Used:

33	183	730	911	1927	1943	2200	2215	3451	3519	4055	5548	5655
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
5785	5897	5905	6118	6296	6625	6771	6831	7151	7806	8077	9024	9328
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

Your Ships

Number of Shots Used:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

2B

My Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A 9047 B 1829	C 3080 D 9994		E 5125 F 1480 G 8212	H 8051 I 1481 J 4712 K 6422	L 7116 M 8944 N 4128	O 6000 P 7432 Q 4110	R 9891 S 1989 T 2050 U 8199	V 4392	W 1062 X 2106 Y 5842 Z 7057

Your Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A B C D	E F G	H I J	K	L M N		O P Q	R S T U	V W X	Y Z

3A

My Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A 9308	E 6519	H 1524	K 4135	L 9050		O 4200	R 3121	V 2385	Y 1990
B 1478	F 2469	I 8112		M 1265		P 7153	S 9503	W 5832	Z 2502
C 8417	G 5105	J 2000		N 5711		Q 6028	T 1114	X 1917	
D 9434							U 7019		

Your Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A	C		E	H	L	O	R	V	W
B	D		F	I	M	P	S		X
			G	J	N	Q	T		Y
				K			U		Z

3B

My Ships

Number of Shots Used:

6123	1519	9024	5164	2038	2142	7156	9974	9375	7104	1004	1023	5108
A	B	C	D	E	F	G	H	I	J	K	L	M
1884	3541	5251	4840	3289	3654	2480	5602	8965	4053	2405	2304	1959
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Your Ships

Number of Shots Used:

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

1A′

My Ships

Number of Shots Used:

2387	9003	3951	5695	1284	4761	7118	1196	1741	3791	3405	3132	6682
A	B	C	D	E	F	G	H	I	J	K	L	M
9493	9864	7359	1250	7036	2916	7562	9299	8910	6713	5173	8617	4222
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Your Ships

Number of Shots Used:

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

1B′

My Ships

Number of Shots Used:

28	326	943	1321	1896	2346	2430	2929	3106	3417	4128	4717	4915
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
5123	5615	6100	7015	7120	7695	7812	8103	8719	9020	9608	9713	9911
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

Your Ships

Number of Shots Used:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

2A′

My Ships

Number of Shots Used:

56	194	306	1024	1510	1807	2500	2812	3011	3902	4178	5902	5915
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
6102	6526	6818	7020	7155	7913	8016	8230	8599	8902	9090	9526	9812
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

Your Ships

Number of Shots Used:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

2B′

3A′

My Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A 1982	C 6113		E 9121	H 5009	L 1248	O 2004	R 9369	V 3285	W 9172
B 7841	D 1055		F 1011	I 2651	M 1716	P 5173	S 1321		X 2052
			G 2984	J 1751	N 2148	Q 2806	T 3004		Y 6012
				K 4848			U 7190		Z 7525

Your Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A	E	H	K	L		O	R	V	Y
B	F	I		M		P	S	W	Z
C	G	J		N		Q	T	X	
D							U		

My Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A 8615	E 1361	H 7726	K 3000	L 1814		O 9656	R 6993	V 8208	Y 2917
B 7003	F 7644	I 9003		M 2002		P 4002	S 3121	W 9423	Z 4122
C 1991	G 5600	J 5557		N 8844		Q 1221	T 4300	X 4176	
D 6211							U 1907		

Your Ships

Number of Shots Used:

0	1	2	3	4	5	6	7	8	9
A	C		E	H	L	O	R	V	W
B	D		F	I	M	P	S		X
			G	J	N	Q	T		Y
				K			U		Z

3B′

What’s it all about?

Computers store a lot of information, and they need to be able to sift through it quickly. One of the biggest search problems in the world is faced by Internet search engines, which must search billions of web pages in a fraction of a second. The data that a computer is asked to look up, such as a word, a bar code number or an author’s name, is called a *search key*.

Computers can process information very quickly, and you might think that to find something they should just start at the beginning of their storage and keep looking until the desired information is found. This is what we did in the Linear Searching Game. But this method is very slow—even for computers. For example, suppose a supermarket has 10,000 different products on its shelves. When a bar code is scanned at a checkout, the computer must look through up to 10,000 numbers to find the product name and price. Even if it takes only one thousandth of a second to check each code, ten seconds would be needed to go through the whole list. Imagine how long it would take to check out the groceries for a family!

A better strategy is *binary search*. In this method, the numbers are sorted into order. Checking the middle item of the list will identify which half the search key is in. The process is repeated until the item is found. Returning to the supermarket example, the 10,000 items can now be searched with fourteen probes, which might take two hundredths of a second—hardly noticeable.

A third strategy for finding data is called *hashing*. Here the search key is manipulated to indicate exactly where to find the information. For example, if the search key is a telephone number, you could add up all the digits in the number and take the remainder when divided by 11. In this respect, a hash key is a little like the check digits discussed in Activity 4—a small piece of data whose value depends on the other data being processed. Usually the computer will find what it is looking for straight away. There is a small chance that several keys end up in the same location in which case the computer will need to search through them until it finds the one it is seeking.

Computer programmers usually use some version of the hashing strategy for searching, unless it is important to keep the data in order, or unless an occasional slow response is unacceptable.

Activity 7

Lightest and Heaviest—*Sorting Algorithms*

Summary

Computers are often used to put lists into some sort of order, for example names into alphabetical order, appointments or e-mail by date, or items in numerical order. Sorting lists helps us find things quickly, and also makes extreme values easy to see. If you sort the marks for a class test into numeric order, the lowest and highest marks become obvious.

If you use the wrong method, it can take a long time to sort a large list into order, even on a fast computer. Fortunately, several fast methods are known for sorting. In this activity students will discover different methods for sorting, and see how a clever method can perform the task much more quickly than a simple one.

Curriculum links

- ✓ Mathematics: Measurement – Carrying out practical weighing tasks.
- ✓ Computing: Algorithms

Skills

- ✓ Using balance scales
- ✓ Ordering
- ✓ Comparing

Ages

- ✓ 8 and up

Materials

Each group of students will need:

- ✓ Sets of 8 containers of the same size but different weights (e.g. milk cartons or film canisters filled with sand)
- ✓ Balance scales
- ✓ Worksheet Activity: Sorting weights (page 74)
- ✓ Worksheet Activity: Divide and conquer (page 75)

Lightest and Heaviest

Discussion

Computers often have to sort lists of things into order. Brainstorm all the places where putting things into order is important. What would happen if these things were not in order?

Computers usually only compare two values at once. The activity on the next page uses this restriction to give students an idea of what this is like.

Activity

1. Divide the students into groups.
2. Each group will need a copy of the activity sheet on page 74, and its own weights and scales.
3. Have the students do the activity, then discuss the result.

Worksheet Activity: Sorting Weights

Aim: To find the best method of sorting a group of unknown weights into order.

You will need: Sand or water, 8 identical containers, a set of balance scales

What to do:

1. Fill each container with a different amount of sand or water. Seal tightly.

2. Mix them up so that you no longer know the order of the weights.

3. Find the lightest weight. What is the easiest way of doing this?

   Note: You are only allowed to use the scales to find out how heavy each container is. Only two weights can be compared at a time.

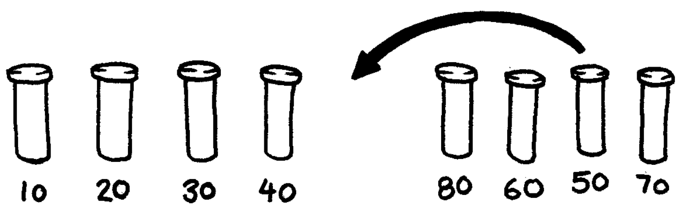
4. Choose 3 weights at random and sort them into order from lightest to heaviest using only the scales. How did you do this? What is the minimum number of comparisons you can make? Why?

5. Now sort all of the objects into order from lightest to heaviest.

When you think you have finished, check your ordering by re-weighing each pair of objects standing together.

Selection Sort

One method a computer might use is called *selection sort*. This is how selection sort works. First find the lightest weight in the set and put it to one side. Next, find the lightest of the weights that are left, and remove it. Repeat this until all the weights have been removed.



Count how many comparisons you made.

Extra for Experts: Show how you can calculate mathematically how many comparisons you need to make to sort 8 objects into order. What about 9 objects? 20?

Worksheet Activity: Divide and Conquer

Quicksort

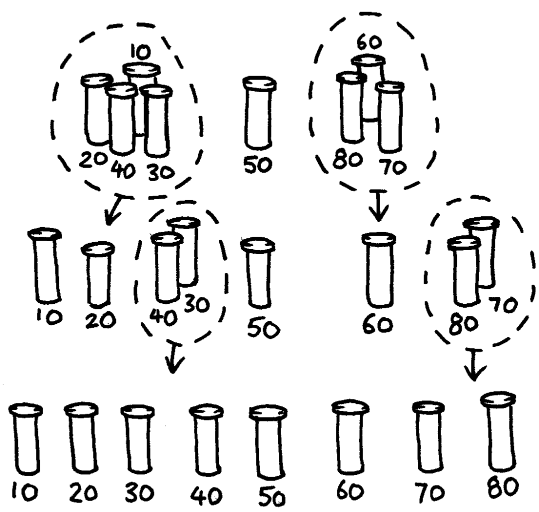
Quicksort is a lot faster than selection sort, particularly for larger lists. In fact, it is one of the best methods known. This is how quicksort works.

Choose one of the objects at random, and place it on one side of the balance scales.

Now compare each of the remaining objects with it. Put those that are lighter on the left, the chosen object in the middle, and the heavier ones on the right. (By chance you may end up with many more objects on one side than on the other.)

Choose one of the groups and repeat this procedure. Do the same for the other group. Remember to keep the one you know in the centre.

Keep repeating this procedure on the remaining groups until no group has more than one object in it. Once all the groups have been divided down to single objects, the objects will be in order from lightest to heaviest.



How many comparisons did this process take?

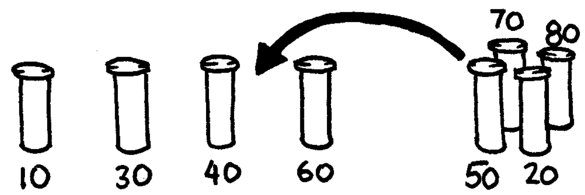
You should find that quicksort is a more efficient method than selection sort unless you happen to have chosen the lightest or heaviest weight to begin with. If you were lucky enough to have chosen the middle weight, you should have taken only 14 comparisons, compared with the 28 for selection sort. At any rate the quicksort method will never be any worse than selection sort and may be much better!

Extra for Experts: If quicksort accidentally always chose the lightest object, how many comparisons would it use?

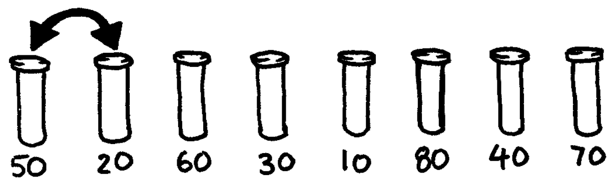
Variations and extensions

Many different methods for sorting have been invented. You could try sorting your weights using these:

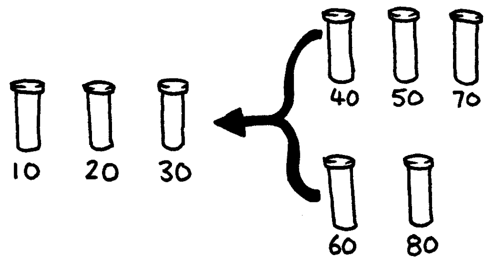
Insertion sort works by removing each object from an unsorted group and inserting it into its correct position in a growing list (see picture below). With each insertion the group of unsorted objects shrinks and the sorted list grows, until eventually the whole list is sorted. Card players often use this method to sort a hand into order.



Bubble sort involves going through the list again and again, swapping any objects side-by-side that are in the wrong order. The list is sorted when no swaps occur during a pass through the list. This method is not very efficient, but some people find it easier to understand than the others.



Mergesort is another method that uses 'divide and conquer' to sort a list of items. First, the list is divided at random into two lists of equal size (or nearly equal if there are an odd number of items). Each of the two half-size lists is sorted, and the two lists are merged together. Merging two sorted lists is easy—you repeatedly remove the smaller of the two items at the front of the two lists. In the figure below, the 40 and 60-gram weights are at the front of the lists, so the next item to add is the 40-gram weight. How do you sort the smaller lists? Simple—just use mergesort! Eventually, all the lists will be cut down into individual items, so you don't need to worry about knowing when to stop.



What's it all about?

Information is much easier to find in a sorted list. Telephone directories, dictionaries and book indexes all use alphabetical order, and life would be far more difficult if they didn't. If a list of numbers (such as a list of expenses) is sorted into order, the extreme cases are easy to see because they are at the beginning and end of the list. Duplicates are also easy to find, because they end up together.

Computers spend a lot of their time sorting things into order, so computer scientists have to find fast and efficient ways of doing this. Some of the slower methods such as insertion sort, selection sort and bubble sort can be useful in special situations, but the fast ones such as quicksort and mergesort are usually used because they are much faster on large lists – for example, for 100,000 items, quicksort is typically about 2,000 times as fast as selection sort, and for 1,000,000 items, it is about 20,000 times as fast. Computers often have to deal with a million items (lots of websites have millions of customers, and even a single photo taken on a cheap camera has over a million pixels); the difference between the two algorithms is the difference between taking 1 second to process the items, and taking over 5 hours to do exactly the same task. Not only would the delay be intolerable, but it will have used 20,000 times as much power (which not only impacts the environment, but also reduces battery life in portable devices), so choosing the right algorithm has serious consequences.

Quicksort uses an approach called *Divide and Conquer*. In quicksort, you keep dividing a list into smaller parts, and then perform a quicksort on each of the parts. The list is divided repeatedly until it is small enough to conquer. For quicksort, the lists are divided until they contain only one item. It is trivial to sort one item into order! Although this seems very involved, in practice it is dramatically faster than other methods. This is an example of of a powerful idea called *Recursion* where an algorithm uses itself to solve a problem – this sounds weird but it can work very well.

Solutions and hints

1. The best way to find the lightest weight is to go through each object in turn, keeping track of the lightest one so far. That is, compare two objects, and keep the lighter one. Now compare that with another, keeping the lighter from the comparison. Repeat until all the objects have been used.

2. Compare the weights on the balance scales. This can easily be done with three comparisons, and sometimes just two will suffice—if the students realize that the comparison operator is transitive (that is, if A is lighter than B and B is lighter than C, then A must be lighter than C).

   Experts:

   Here is a short cut for adding up the number of comparisons that selection sort makes.

   To find the minimum of two objects you need one comparison, three needs two, four needs three, and so on. To sort eight objects using selection sort takes 7 comparisons to find the first one, six to find the next, five to find the next and so on. That gives us:

   7 + 6 + 5 + 4 + 3 + 2 + 1 = 28 comparisons.

   n objects will take 1 + 2 + 3 + 4 +... + n – 1 comparisons to sort.

   Adding up these numbers is easy if we regroup them.

   For example, to add up the numbers 1 + 2 + 3 + ... + 20, regroup them as

   (1 + 20) + (2 + 19) + (3 + 18) + (4 + 17) + (5 + 16) +

   (6 + 15) + (7 + 14) + (8 + 13) + (9 + 12) + (10 + 11)

   = 21 × 10

   = 210

   In general, the sum 1 + 2 + 3 + 4 ... + n – 1 = $n(n – 1)/2$.

Activity 8

Beat the Clock—*Sorting Networks*

Summary

Even though computers are fast, there is a limit to how quickly they can solve problems. One way to speed things up is to use several computers to solve different parts of a problem. In this activity we use sorting networks which do several sorting comparisons at the same time.

Curriculum Links

- ✓ Mathematics: Number – Exploring number: Greater than, less than

Skills

- ✓ Comparing
- ✓ Ordering
- ✓ Developing algorithms
- ✓ Co-operative problem solving

Ages

- ✓ 7 years and up

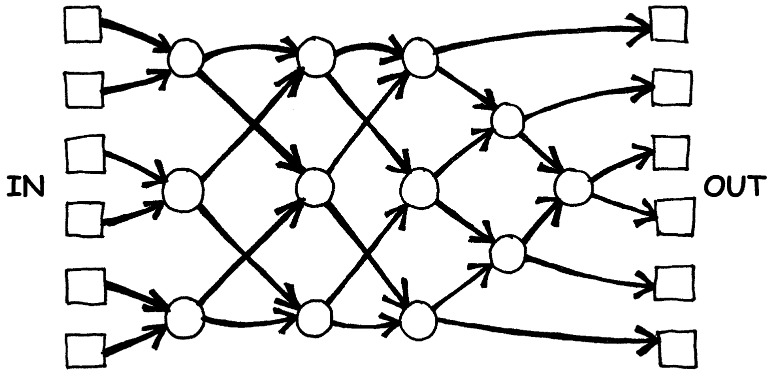
Materials

This is an outdoor group activity.

- ✓ Chalk
- ✓ Two sets of six cards.
  Copy Photocopy Master: Sorting networks (page 83) onto card and cut out
- ✓ Stopwatch

Sorting Networks

Prior to the activity use chalk to mark out this network on a court.



Instructions for Students

This activity will show you how computers sort random numbers into order using a thing called a sorting network.

1. Organise yourselves into groups of six. Only one team uses the network at a time.

2. Each team member takes a numbered card.

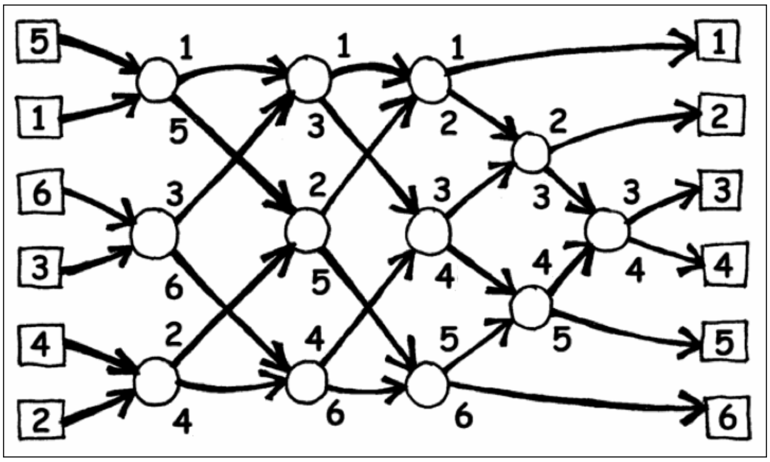
3. Each member stands in a square on the left hand (IN) side of the court. Your numbers should be in jumbled order.

4. You move along the lines marked, and when you reach a circle **you must wait for someone else to arrive.**

5. When another team member arrives in your circle compare your cards. The person with the smaller number takes the exit to their left. If you have the higher number on your card take the right exit.

6. Are you in the right order when you get to the other end of the court?

If a team makes an error the students must start again. Check that you have understood the operation of a node (circle) in the network, where the smaller value goes left and the other goes right. For example:


5
1
6
3
4
2
1
5
3
6
2
4
1
3
2
5
4
6
1
2
3
4
5
6
2
3
4
5
3
4
1
2
3
4
5
6

Photocopy Master: Sorting networks

1	2
3	4
5	6

156	221
289	314
422	499

Variations

1. When the students are familiar with the activity use a stopwatch to time how long each team takes to get through the network.

2. Use cards with larger numbers (e.g. the three-digit ones in the photocopy master).

3. Make up cards with even larger numbers that will take some effort to compare, or use words and compare them alphabetically.

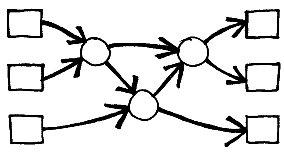
4. This can be used as an exercise for other subjects e.g. in music you can compare notes printed on cards, and sort them from lowest to highest, or shortest to longest.

Extension Activities

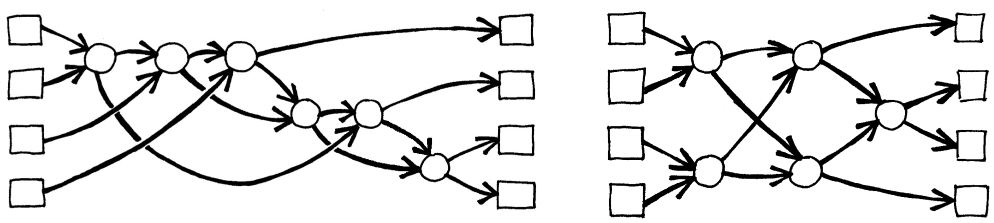
1. What happens if the smaller one goes right instead of left and vice versa? (The numbers will be sorted in reverse order.)

   Does it work if the network is used backwards? (It will not necessarily work, and the students should be able to find an example of an input that comes out in the wrong order.)

2. Try to design smaller or larger networks. For example, here is a network that sorts just three numbers. The students should try to come up with this on their own.

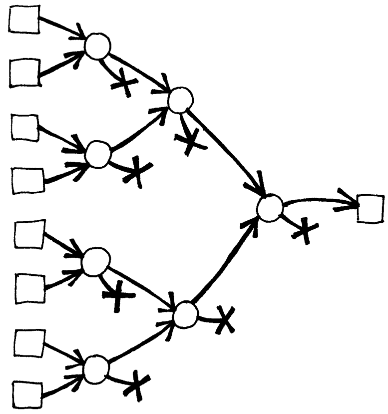


3. Below are two different networks that will sort four inputs. Which is the faster? (The second one is. Whereas the first requires all comparisons to be done serially, one after the other, the second has some being performed at the same time. The first network is an example of serial processing, whereas the second uses parallel processing to run faster.)



4. Try to make a larger sorting network.

5. Networks can also be used to find the minimum or maximum value of the inputs. For example, here is a network with eight inputs, and the single output will contain the minimum of the inputs (the other values will be left at the dead ends in the network).



6. What processes from everyday life can or can’t be accelerated using parallelism? For example, cooking a meal would be a lot slower using only one cooking element, because the items would have to be cooked one after another. What jobs can be completed faster by employing more people? What jobs can’t?

What's it all about?

As we use computers more and more we want them to process information as quickly as possible.

One way to increase the speed of a computer is to write programs that use fewer computational steps (as shown in Activities 6 and 7).

Another way to solve problems faster is to have several computers work on different parts of the same task at the same time. For example, in the six-number sorting network, although a total of 12 comparisons are used to sort the numbers, up to three comparisons are performed simultaneously. This means that the time required will be that needed for just 5 comparison steps. This parallel network sorts the list more than twice as quickly as a system that can only perform one comparison at a time.

Not all tasks can be completed faster by using parallel computation. As an analogy, imagine one person digging a ditch ten metres long. If ten people each dug one metre of the ditch the task would be completed much faster. However, the same strategy could not be applied to a ditch ten metres deep—the second metre is not accessible until the first metre has been dug. Computer Scientists are still actively trying to find the best ways to break problems up so that they can be solved by computers working in parallel.

Activity 9

The Muddy City—*Minimal Spanning Trees*

Summary

Our society is linked by many networks: telephone networks, utility supply networks, computer networks, and road networks. For a particular network there is usually some choice about where the roads, cables, or radio links can be placed. We need to find ways of efficiently linking objects in a network.

Curriculum Links

- ✓ Mathematics: Geometry – Exploring shape and space: Finding the shortest paths around a map

Ages

- ✓ 9 and up

Skills

- ✓ Problem solving

Materials

Each student will need:

- ✓ Workshop Activity: The muddy city problem (page 89)
- ✓ Counters or squares of cardboard (approximately 40 per student)

The Muddy City

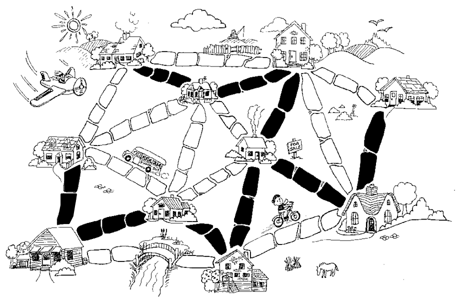
Introduction

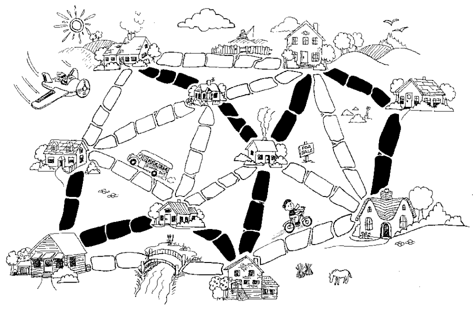
This activity will show you how computers are used to find the best solutions for real-life problems such as how to link power lines between houses. Have the students use the worksheet on page 89, which explains the 'Muddy City' problem.

Follow-up discussion

Share the solutions the students have found. What strategies did they use?

One good strategy to find the best solution is to start with an empty map, and gradually add counters until all of the houses are linked, adding the paths in increasing order of length, but not linking houses that are already linked. Different solutions are found if you change the order in which paths of the same length are added. Two possible solutions are shown below.





Another strategy is to start with all of the paths paved, and then remove paths you don't need. This takes much more effort, however.

Where would you find networks in real life?

Computer scientists call the representations of these networks "graphs". Real networks can be represented by a graph to solve problems such as designing the best network of roads between local cities, or aeroplane flights around the country.

There are also many other algorithms that can be applied to graphs, such as finding the shortest distance between two points, or the shortest route that visits all the points.

Worksheet Activity: The Muddy City Problem

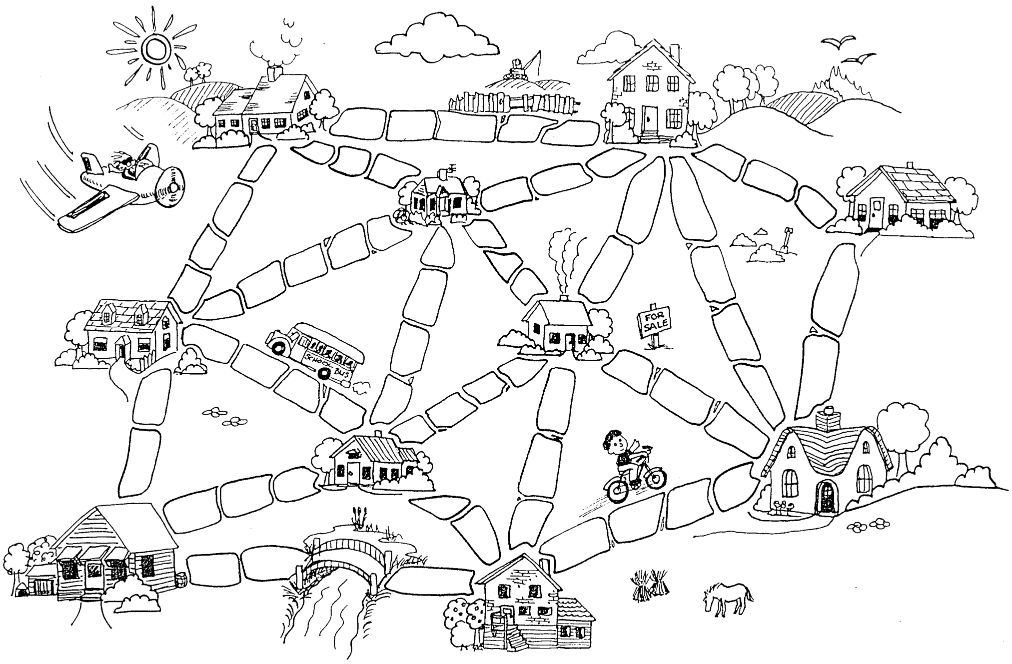
Once upon a time there was a city that had no roads. Getting around the city was particularly difficult after rainstorms because the ground became very muddy—cars got stuck in the mud and people got their boots dirty. The mayor of the city decided that some of the streets must be paved, but didn't want to spend more money than necessary because the city also wanted to build a swimming pool. The mayor therefore specified two conditions:

1. Enough streets must be paved so that it is possible for everyone to travel from their house to anyone else's house only along paved roads, and

2. The paving should cost as little as possible.

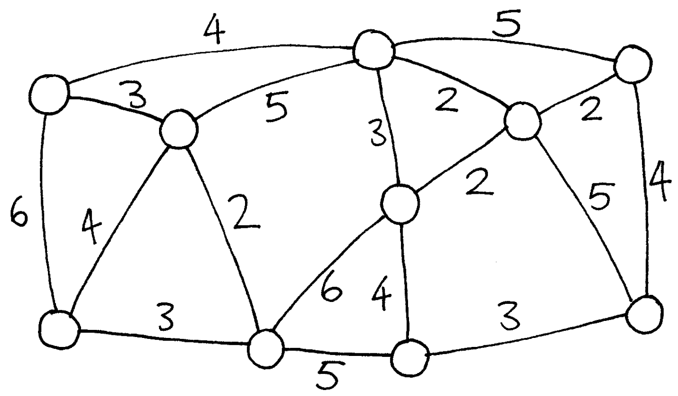
Here is the layout of the city. The number of paving stones between each house represents the cost of paving that route. Find the best route that connects all the houses, but uses as few counters (paving stones) as possible.

What strategies did you use to solve the problem?



Variations and extensions

Here is another way of representing the cities and roads:



The houses are represented by circles, the muddy roads by lines, and the length of a road is given by the number beside the line.

Computer scientists and mathematicians often use this sort of diagram to represent these problems. They call it a *graph*. This may be confusing at first because “graph” is sometimes used in statistics to mean a chart displaying numerical data, such as a bar graph, but the graphs that computer scientists use are not related to these. The lengths do not have to be drawn to scale.

Make up some of your own muddy city problems and try them out on your friends.

Can you find out a rule to describe how many roads or connections are needed for a best solution? Does it depend on how many houses there are in the city?

What’s it all about?

Suppose you are designing how a utility such as electricity, gas, or water should be delivered to a new community. A network of wires or pipes is needed to connect all the houses to the utility company. Every house needs to be connected into the network at some point, but the route taken by the utility to get to the house doesn’t really matter, just so long as a route exists.

The task of designing a network with a minimal total length is called the *minimal spanning tree* problem.

Minimal spanning trees aren’t only useful in gas and power networks; they also help us solve problems in computer networks, telephone networks, oil pipelines, and airline routes. However, when deciding the best routes for people to travel, you do have to take into account how convenient the trip will be for the traveller as well as how much it will cost. No-one wants to spend hours in an aeroplane taking the long way round to a new country just because it is cheaper. The muddy city algorithm may not be much use for these networks, because it simply minimizes the *total* length of the roads or flight paths.

Minimal spanning trees are also useful as one of the steps for solving other problems on graphs, such as the “travelling salesperson problem” which tries to find the shortest route that visits every point in the network.

There are efficient algorithms (methods) for solving minimal spanning tree problems. A simple method that gives an optimal solution is to start with no connections, and add them in increasing order of size, only adding connections that join up part of the network that wasn’t previously connected. This is called Kruskal’s algorithm after J.B. Kruskal, who published it in 1956.

For many problems on graphs, including the “travelling salesperson problem”, computer scientists are yet to find fast enough methods that find the best possible solution.

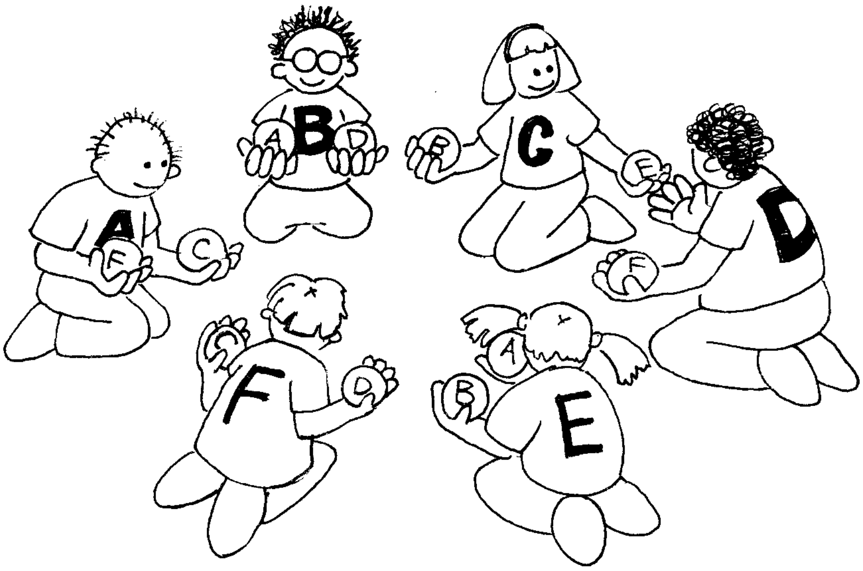
Solutions and hints

Variations and extensions (page 89)

How many roads or connections are needed if there are *n* houses in the city? It turns out that an optimal solution will always have exactly n–1 connections in it, as this is always sufficient to link up the *n* houses, and adding one more would create unnecessary alternative routes between houses.

Activity 10

The Orange Game—*Routing and Deadlock in Networks*



Summary

When you have a lot of people using one resource (such as cars using roads, or messages getting through the Internet), there is the possibility of "deadlock". A way of working co-operatively is needed to avoid this happening.

Curriculum Links

- ✓ Mathematics: Developing logic and reasoning

Skills

- ✓ Co-operative problem solving
- ✓ Logical reasoning

Ages

- ✓ 9 years and up

Materials

Each student will need:

- ✓ Two oranges or tennis balls labeled with the same letter, or two pieces of fruit each (artificial fruit is best)
- ✓ Name tag or sticker showing their letter, or a coloured hat, badge or top to match their fruit

The Orange Game

Introduction

This is a co-operative problem solving game. The aim is for each person to end up holding the oranges labelled with their own letter.

1. Groups of five or more students sit in a circle.

2. The students are labelled with a letter of the alphabet (using name tags or stickers), or each is allocated a colour (perhaps with a hat, or the colour of their cloths). If letters of the alphabet are used, there are two oranges with each student's letter on them, except for one student, who only has one corresponding orange to ensure that there is always an empty hand. If fruit is used, there are two pieces of fruit for each child e.g. a child with a yellow hat might have two bananas, and a child with a green hat may have two green apples, except one child has only one piece of fruit.

3. Distribute the oranges or fruit randomly to the students in the circle. Each student has two pieces, except for one student who has only one. (No student should have their corresponding orange or colour of fruit.)

4. The students pass the oranges/fruit around until each student gets the one labelled with their letter of the alphabet (or their colour). You must follow two rules:

   a) Only one piece of fruit may be held in a hand.

   b) A piece of fruit can only be passed to an empty hand of an immediate neighbour in the circle. (A student can pass either of their two oranges to their neighbour.)

Students will quickly find that if they are "greedy" (hold onto their own fruit as soon as they get them) then the group might not be able to attain its goal. It may be necessary to emphasize that individuals don't "win" the game, but that the puzzle is solved when everyone has the correct fruit.

Follow up Discussion

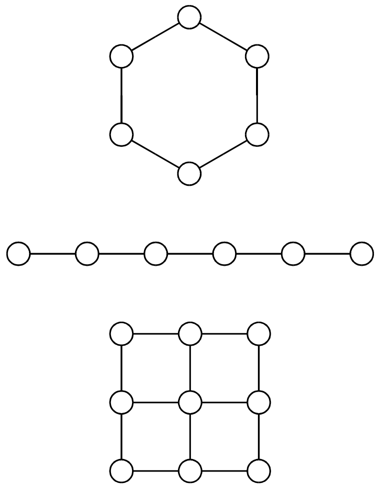
What strategies did the students use to solve the problem?

Where in real life have you experienced deadlock? (Some examples might be a traffic jam, getting players around bases in baseball, or trying to get a lot of people through a doorway at once.)

Extension Activities

Try the activity with a smaller or larger circle.

- Have the students come up with new rules.
- Carry out the activity without any talking.
- Try different configurations such as sitting in a line, or having more than two neighbours for some students. Some suggestions are shown here.



What's it all about?

Routing and deadlock are problems in many networks, such as road systems, telephone and computer systems. Engineers spend a lot of time figuring out how to solve these problems—and how to design networks that make the problems easier to solve.

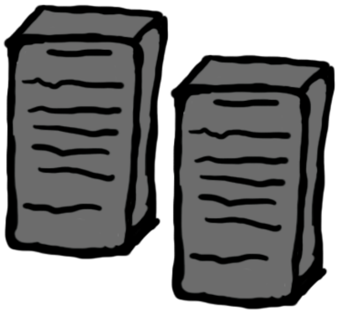
Routing, congestion and deadlock can present frustrating problems in many different networks. Just think of your favourite rush-hour traffic! It has happened several times in New York City that the traffic in the streets has become so congested that it deadlocks: no-one can move their car! Sometimes when the computers are "down" in businesses (such as banks) the problem is caused by a communication network deadlock. Designing networks so that routing is easy and efficient and congestion is minimized is a difficult problem faced by many kinds of engineers.

Sometimes more than one person wants the same data at the same time. If a piece of data (such as a customer's bank balance) is being updated, it is important to "lock" it during the update. If it is not locked, someone else could update it at the same time and the balance might be recorded incorrectly. However, if this locking is interfered with by the locking of another item, deadlock may occur.

One of the most exciting developments in computer design is the advent of parallel computing, where hundreds or thousands of PC-like processors are combined (in a network) to form a single powerful computer. Many problems like the Orange Game must be played on these networks continuously (but much faster!) in order for these parallel computers to work.

Activity 11

Tablets of Stone—*Network Communication Protocols*



Summary

Computers talk to each other over the internet via messages. However, the internet is not reliable and sometimes these messages get lost. There are certain bits of information we can add to messages to make sure they are sent. This information makes up a protocol.

Curriculum Links

- ✓ Mathematics: Developing logic and reasoning
- ✓ English: Communication, interpersonal listening

Skills

- ✓ Co-operative problem solving
- ✓ Logical reasoning

Ages

- ✓ 9 years and up

Materials

Each student will need:

- ✓ Many blank "Tablets"

Each messenger will need:

- ✓ A set of message action cards

The teacher will need:

- ✓ A timer

Tablets of Stone

Introduction

In this activity students consider how different methods of communication operate successfully. By looking at rules and procedures in place, students are introduced to communication protocols. By working through a role-play scenario, pupils test their own protocol operating in an unreliable environment similar to that found in packet switching on the Internet, specifically, TCP/ IP.

Preparation (30 minutes)

1. First gather the cards. You'll need to print out the action cards (below) and cut them up. These form the basis of the game.
2. Next, decide on some messages for student to send. It's important that they're *not* English sentences or anything that can be put back together by their structure. Something like "1LHC255HD(RLLS" would be a suitable message, or a phone number.
3. Print out copies of the "tablets'. Each tablet has places for six characters or numbers, so you cannot fit the whole message on one tablet. You will be need roughly 30 tablets per student, depending on how long you wish to run the game.

Note: The action cards are three types; delay, don't deliver, deliver. Adjusting the ratio between these will represent the quality of your messengers. More "deliver" cards means a more reliable messenger. More "delay" and "don't delivers means a less reliable network. These cards are analogous to a computer network/communication channel.

Playing the game

1. Split your class into pairs. It is crucial for the pairs to sit apart from one another where they cannot see or communicate with each other. Two rooms are ideal but sitting students on opposing sides of a classroom should suffice.
2. Give one of each pair a message to deliver to their partner.
3. Shuffle the Action Cards and choose a messenger. You could be the messenger or if you use a student if you have an odd number. You might need more than one messenger if you have a large class.
4. A student is now to write on their tablet and give it to the messenger. The tablet should at least say the name of the other person on it.

5. The messenger now picks the top action card, turns it over, reads it and uses it to decide what to do with the tablet.
6. Repeat steps 4 and 5 with each tablet

After 5 or so minutes of chaos and frustration, your students should realise that names alone are not good enough for a protocol. Stop the class and discuss this… what is the first issue they're having? Is it order? Perhaps it would be be best to use one of those 6 slots to put a tablet number in? This means there is less room for the actual data – what does this mean in terms of the number of tablets we have to use now?

After some more time, they might notice other problems, and these should also be discussed. Possible problems could be missing tablet, not knowing if the tablet was delivered, not knowing whether to resend a tablet. Solutions you could suggest would be a sending back acknowledgements and waiting to hear back for these before re-sending another – this means that the receiving student(s) also need blank tablets to send messages, and they will have to agree on what their 6-character responses mean before they play the game again.

You'll need at least two students for this game, but we recommend having as many as possible. If you have a large class, consider a few messengers. Once again, discuss this with your class… what happens if you have many messengers? What happens if you had one?

Deliver this tablet now	Deliver this message after the next one
Deliver this tablet now	Deliver this message after the next one
Deliver this tablet now	Deliver this message after the next one
Deliver this tablet now	Don’t deliver this message
Deliver this tablet now	Don’t deliver this message

To:

From:

To:

From:

To:

From:

To:

From:

To:

From:

To:

From:

To:

From:

To:

From:

Tablets of Stone

In an ancient city there are a number of very important Governors. These Governors decide how the city is run and make very important decisions. They each live in different houses all over the city.

The Governors often want to communicate, they need to send and receive messages all over the city. Governors are identified by their house number and they all have access to a group of messengers whose job it is to deliver the messages.

The only way to send messages is by writing them on large rectangular stone tablets, which the messengers carry to their destination. The stone tablets are of a fixed size and can only fit 6 pieces of information on them. One piece of information can be one letter or one number. Messages are often split over a number of tablets, and as these tablets are very heavy they can only be carried one at a time.

The messengers cannot be trusted to always deliver the message correctly as they are forgetful and lazy. They often stop for long breaks during working hours and even try to escape from the city.

The Governors want to find a way of making their communication reliable, they want to develop a set of rules that they will all follow. By doing this they can tell whether or not their message has been delivered and if the message was correct. The Governors have already decided that the destination should be written on the tablet.

In your groups your task is develop the rules that the Governors will use to communicate…

What's it all about?

On the internet, data is broken into packets for transportation. However, the channels in which these packets travel is not always reliable. Individual packets sometimes are damaged, lost or lose their ordering.

In Tablets of Stone, tablets are packets and their contents is data. Packets contain both data and *header* information. The size of the header information affects how much data can be transferred – so a balance has to be reached, as packets are of finite size.

Students will find that they will need to swap some of their data boxed for information such as packet number and total packets, or whether or not the packet is an acklnowledgement packet. Due to this information taking up data boxes, overall more packets will be needed.

Internet protocols such as TCP and UDP balance these factors to create reliable and efficient data transfer.

This activity is adapted from one available through the "Computing Science Inside" project (csi.dcs.gla.ac.uk).

Part III

Telling Computers What To Do—*Representing Procedures*

Telling Computers What To Do

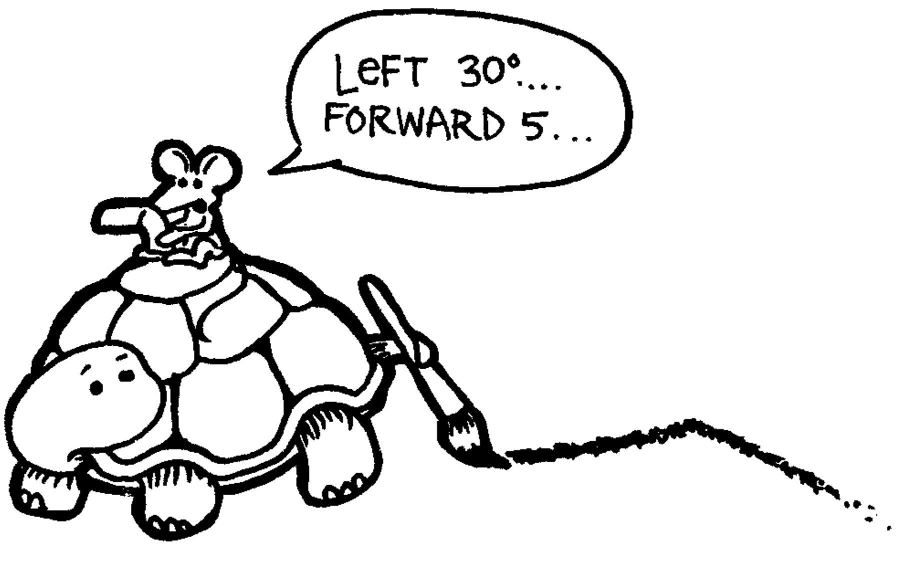
Computers follow instructions—millions of instructions every second. To tell a computer what to do, all you have to do is give it the right instructions. But that's not as easy as it sounds!

When we are given instructions we use common sense to interpret what is meant. If someone says "go through that door," they don't mean to actually smash through the door—they mean go through the door*way*, if necessary opening the door first! Computers are different. Indeed, when they are attached to mobile robots you need to be careful to take safety precautions to avoid them causing damage and danger by interpreting instructions literally—like trying to go through doors. Dealing with something that obeys instructions exactly, without "thinking," takes some getting used to.

The two activities in this section give us some idea of what it is like to communicate to literal-minded machines using a fixed set of instructions.

The first will teach us about a "machine" that computers use to recognise words, numbers or strings of symbols that the computer can work with. These "machines" are called finite-state automata.

The second activity introduces us to how we can communicate with computers. A good programmer has to learn how to tell the computer what to do using a fixed set of instructions that are interpreted literally. The list of instructions is the program. There are lots of different programming languages a programmer can choose to write these instructions in, but we will be using a simple language that can be used without a computer.



Activity 12

Treasure Hunt—*Finite-State Automata*

Summary

Computer programs often need to process a sequence of symbols such as letters or words in a document, or even the text of another computer program. Computer scientists often use a finite-state automaton to do this. A finite-state automaton (FSA) follows a set of instructions to see if the computer will recognise the word or string of symbols. We will be working with something equivalent to a FSA—treasure maps!

Curriculum Links

- ✓ Mathematics: Developing logic and reasoning—using words and symbols to describe and continue patterns
- ✓ Social Studies
- ✓ English

Skills

- ✓ Simple map reading
- ✓ Recognising patterns
- ✓ Logic
- ✓ Following instructions

Ages

- ✓ 9 and up

Materials

You will need:

- ✓ One set of island cards (the instructions must be kept hidden from those trying to draw the map!)
  Copy Photocopy Master: Island cards (page 114 onwards) and cut out. Fold along the dotted line and glue, so that the front of the card has the name of the island, and the back has the instructions.

Each student will need:

- ✓ Worksheet Activity: Find your way to the riches on Treasure Island (page 113)
- ✓ Pen or pencil

There are optional extension activities, for which each student will need:

- ✓ Worksheet Activity: Treasure islands (page 119)
- ✓ Worksheet Activity: The mysterious coin game (page 120)

Treasure Island

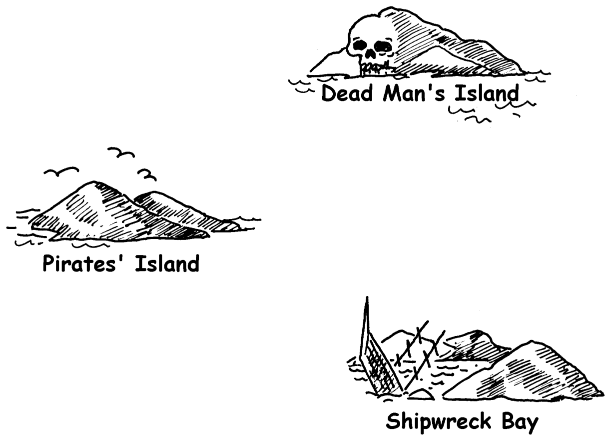
Introduction

Your goal is to find Treasure Island. Friendly pirate ships sail along a fixed set of routes between the islands in this part of the world, offering rides to travellers. Each island has two departing ships, A and B, which you can choose to travel on. You need to find the best route to Treasure Island. At each island you arrive at you may ask for either ship A or B (not both). The person at the island will tell you where your ship will take you to next, but the pirates don't have a map of all the islands available. Use your map to keep track of where you are going and which ship you have travelled on.

Demonstration

(**Note:** This is a different map from the actual activity.)

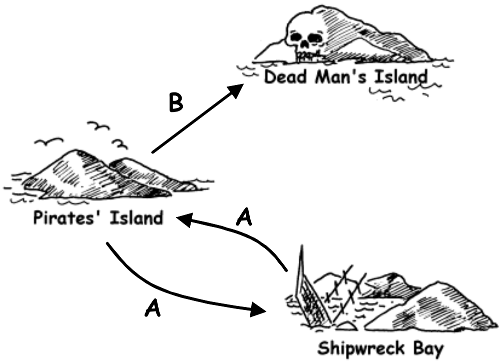
Using a board, draw a diagram of three islands as shown here:



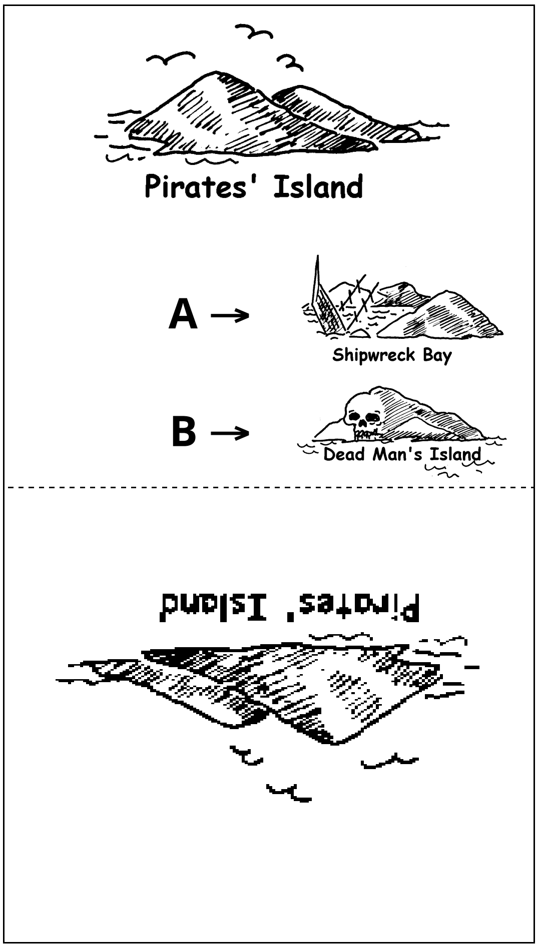
Copy the three cards on the next two pages, and have one student hold each card. Note that the routes on these cards are different from those in the main activity.

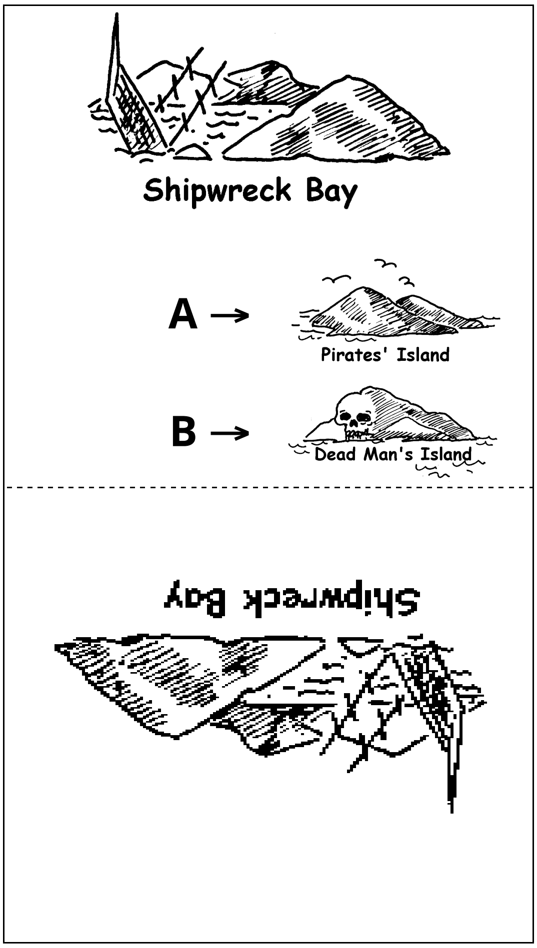
Starting at Pirates' Island ask for ship A. The student should direct you to Shipwreck Bay. Mark the route in on the map. At Shipwreck Bay ask for ship A again. You will be directed back to Pirates' island. Mark this on the map. This time ask for ship B. Mark this on the map. This route goes to Dead Man's Island, at which stage you will be stuck!

Your final map should look like this:



Cards for demonstration activity





Cards for demonstration activity

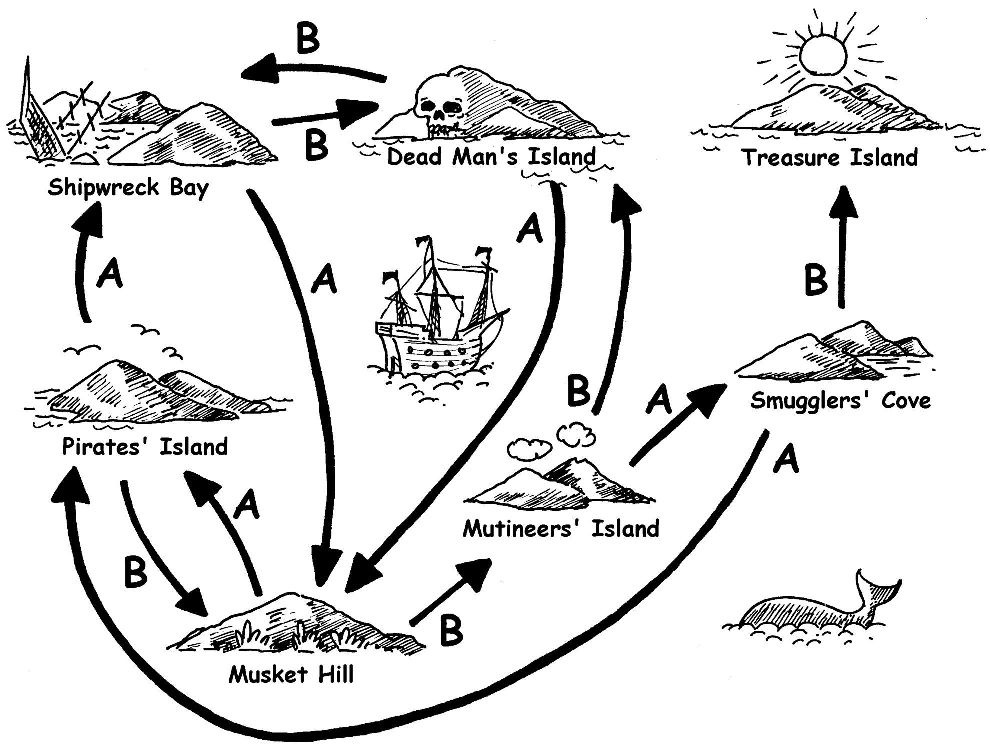
No ships sail from Dead Man’s Island!

Activity

Choose 7 students to be “islands”. The students will hold cards identifying their island, with the secret instructions on the back. Position them randomly around the room or playground. The rest of the students are given the blank map and have to navigate a route from Pirates’ Island to Treasure Island, marking it carefully on their maps. (It is a good idea to send the students off one at a time so they cannot hear the routes in advance.)

Fast Finishers: Try to find more than one route.

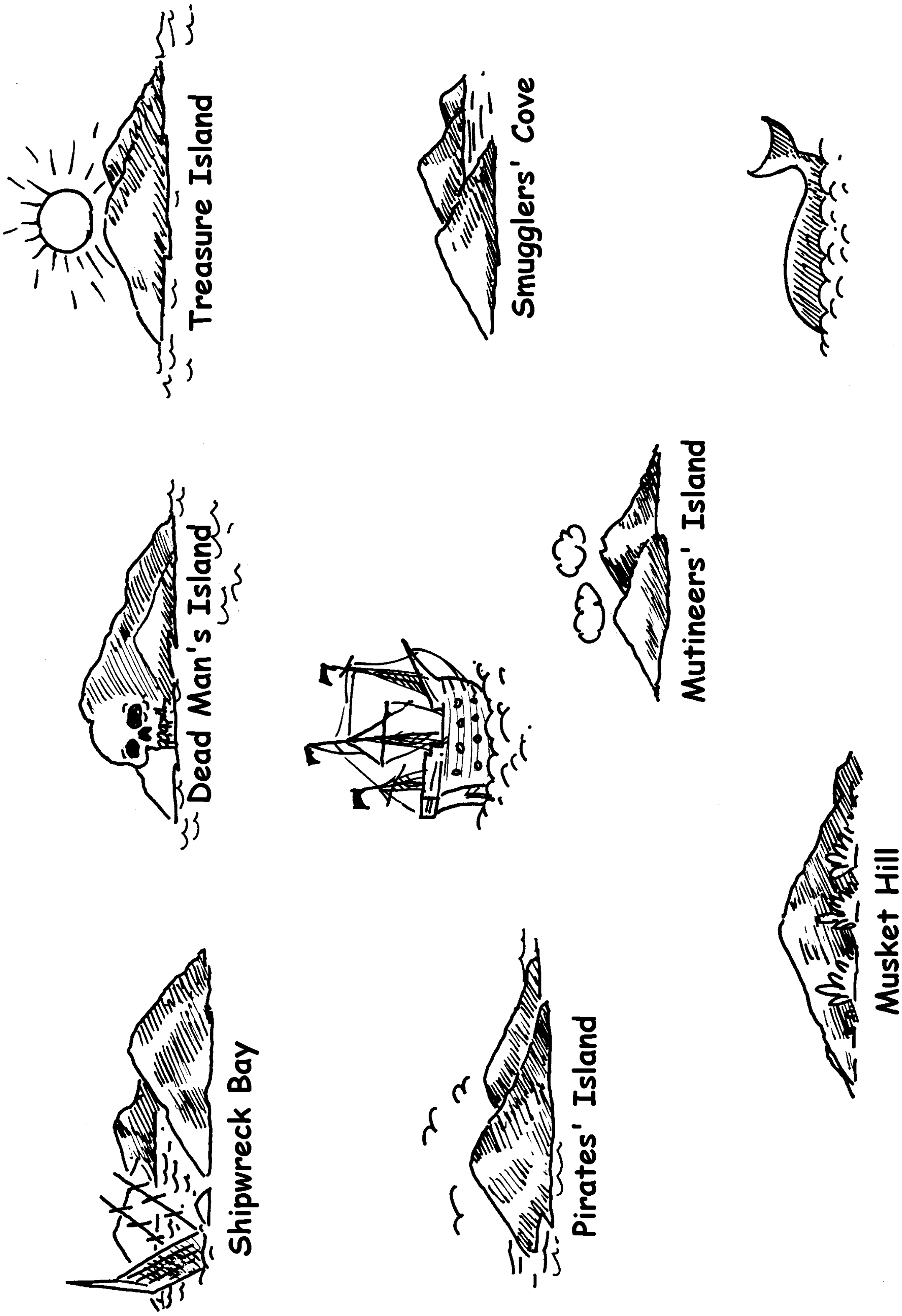
The complete map looks like this:



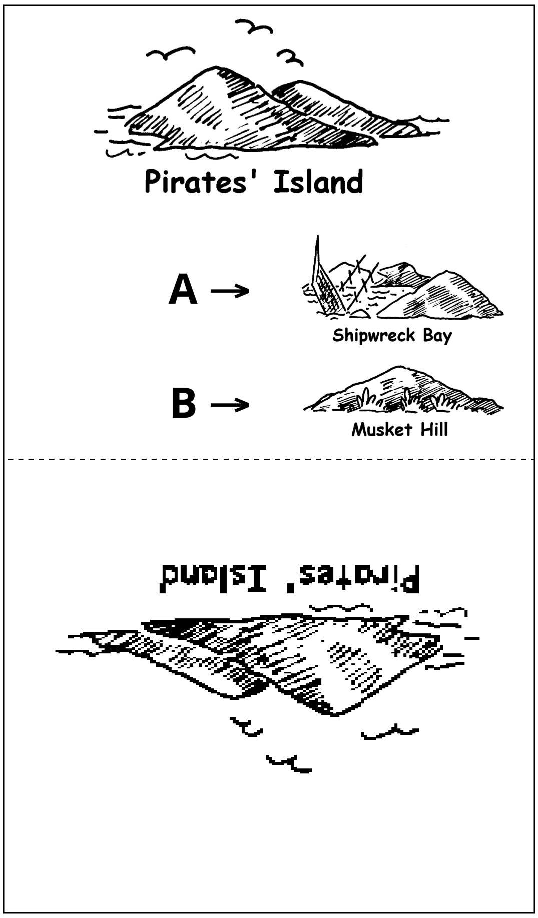
Follow-up discussion

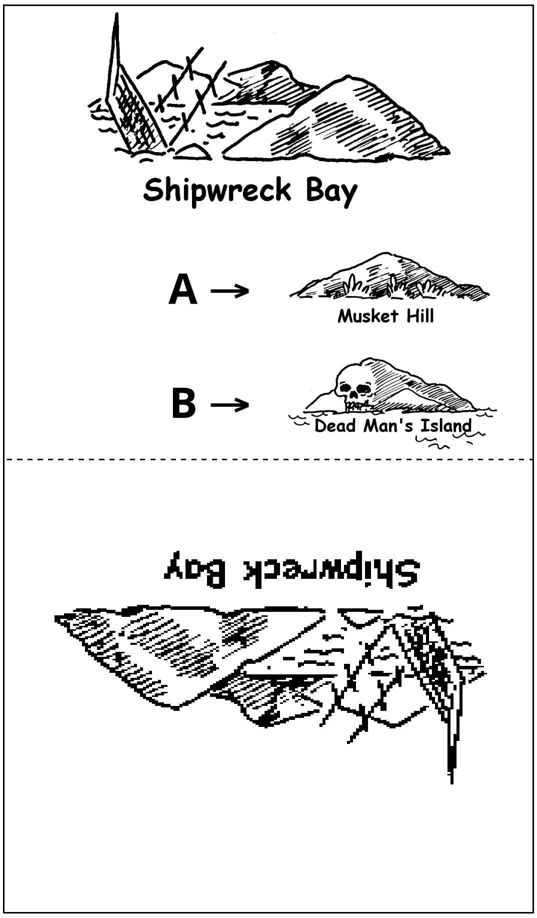
What is the quickest route? What would be a very slow route? Some routes may involve loops. Can you find an example of this? (For example, BBBABAB and BBBABBABAB both get to Treasure Island.)

Worksheet Activity:
Find your way to the riches on Treasure Island

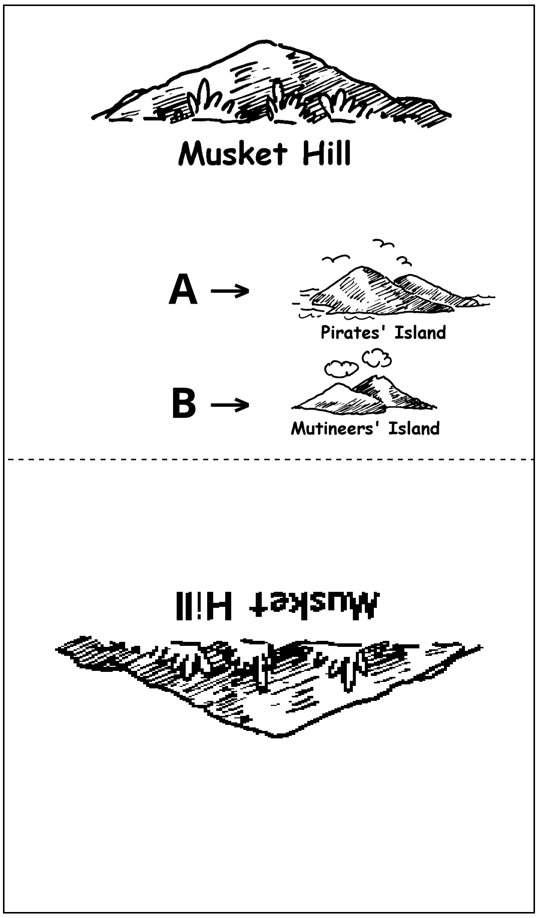


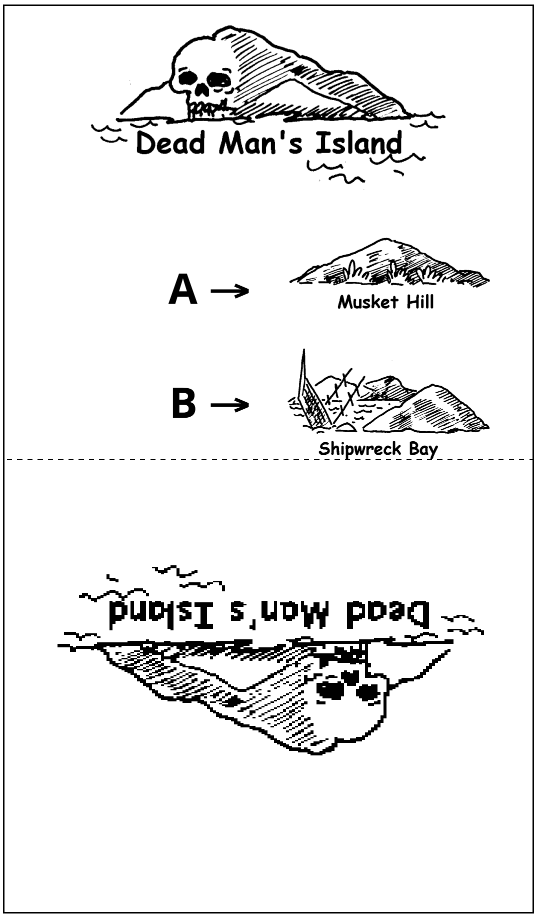
Photocopy Master: Island cards (1/4)



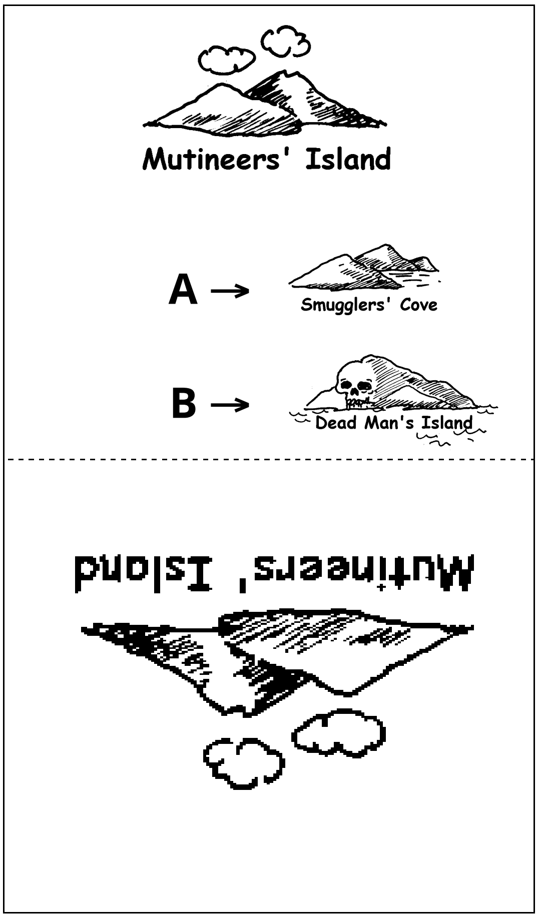


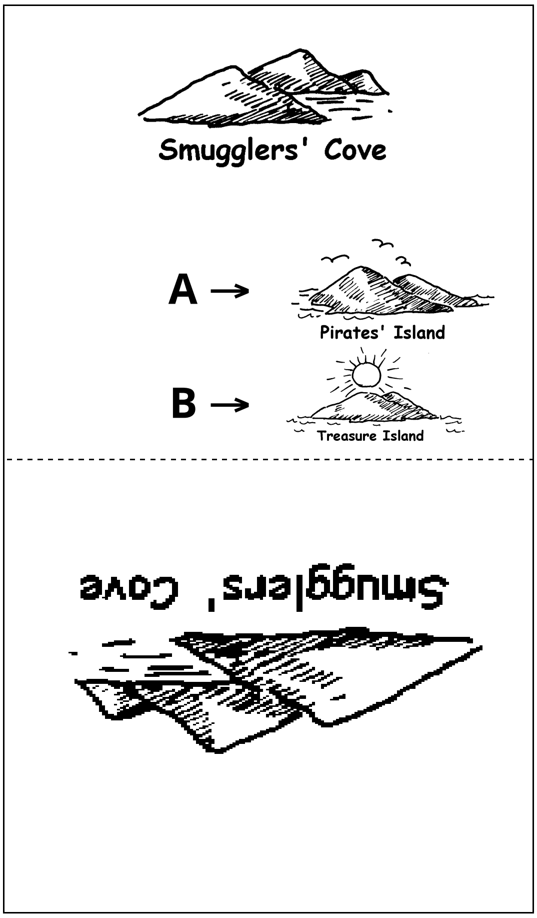
Photocopy Master: Island cards (2/4)



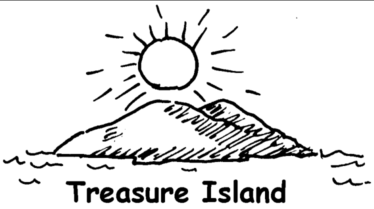


Photocopy Master: Island cards (3/4)

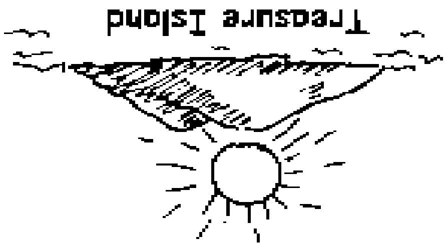




Photocopy Master: Island cards (4/4)

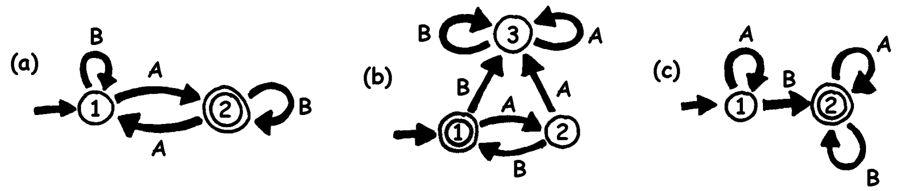


Congratulations!



Finite-State Automata

Another way of drawing a map is like this:



The islands are shown as numbered circles, and the final island (with the treasure) has a double circle. What routes can we travel around to get to the final island? (It's good to explore these by considering examples e.g. does "A" get to the double circled state? "AA"? "ABA"? "AABA"? What's the general pattern?

Solutions:

Map (a) will finish at the double circle (island 2) only if the sequence has an odd number of As (for example, AB, BABAA, or AAABABA).

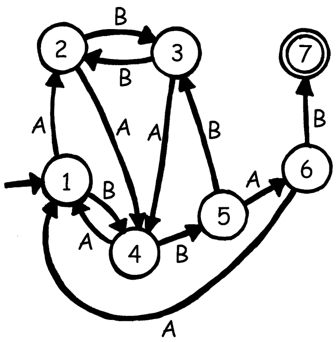
Map (b) only gets to the double circle with an alternating sequence of As and Bs (AB, ABAB, ABABAB, ...).

Map (c) requires that the sequence contains at least one B (the only sequences *not* suitable are A, AA, AAA, AAAA, ...).

Worksheet Activity: Treasure Islands

Can you hide your buried treasure well? How hard can you make it to find the treasure? It's time to make your own map!

1. Here is a more complicated version of the same idea of representing a map. This map is the same as for the previous exercise. Computer Scientists use this quick and easy way of designing routes for their patterns.

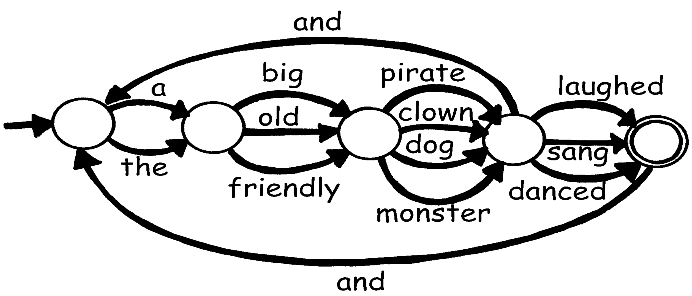


Draw your own basic plan like this so you can clearly see the routes your Pirate ships will travel and then make up your own blank maps and island cards. What is the most efficient sequence of routes to reach your Treasure Island?

2. How well can your friends follow your map? Give them a sequence of As and Bs, and see if they can reach the correct island.

You can make up a variety of games and puzzles based on this idea of finite-state automata.

3. Here is a way of constructing sentences by choosing random paths through the map and noting the words that are encountered.



Now try the same idea for yourself. Perhaps you could even make up a funny story!

Worksheet Activity: The Mysterious Coin Game

Some friends downloaded a game from the Internet in which a robot flipped a coin and they had to guess whether it would turn up heads or tails. At first the game looked very easy. At least they would have a 50/50 chance of winning—or so they thought! After a while though they started to get suspicious. There seemed to be a pattern in the coin tosses. Was the game rigged? Surely not! They decided to investigate. Joe wrote down the results of their next attempts at the game and this is what they found: (h = heads, t = tails)

h h t h h t h h h t t h h h t t h t t t h h h h t h h h t t t h h h t t t h h h h h t
t h t t t t h t t h t t t h h h t t h h h t h h h h h h h h t t h h h t t t t h h h h h t t
t t t t t

Can you find a predictable pattern?

There is a very simple 'map' that will describe the sequence of coin tosses. See if you can figure it out. (**Hint:** it has just 4 'islands')

What’s it all about?

Finite-state automata are used in computer science to help a computer process a sequence of characters or events.

A simple example is when you dial up a telephone number and you get a message that says “Press 1 for this … Press 2 for that … Press 3 to talk to a human operator.” Your key presses are inputs for a finite state automaton at the other end of the phone. The dialogue can be quite simple, or very complex. Sometimes you are taken round in circles because there is a peculiar loop in the finite-state automaton. If this occurs, it is an error in the design of the system—and it can be extremely frustrating for the caller!

Another example is when you get cash from a bank cash machine. The program in the machine’s computer leads you through a sequence of events. Inside the program all the possible sequences are kept as a finite-state automaton. Every key you press takes the automaton to another state. Some of the states have instructions for the computer on them, like “dispense $100 of cash” or “print a statement” or “eject the cash card”.

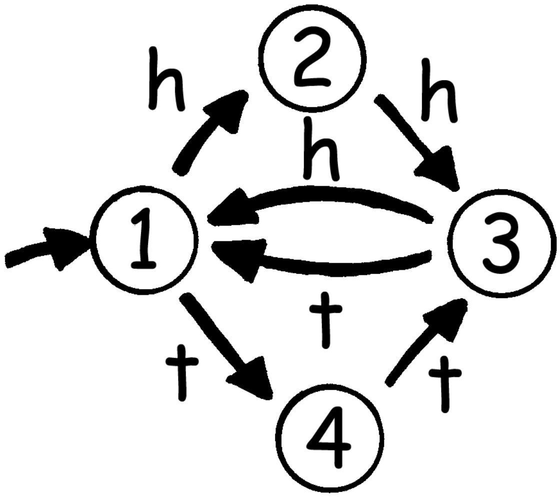
Some computer programs really do deal with English sentences using maps like the one on page 119. They can both generate sentences themselves, and process sentences that the user types in. In the 1960s a computer scientist wrote a famous program called “Eliza” (after Eliza Dolittle) that had conversations with people. The program pretended to be a psychotherapist, and came out with leading questions like “Tell me about your family” and “Do go on.” Although it didn’t “understand” anything, it was sufficiently plausible—and its human users were sufficiently gullible—that some people really did think they were talking to a human psychotherapist.

Although computers are not really very good at understanding natural language, they can readily process artificial languages. One important type of artificial language is the programming language. Computers use finite-state automata to read in programs and translate them into the form of elementary computer instructions, which can then be “executed” directly by the computer.

Solutions and hints

The Mysterious Coin Game (page 120)

The mysterious coin game uses the following map to toss coins:



If you follow it, you will see that the first two coin tosses of each three have the same outcome.

Activity 13

Marching Orders—*Programming Languages*

Summary

Computers are usually programmed using a “language,” which is a limited vocabulary of instructions that can be obeyed. One of the most frustrating things about programming is that computers always obey the instructions to the letter, even if they produce a crazy result. This activity gives students some experience with this aspect of programming.

Curriculum Links

✓ English: Interpersonal Listening

Skills

✓ Giving and following instructions.

Ages

✓ 7 years and up

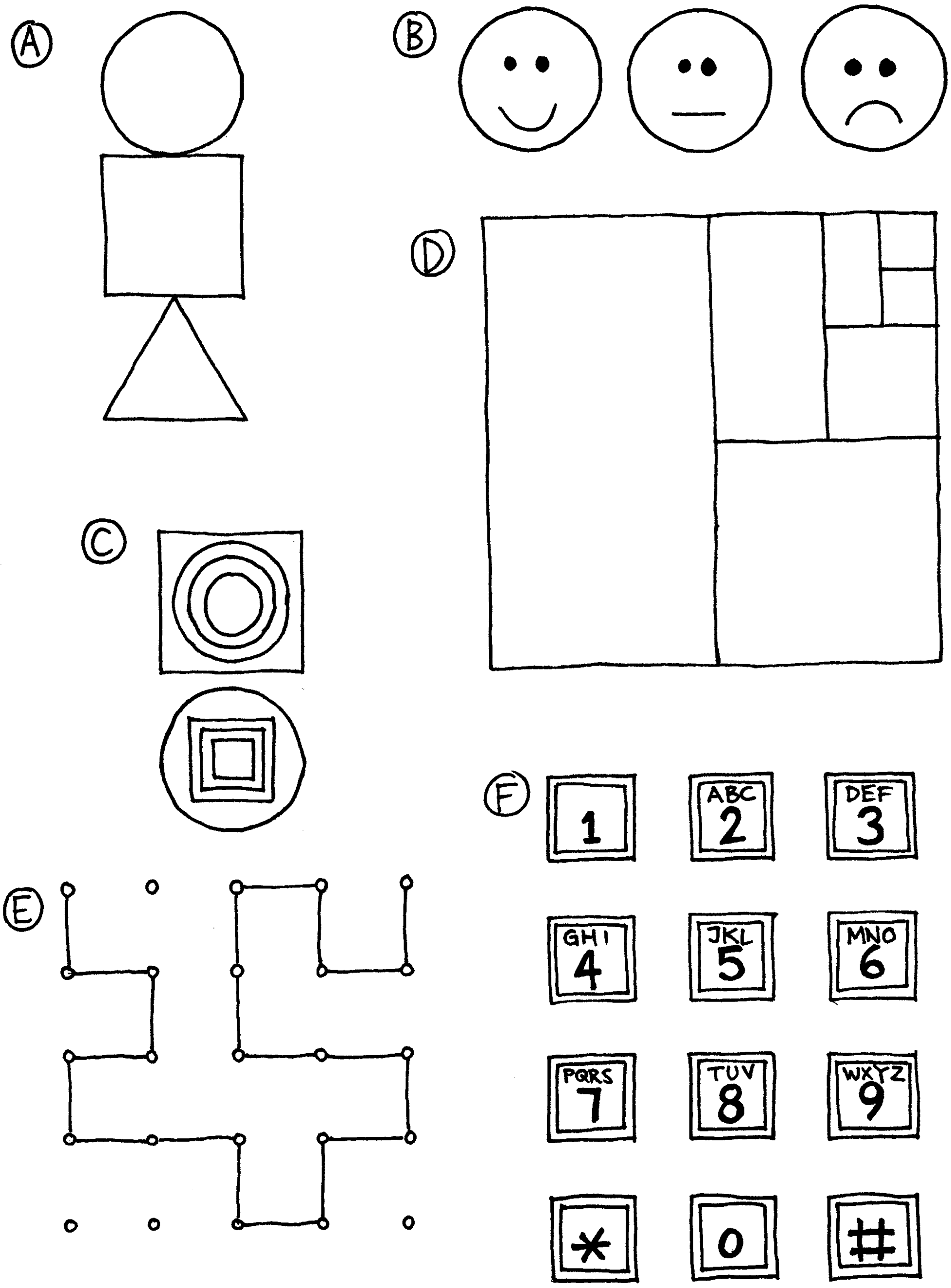
Materials

You will need:

✓ Cards with pictures such as the ones shown on the next page.

Each student will need:

✓ Pencil, paper and ruler


A
B
C
D
E
F
1
ABC 2
DEF 3
GHI 4
JKL 5
MNO 6
PQRS 7
TUV 8
WXYZ 9
*
0
#

Marching Orders

Introduction

Discuss whether it would be good if people followed instructions exactly. For example, what would happen if you pointed to a closed door and said, “Go through that door”?

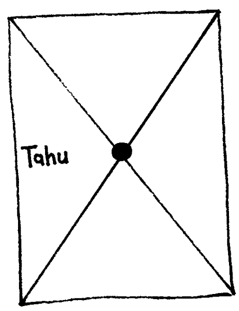
Computers work by following lists of instructions, and they do exactly what the instructions say—even if they don’t make sense!

Demonstration Example

See if the students can draw the picture from these instructions.

1. Draw a dot in the centre of your page.
2. Starting at the top left-hand corner of the page rule a straight line through the dot finishing at the bottom right hand corner.
3. Starting at the bottom left-hand corner of the page rule a line through the dot, finishing at the top right hand corner.
4. Write your name in the triangle in the centre of the left-hand side of the page.

The result should look something like this:



Activities

Choose a student and give them an image (such as the examples on page 124). The student describes the picture for the class to reproduce. The students can ask questions to clarify the instructions. The object is to see how quickly and accurately the exercise can be completed.

Repeat the exercise, but this time the students are not allowed to ask questions. It is best to use a simpler image for this exercise, as the students can get lost very quickly.

Now try the exercise with the instructing student hidden behind a screen, without allowing any questions, so that the only communication is in the form of instructions.

Point out that this form of communication is most like the one that computer programmers experience when writing programs. They give a set of instructions to the computer, and don't find out the effect of the instructions until afterwards.

Have the students draw a picture and write down their own instructions. Try them out in pairs or as a whole class.

Variations

1. Write instructions to construct a paper dart.

2. Write instructions on how to get to a mystery location around the school using such instructions as "Go forward *x* metres", "turn left" (90 degrees), and "turn right" (90 degrees).

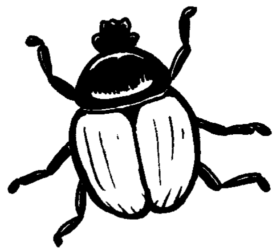
   Students should test and refine their instructions until they have the desired effect.

3. Blind Game. Blindfold a student and have the other students direct them around the room.

What’s it all about?

Computers operate by following a list of instructions, called a program, that has been written to carry out a particular task. Programs are written in languages that have been specially designed, with a limited set of instructions, to tell computers what to do. Some languages are more suitable for some purposes than others.

Regardless of what language they use, programmers must become adept at specifying *exactly* what they want the computer to do. Unlike human beings, a computer will carry out instructions to the letter even if they are patently ridiculous.



It is important that programs are well written. A small error can cause a lot of problems. Imagine the consequences of an error in the program of a computer in a space shuttle launch, a nuclear power plant, or the signals on a train track! Errors are commonly called “bugs” in honour (so it is said) of a moth that was once removed (“debugged”) from an electrical relay in an early 1940s electronic calculating machine.

The more complex the program, the more errors there are likely to be. This became a major issue when the USA was working on the Strategic Defence Initiative (“Star Wars”) program, a computer controlled system that was intended to form an impenetrable defence against nuclear attack. Some computer scientists claimed that it could never work because of the complexity and inherent unreliability of the software required. Software needs to be tested carefully to find as many bugs as possible, and it wouldn’t be feasible to test this system since one would have to fire missiles at the United States to be sure that it worked!

Part IV

Really hard problems—*Intractability*

Intractability

Are there problems that are too hard even for computers? Yes. We will see in Activity 20 that just having a conversation—chatting—is something computers can't do, not because they can't speak but because they can't understand or think of sensible things to say, but that's not the kind of hard problem we're talking about here – it's not that computers can't have conversations, more that we don't know just how we do it ourselves and so we can't tell the computer what to do. But in this section we're going to look at problems where it's easy to tell the computer what to do—by writing a program—but the computer can't do what we want because it takes far too long: millions of centuries, perhaps. Not much good buying a faster computer: if it were a hundred times faster it would still take millions of years; even one a million times faster would take hundreds of years. That's what you call a *hard* problem—one where it takes far longer than the lifetime of the fastest computer imaginable to come up with a solution!

The activities in Part II on algorithms showed you how to find ways of making computer programs run more efficiently. In this section we look at problems for which *no* efficient solutions are known, problems that take computers millions of centuries to solve. And we will encounter what is surely the greatest mystery in computer science today: that *no-one knows* whether there's a more efficient way of solving these problems! It may be just that no-one has come up with a good way yet, or it may be that there is no good way. We don't know which. And that's not all. There are thousands of problems that, although they look completely different, are equivalent in the sense that if an efficient method is found to solve one, it can be converted into an efficient method to solve them all. In these activities you will learn about these problems.

For teachers

There are three activities in this section. The first involves coloring maps and counting how many colors are needed to make neighboring countries different. The second requires the ability to use a simple street map, and involves placing ice-cream vans at street corners so that nobody has to go too far to get an ice-cream. The third is an outdoor activity that uses string and pegs to explore how to make short networks connecting a set of points.

The activities provide a hands-on appreciation of the idea of complexity—how problems that are very simple to state can turn out to be incredibly hard to solve. And these problems are not abstruse. They are practical questions that arise in everyday activities such as mapping, school time-tabling, and road building. The computational underpinning rests on a notion called "NP-completeness” that is explained in the *What's it all about?* sections at the end of each activity. Although the activities themselves can be tackled in any order, these sections are intended to be read in the order in which they appear. By the time you reach the end you will have a firm grip on the most important open question in contemporary computer science.

The technical name for this part is “intractability” because problems that are hard to solve are called *intractable*. The word comes the Latin *tractare* meaning to draw or drag, leading to the modern usage of *tractable* as easy to handle, pliant, or docile. Intractable problems are ones that are not easily dealt with because it would take too long to come up with an answer. Although it may sound esoteric, intractability is of great practical interest because a breakthrough in this area would have major ramifications for many different lines of research. For example, most cryptographic codes rely on the intractability of some problems, and a criminal who managed to come up with an efficient solution could have a field day decoding secrets and selling them, or—more simply—just making phoney bank transactions. We will look at these things in Part V—Cryptography.

Activity 14

The poor cartographer—Graph coloring

Summary

Many optimization problems involve situations where certain events cannot occur at the same time, or where certain members of a set of objects cannot be adjacent. For example, anyone who has tried to timetable classes or meetings will have encountered the problem of satisfying the constraints on all the people involved. Many of these difficulties are crystallized in the map coloring problem, in which colors must be chosen for countries on a map in a way that makes bordering countries different colors. This activity is about that problem.

Curriculum Links

- ✓ Mathematics: Number – Exploring numbers in other bases. Representing numbers in base two.
- ✓ Mathematics: Algebra – Continue a sequential pattern, and describe a rule for this pattern. Patterns and relationships in powers of two.

Skills

- ✓ Problem solving.
- ✓ Logical reasoning.
- ✓ Algorithmic procedures and complexity.
- ✓ Communication of insights.

Ages

- ✓ 7 and up

Materials

- ✓ a whiteboard or similar writing surface.

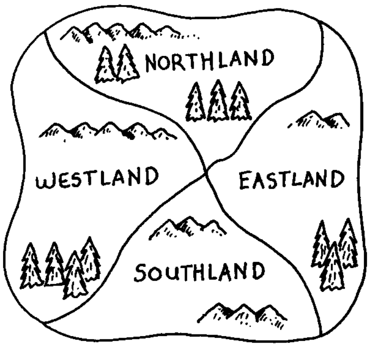
Each student will need:

- ✓ a copy of one or more of the worksheets,
- ✓ movable small colored markers (e.g. counters or poker chips), and
- ✓ four crayons of different colors (or colored pencils, felt tips etc.)

Graph Coloring

Introduction

This activity revolves around a story in which the students have been asked to help out a cartographer, or map-maker, who is coloring in the countries on a map. It doesn't matter which color a country is, so long as it's different to all bordering countries.



For example, this map shows four countries. If we color Northland red, then Westland and Eastland cannot be red, since their border with Northland would be hard to see. We could color Westland green, and it is also acceptable to color Eastland green because it does not share a border with Westland. (If two countries meet only at a single point, they do not count as sharing a border and hence can be made the same color.) Southland can be colored red, and we end up needing only two colors for the map.

In our story, the cartographer is poor and can't afford many crayons, so the idea is to use as few colors as possible.

Discussion

Describe the problem that the students will be working on, demonstrating the coloring process on a blackboard.

Give out a copy of the first worksheet. This map can be colored correctly using only two colors. Although restricting the number of colors to just two might sound particularly challenging, the task is quite simple compared with maps that require more colors because there is very little choice about what color each country can be.

Have the students try to color the map in with only two colors. In the process they may discover the "has-to-be" rule: once one country is colored in, any bordering country has to be the opposite color. This rule is applied repeatedly until all countries are colored in. It is best if the students can discover this rule for themselves, rather than being told it, as it will give them a better insight into the process.

As students complete each exercise they can be given the next sheet to try.

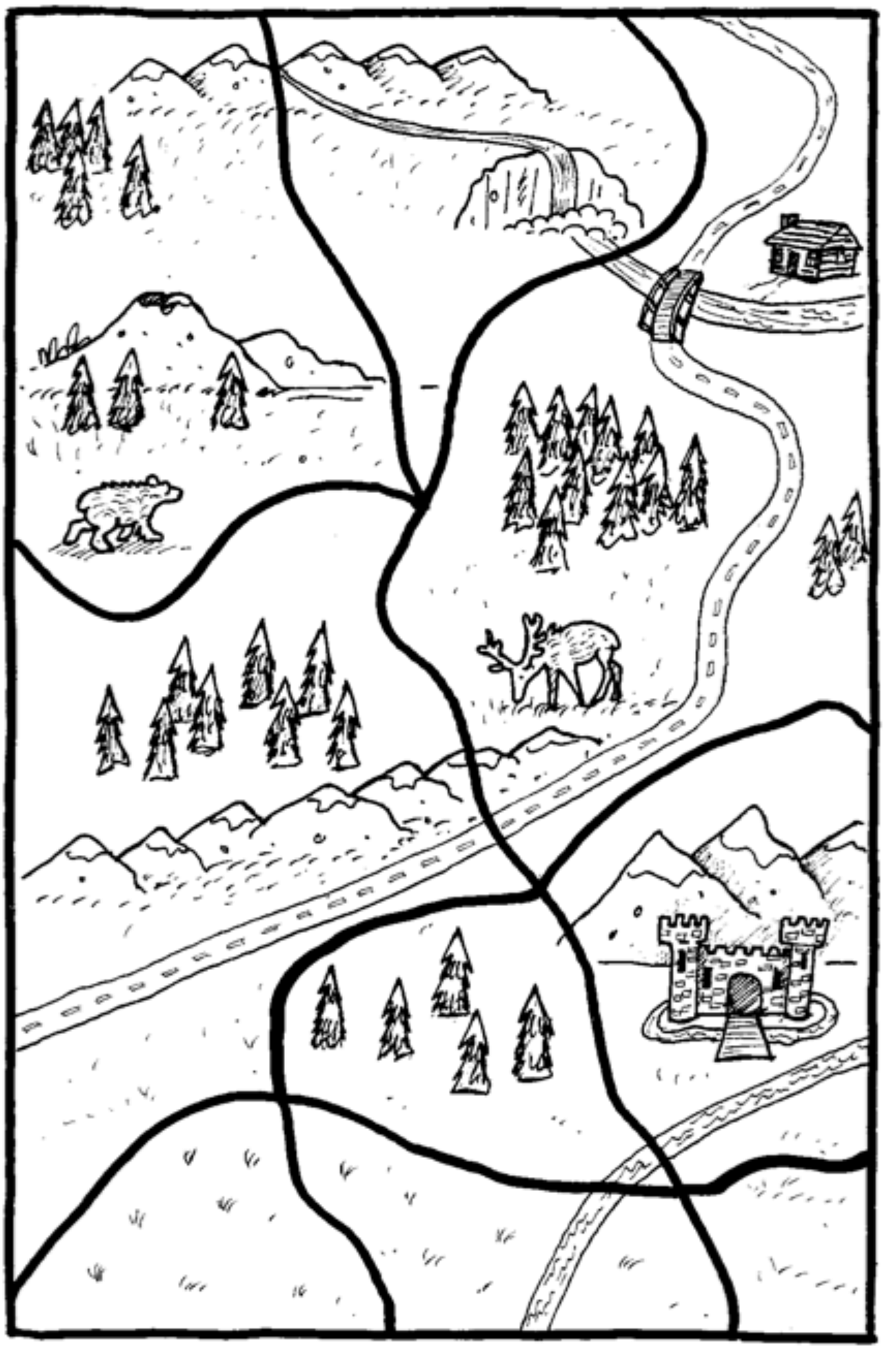
The students may also discover that it is better to use place-holders, such as colored counters, instead of coloring the countries straight away, since this makes it easier for them to change their mind.

For older students, ask them to explain how they know that they have found the minimum number of colors. For example, at least three colors are required for this map because it includes a group of three countries (the largest three), each of which has borders with the other two.

If a student finishes all the sheets early, ask them to try to devise a map that requires five different colors. It has been proved that any map can be colored with only four colors, so this task will keep them occupied for some time! In our experience students will quickly find maps that they believe require five colors, but of course it is always possible to find a four-color solution to their maps.

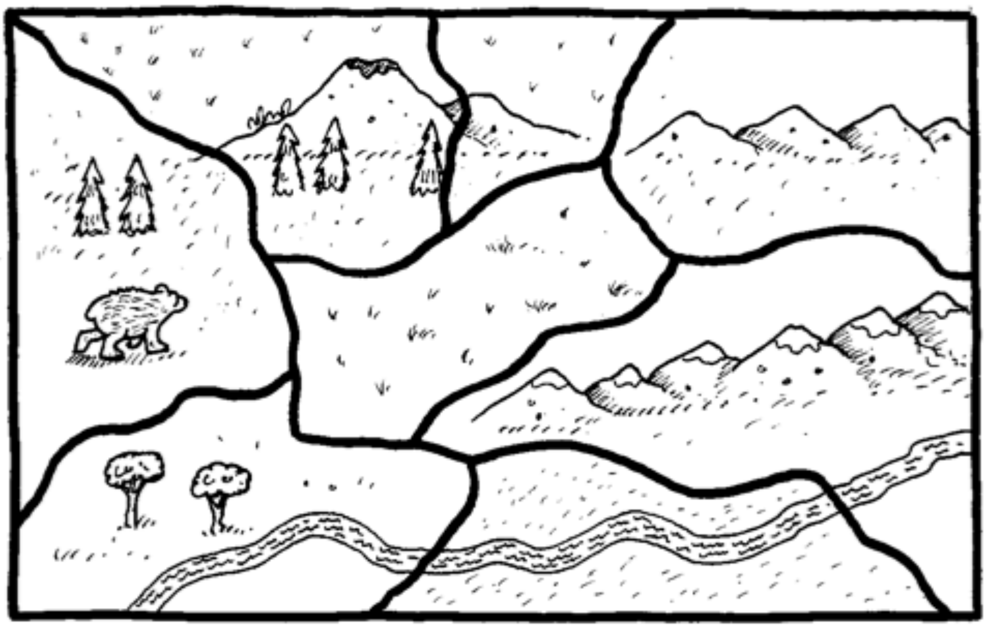
Worksheet Activity: Graph Coloring 1

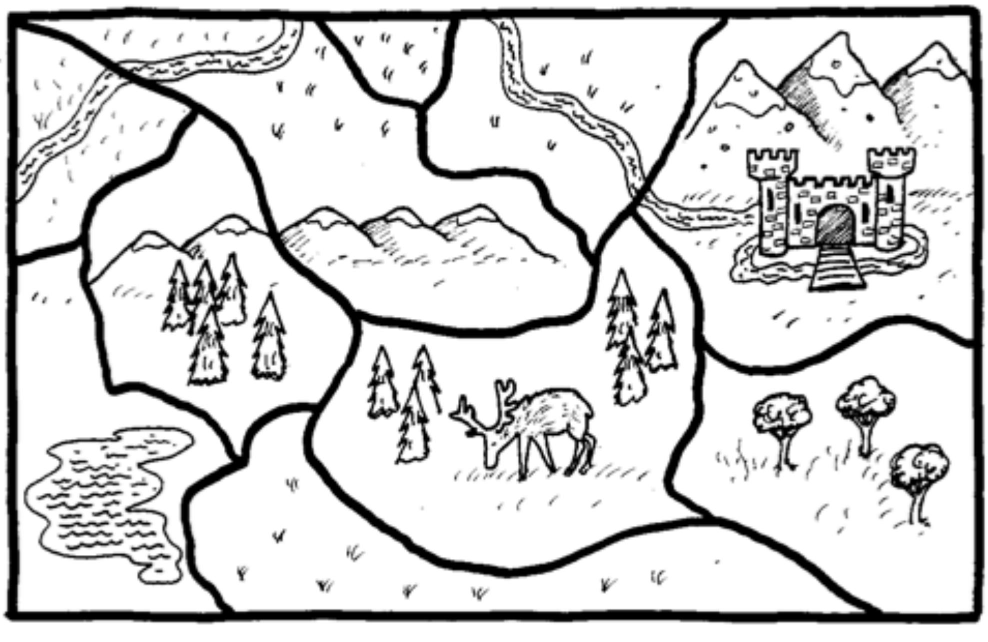
Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.



Worksheet Activity: Graph Coloring 2

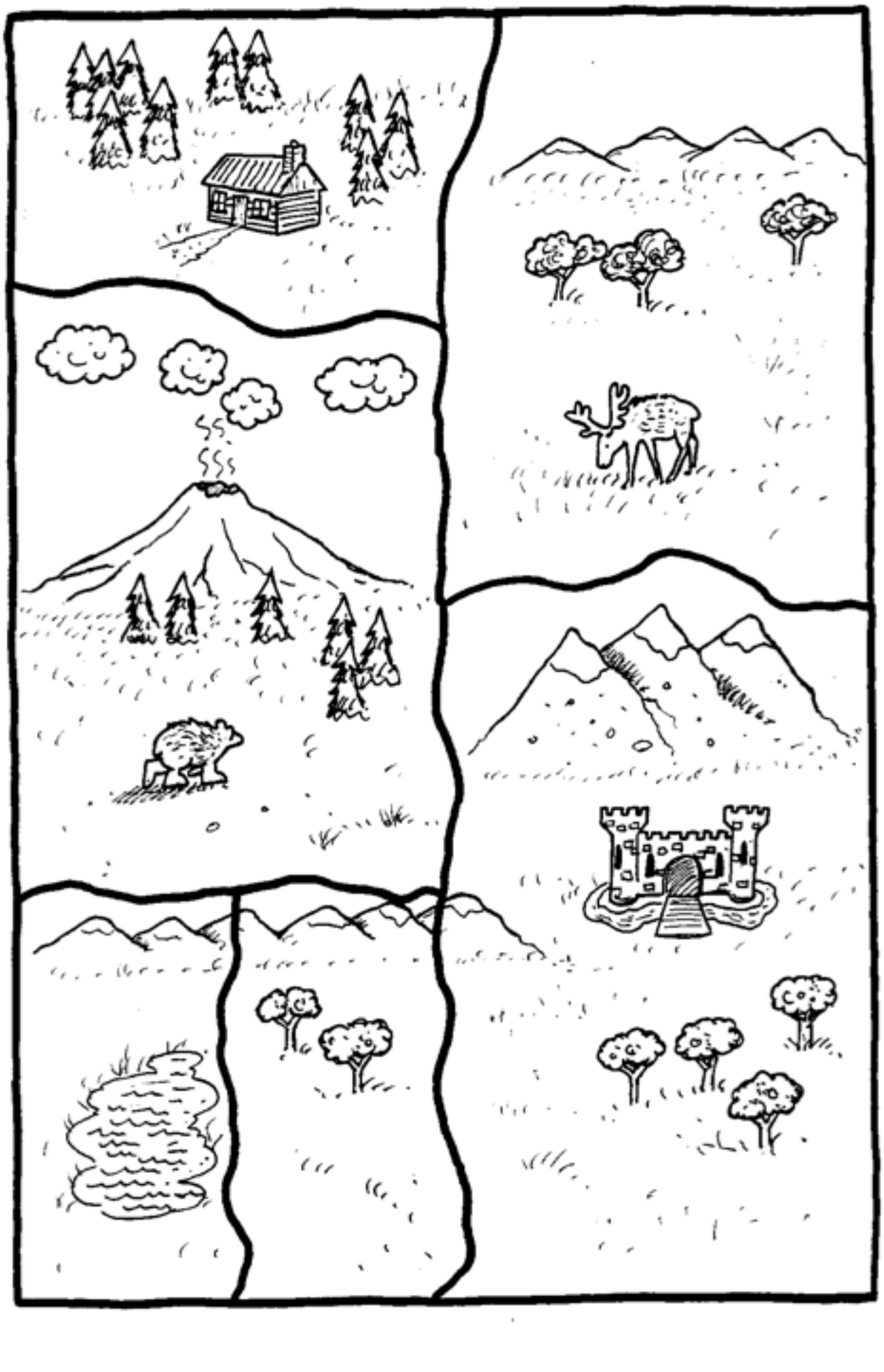
Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.





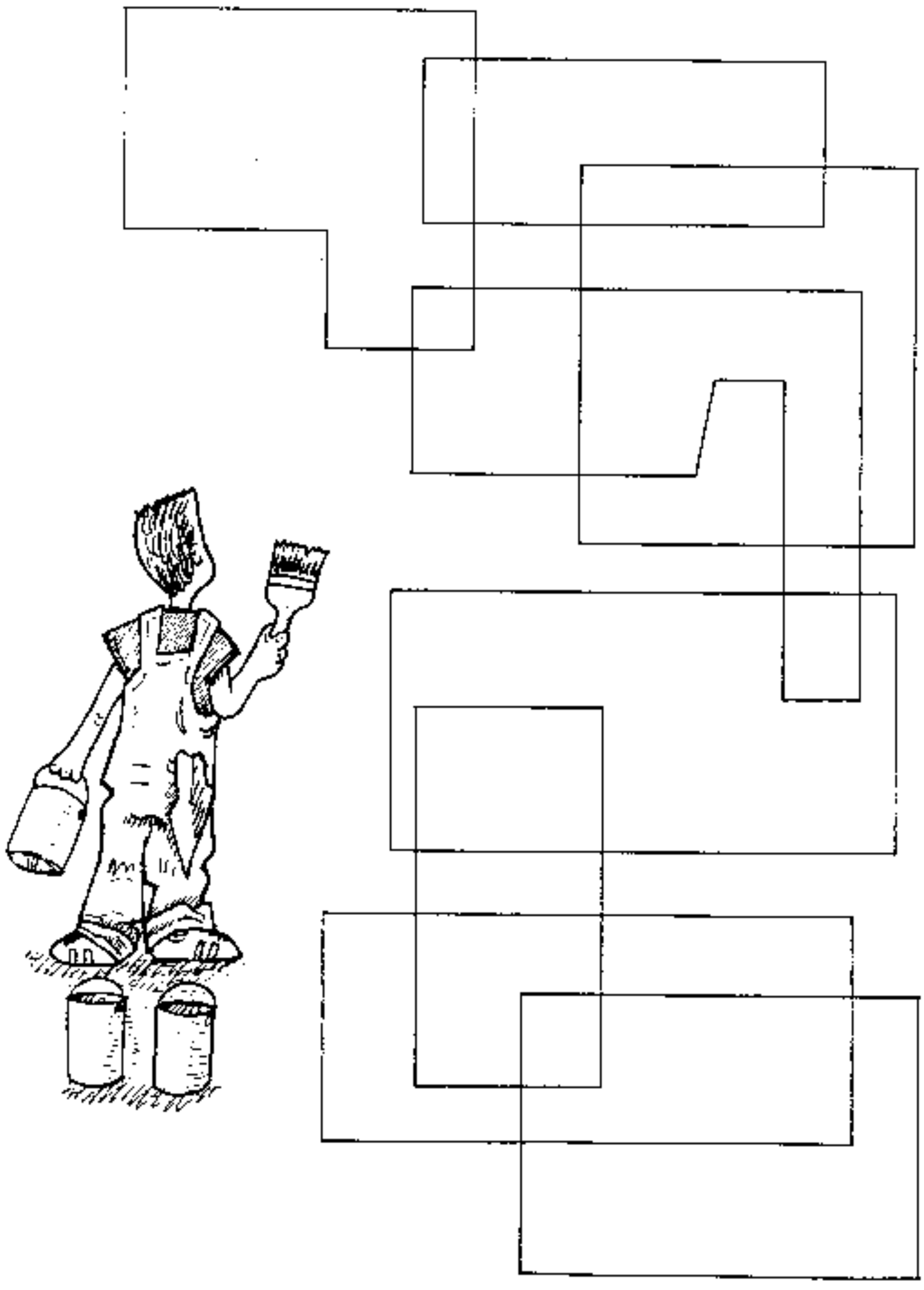
Worksheet Activity: Graph Coloring 3

Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.

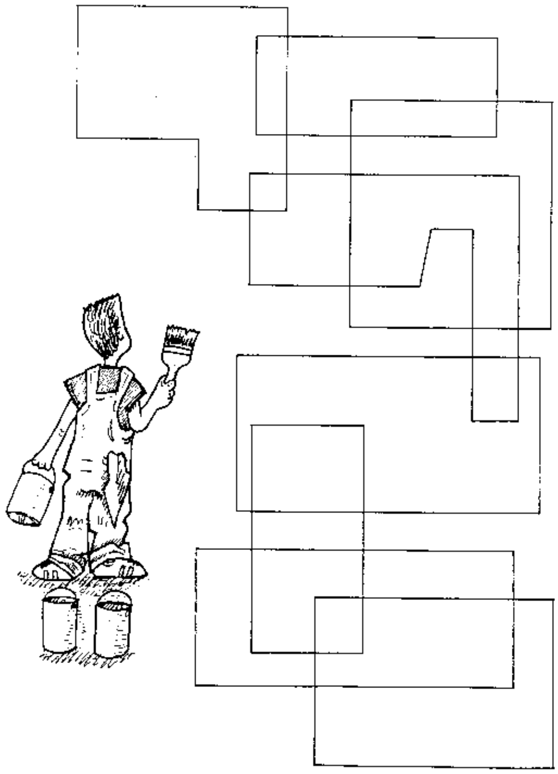


Worksheet Activity: Graph Coloring 4

Color in the countries on this map with as few colors as possible, but make sure that no two bordering countries are the same color.



Variations and Extensions



There is a simple way to construct maps that require only two colors, as shown here. This map was drawn by overlaying closed curves (lines whose beginning joins up with their end). You can draw any number of these curves, of any shape, on top of each other, and you will always end up with a map that can be colored with two colors. Students can experiment with creating this type of map.

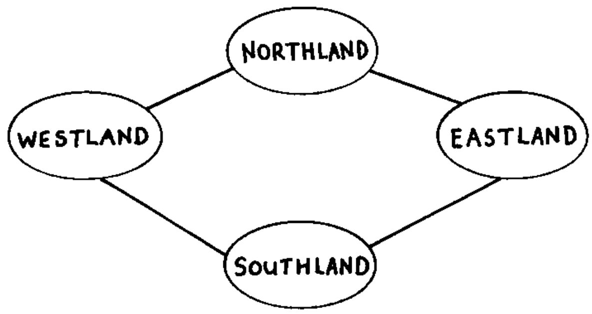
Four colors are always enough to color a map drawn on a sheet of paper or on a sphere (that is, a globe). One might wonder (as scientists are paid to do) how many colors are needed for maps drawn on weirder surfaces, such as the torus (the shape of a donut). In this case, one might need five colors, and five is always enough. Students might like to experiment with this.

There are many other entertaining variations on the map-coloring problem that lead off into directions where much is currently unknown. For example, if I am coloring a map on a sheet of paper by myself, then I know that if I work cleverly, four colors will be enough. But suppose that instead of working alone I am working with an incompetent (or even adversarial) partner, and we take turns at choosing the color for countries. Assume that I work cleverly, while my partner only works “legally” as we take turns coloring countries on the map. How many crayons need to be on the table in order for me in my cleverness to be able to make up for my partner's legal but not very bright (or even subversive) moves? The maximum number isn’t known! In 1992 it was proved that 33 crayons will always be enough, and in 2008 this was improved by a proof that 17 would be sufficient, but we still don't know that this many is ever actually required. (Experts conjecture that fewer than 10 colors are sufficient.) Students might enjoy acting out this situation, which can be played as a two-person game where you try to maximise the number of colours that your opponent needs.

In another variation of map coloring know as *empire coloring*, we start with two different maps on two sheets of paper having the same number of countries. Each country on one of the maps (say, the Earth) is paired with exactly one country on the other map (which might be colonies on the Moon). In addition to the usual coloring requirement of different colors for countries that share a border (for both maps) we add the requirement that each Earth country must be colored the same as its colony on the Moon. How many colors do we need for this problem? The answer is currently unknown.

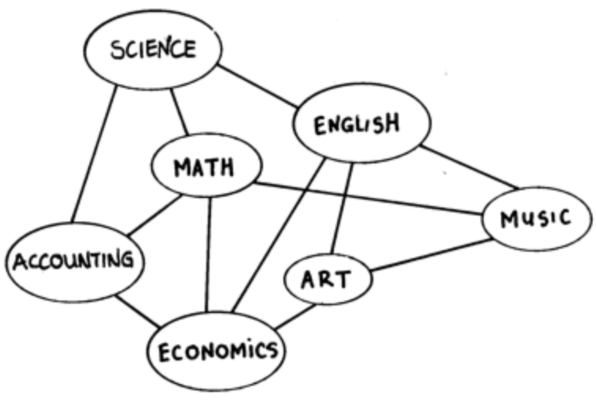
What's it all about?

The map coloring problem that we have explored in this activity is essentially to find the minimum number of colors—two, three, or four—that are necessary to color a particular map. The conjecture that any map can be colored using only four colors was formulated in 1852, but it was not proved until 1976. Computer science is full of unsolved problems, and knowing that the four-color theorem was proved after more than 120 years of attention from researchers is an encouragement to those working on other problems whose solution has eluded them for decades.



Map coloring belongs to a general class of problems known as "graph coloring." In computer science, a graph is an abstract representation of relationships, as shown here.

As mentioned in Activity 9 on the Muddy City, the term graph is used in a different sense in mathematics to mean a chart displaying numerical data, such as a bar graph, but the graphs that computer scientists use are not related to these. In computer science, graphs are drawn using circles or large dots, technically called "nodes," to denote objects, with lines between them to indicate some sort of relationship between the objects. The above graph happens to represent the map at the beginning of this activity. The nodes represent the countries, and a line between two nodes indicates that those two countries share a common border. On the graph, the coloring rule is that no connected nodes should be allocated the same color. Unlike a map, there is no limit to the number of colors that a general graph may require, because many different constraints may be drawn in as connecting lines, whereas the two-dimensional nature of maps restricts the possible arrangements. The "graph coloring problem" is to find the minimum number of colors that are needed for a particular graph.



In the graph on the right the nodes correspond to subjects in a school. A line between two subjects indicates that at least one student is taking both subjects, and so they should not be timetabled for the same period. Using this representation, the problem of finding a workable timetable using the minimum number of periods is equivalent to the coloring problem, where the different colors correspond to different periods. Graph coloring algorithms are of great interest in computer science, and are used for many real-world problems, although they are probably never used to color in maps!—our poor cartographer is just a fiction.

There are literally thousands of other problems based on graphs. Some are described elsewhere in this book, such as the minimal spanning tree of Activity 9 and the dominating sets of Activity 14. Graphs are a very general way of representing data and can be used to represent all sorts of situations, such as a map made up of roads and intersections, connections between atoms in a molecule, paths that messages can take through a computer network, connections between components on a printed circuit board, and relationships between the tasks required to carry out a large project. For this reason, problems involving graph representations have long fascinated computer scientists.

Many of these problems are very difficult—not difficult conceptually, but difficult because they take a long time to solve. For example, to determine the most efficient solution for a graph coloring problem of moderate size—such as finding the best way to timetable a school with thirty teachers and 800 students—could take years, even centuries, for a computer using the best known algorithm. The problem would be irrelevant by the time the solution was found—and that's assuming the computer doesn't break down or wear out before it finishes! Such problems are only solved in practice because we are content to work with sub-optimal, but still very good, solutions. If we were to insist on being able to guarantee that the solution found was the very best one, the problem would be completely intractable.

The amount of computer time needed to solve coloring problems increases exponentially with the size of the graph. Consider the map coloring problem. It can be solved by trying out all possible ways to color the map. We know that at most four colors are required, so we need to evaluate every combination of assigning the four colors to the countries. If there are n countries, there are 4^n combinations. This number grows very rapidly: every country that is added multiplies the number of combinations by four, and hence quadruples the solution time. Even if a computer were invented that could solve the problem for, say, fifty countries in just one hour, adding one more country would require four hours, and we would only need to add ten more countries to make the computer take over a year to find the solution. This kind of problem won't go away just because we keep inventing faster and faster computers!

Graph coloring is a good example of a problem whose solution time grows exponentially. For very simple instances of the problem, such as the small maps used in this activity, it is quite easy to find the optimal solution, but as soon as the number of countries increases beyond about ten, the problem becomes very difficult to do by hand, and with a hundred or more countries, even a computer can take many years to try out all the possible ways of coloring the map in order to choose the optimal one.

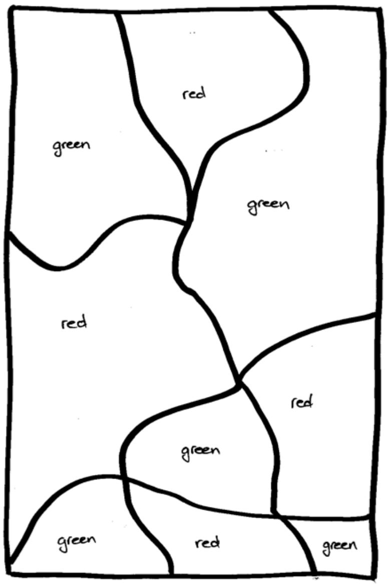
Many real-life problems are like this, but must be solved anyway. Computer scientists use methods that give good, but not perfect, answers. These *heuristic* techniques are often very close to optimal, very fast to compute, and give answers that are close enough for all practical purposes. Schools can tolerate using one more classroom than would be needed if the timetable were perfect, and perhaps the poor cartographer could afford an extra color even though it is not strictly necessary.

No-one has proved that there isn't an efficient way to solve this sort of problem on conventional computers, but neither has anyone proved that there is, and computer scientists are sceptical that an efficient method will ever be found. We will learn more about this kind of problem in the next two activities.

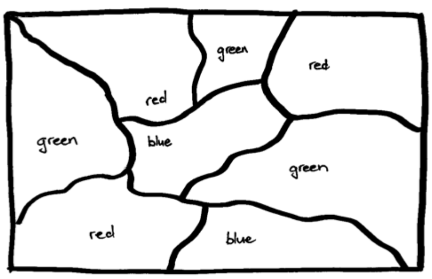
Further reading

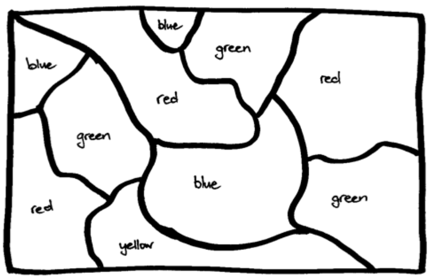
Harel discusses the four-color theorem, including its history, in *Algorithmics*. More aspects of the map-coloring problem are discussed in *This is MEGA-Mathematics!* by Casey and Fellows. Kubale's 2004 book, *Graph Colorings,* includes a history of the problem. There are many websites that cover this topic.

Solutions and Hints

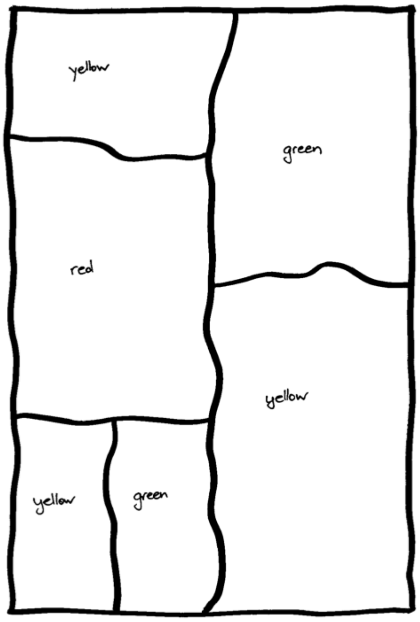


This is the only possible solution for the map on worksheet 1 (of course, the choice of colors is up to the student, but only two different colors are required).

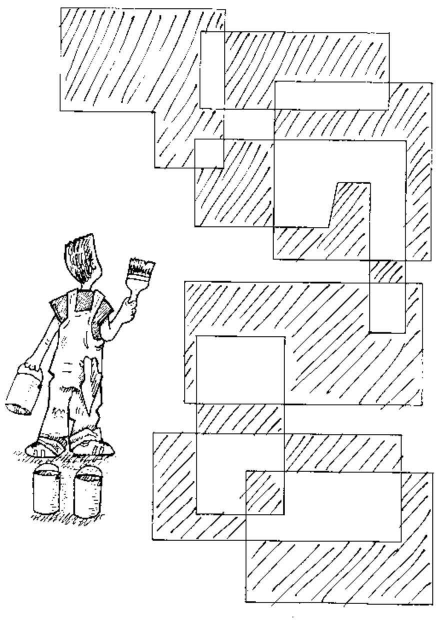




The map at the top of worksheet 2 can be colored correctly using three colors, while the one at the bottom requires four. Here are two possible solutions.



The map on worksheet 3 is a simpler three-color map, with a possible solution shown here.



Solution for worksheet 4 using just two colors (shaded and white).

Activity 15

Tourist town—*Dominating sets*

Summary

Many real-life situations can be abstracted into the form of a network or "graph" of the kind used for coloring in Activity 13. Networks present many opportunities for the development of algorithms that are practically useful. In this activity, we want to mark some of the junctions, or "nodes," in such a way that all other nodes are at most one step away from one of the marked ones. The question is, how few marked nodes can we get away with? This turns out to be a surprisingly difficult problem.

Curriculum Links

- ✓ Mathematics – Position and orientation
- ✓ Mathematics – Logical reasoning

Skills

- ✓ Maps
- ✓ Relationships
- ✓ Puzzle solving
- ✓ Iterative goal seeking

Ages

- ✓ 7 and up

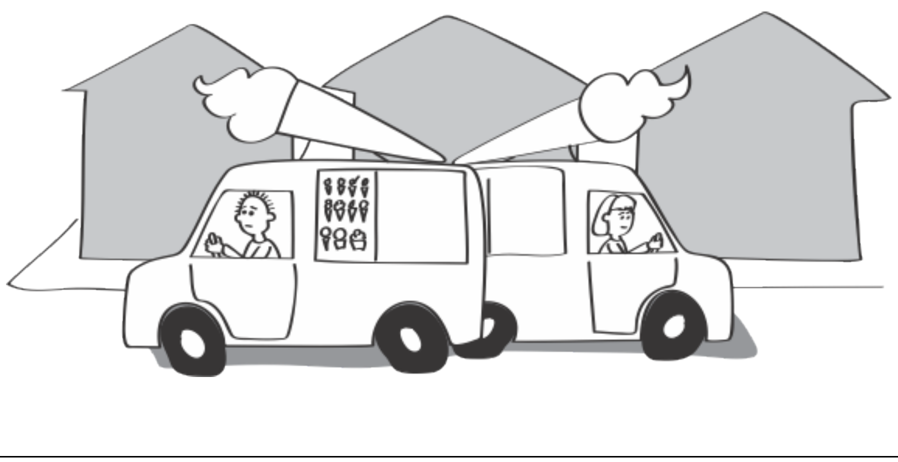
Materials

Each group of students will need:

- ✓ a copy of the blackline master *Ice Cream Vans*, and
- ✓ several counters or poker chips of two different colors.

You will need

- ✓ a projector image of the blackline master Ice Cream Vans Solution on a whiteboard, or a whiteboard to draw it on.



Dominating Sets

Introduction

The *Ice Cream Vans* worksheet shows a map of Tourist Town. The lines are streets and the dots are street corners. The town lies in a very hot country, and in the summer season ice-cream vans park at street corners and sell ice-creams to tourists. We want to place the vans so that anyone can reach one by walking to the end of their street and then at most one block further. (It may be easier to imagine people living at the intersections rather than along the streets; then they must be able to get ice-cream by walking at most one block.) The question is, how many vans are needed and on which intersections should they be placed?

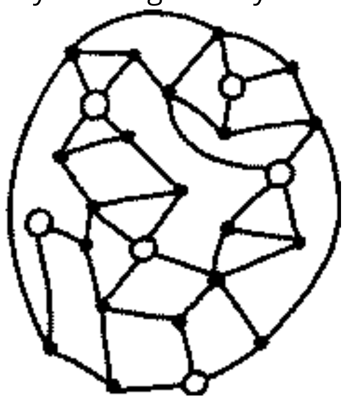
Discussion

1. Divide the students into small groups, give each group the Tourist Town map and some counters, and explain the story.

2. Show the students how to place a counter on an intersection to mark an ice-cream van, and then place counters of another color on the intersections one street away. People living at those intersections (or along the streets that come into them) are served by this ice-cream van.

3. Have the students experiment with different positions for the vans. As they find configurations that serve all houses, remind them that vans are expensive and the idea is to have as few of them as possible. It is obvious that the conditions can be met if there are enough vans to place on all intersections—the interesting question is how few you can get away with.

4. The minimum number of vans for Tourist Town is six, and a solution is shown here. But it is very difficult to find this solution! After some time, tell the class that six vans suffice and challenge them to find a way to place them. This is still quite a hard problem: many groups will eventually give up. Even a solution using eight or nine vans can be difficult to find.

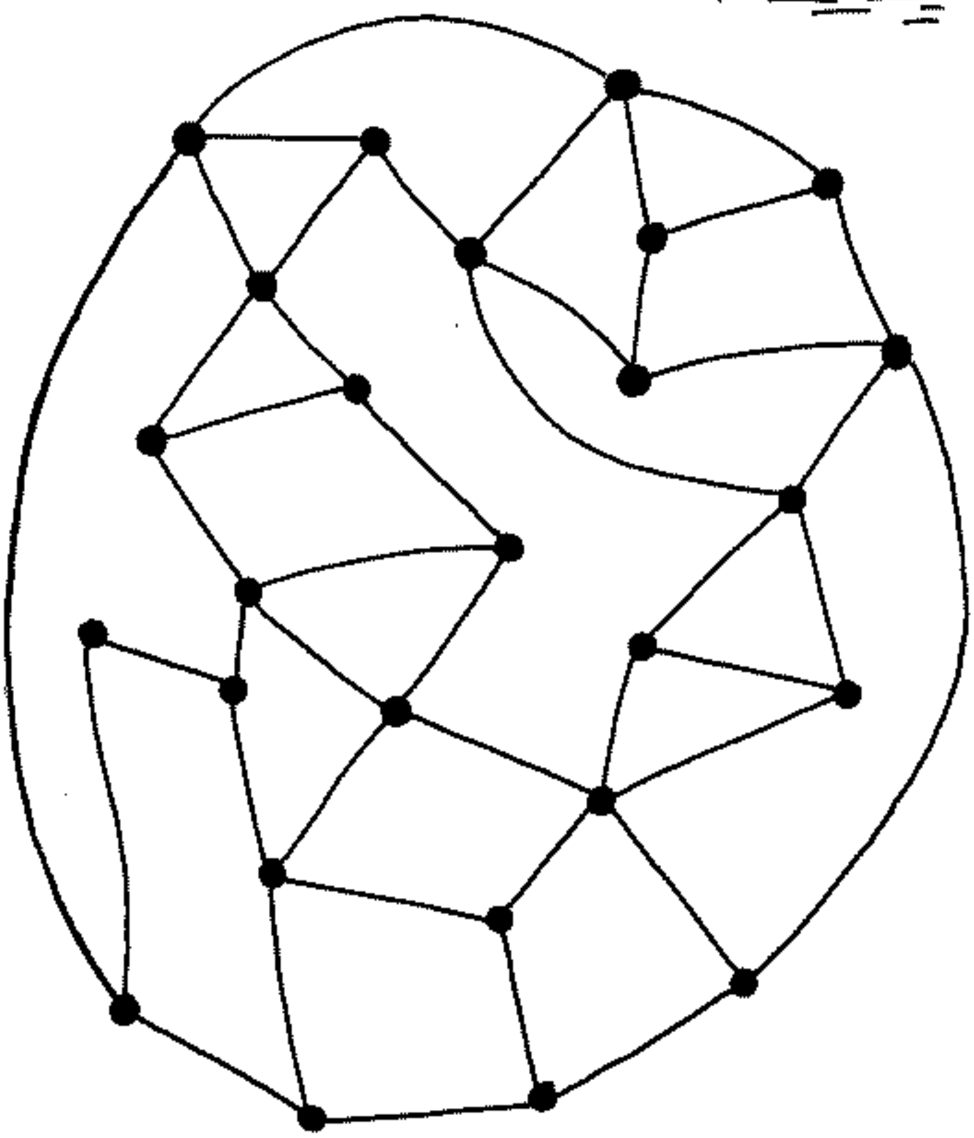


5. The map of Tourist Town was made by starting with the six map pieces at the bottom of the *Ice Cream Vans solution* worksheet, each of which obviously requires only one ice-cream van, and connecting them together with lots of streets to disguise the solution. The main thing is not to put any links between the solution intersections (the open dots), but only between the extra ones (the solid dots). Show the class this technique on the board or using a projector.

6. Get the students to make their own difficult maps using this strategy. They may wish to try them on their friends and parents–they will find that they can create puzzles that they can solve but others can't! These are examples of what is called a "one-way function": it's easy to come up with a puzzle that is very difficult to solve—unless you're the one who created it in the first place. One-way functions play a crucial role in cryptography (see Activities 17 and 18).

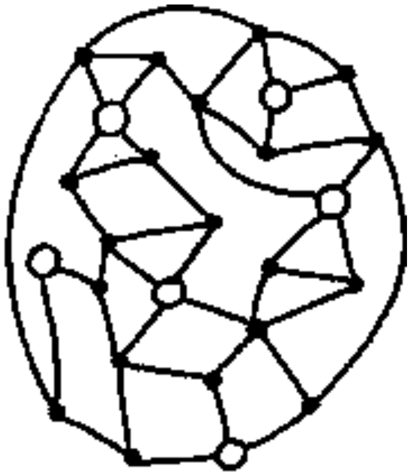
Worksheet Activity: Ice Cream Vans

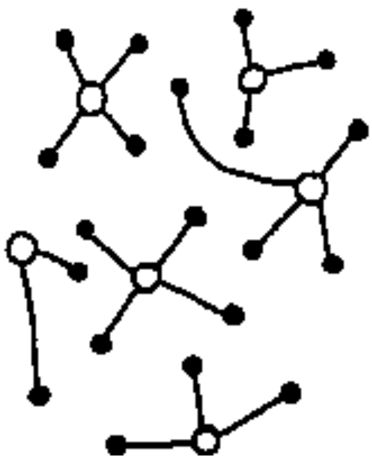
Work out how to place ice-cream vans on the street intersections so that every other intersection is connected to one that has a van on it.



Worksheet Activity: Ice Cream Vans Solution

Display this to the class to show how the puzzle was constructed.





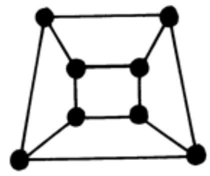
Variations and extensions

There are all sorts of situations in which one might be faced with this kind of problem in town planning: locating mailboxes, wells, fire-stations, and so on. And in real life, the map won't be based on a trick that makes it easy to solve. If you really have to solve a problem like this, how would you do it?

There is a very straightforward way: consider all possible ways of placing ice-cream vans and check them to see which is best. With the 26 street corners in Tourist Town, there are 26 ways of placing one van. It's easy to check all 26 possibilities, and it's obvious that none of them satisfies the desired condition. With two vans, there are 26 places to put the first, and, whichever one is chosen for the first, there are 25 places left to put the second (obviously you wouldn't put both vans at the same intersection): 26 × 25 = 650 possibilities to check. Again, each check is easy, although it would be very tedious to do them all. Actually, you only need to check half of them (325), since it doesn't matter which van is which: if you've checked van 1 at intersection A and van 2 at intersection B then there's no need to check van 1 at B and van 2 at A. You could carry on checking three vans (2600 possibilities), four vans (14950 possibilities), and so on. Clearly, 26 vans are enough since there are only 26 intersections and there's no point in having more than one van at the same place. Another way of assessing the number of possibilities is to consider the total number of configurations with 26 intersections and any number of vans. Since there are two possibilities for each street corner—it may or may not have a van—the number of configurations is 2^{26}, which is 67,108,864.

This way of solving the problem is called a "brute-force" algorithm, and it takes a long time. It's a widely held misconception that computers are so fast they can solve just about any problem quickly, no matter how much work it involves. But that's not true. Just how long the brute-force algorithm takes depends on how quick it is to check whether a particular configuration is a solution. To check this involves testing every intersection to find the distance of the nearest van. Suppose that an entire configuration can be tested in one second. How long does it take to test all 2^{26} possibilities for Tourist Town? (Answer: 2^{26} is about 67 million; there are 86,400 seconds in a day, so 2^{26} seconds is about 777 days, or around two years.) And suppose that instead of one second, it took just one thousandth of a second to check each particular configuration. Then the same two years would allow the computer to solve a 36-intersection town, because 2^{36} is about 1000 times 2^{26}. Even if the computer was a million times faster, so that one million configurations could be checked every second, then it would take two years to work on a 46-intersection town. And these are not very big towns! (How many intersections are there in your town?)

Since the brute-force algorithm is so slow, are there other ways to solve the problem? Well, we could try the greedy approach that was so successful in the muddy city (Activity 9). We need to think how to be greedy with ice-creams—I mean how to apply the greedy approach to the ice-cream van problem. The way to do it is by placing the first van at the intersection that connects the greatest number of streets, the second one at the next most connected intersection, and so on. However, this doesn't necessarily produce a minimum set of ice-cream van positions—in fact, the most highly connected intersection in Tourist Town, which has five streets, isn't a good place to put a van (check this with the class).



Let's look at an easier problem. Instead of being asked to find a minimum configuration, suppose you were given a configuration and asked whether it was minimal or not. In some cases, this is easy. For example, this diagram shows a much simpler map whose solution is quite straightforward. If you imagine the streets as edges of a cube, it's clear that two ice-cream vans at diagonally opposite cube vertices are sufficient. Moreover, you should be able to convince yourself that it is not possible to solve the problem with fewer than two vans. It is much harder—though not impossible—to convince oneself that Tourist Town cannot be serviced by less than six vans. For general maps it is extremely hard to prove that a certain configuration of ice-cream vans is a minimal one.

What’s it all about?

One of the interesting things about the ice-cream problem is that *no-one knows* whether there is an algorithm for finding a minimum set of locations that is significantly faster than the brute-force method! The time taken by the brute-force method grows exponentially with the number of intersections—it is called an *exponential-time* algorithm. A *polynomial-time* algorithm is one whose running time grows with the square, or the cube, or the seventeenth power, or any other power, of the number of intersections. A polynomial-time algorithm will always be faster for sufficiently large maps—even (say) a seventeenth-power algorithm—since an exponentially-growing function outweighs any polynomially-growing one once its argument becomes large enough. (For example, if you work it out, whenever n is bigger than 117 then n^{17} is smaller than 2^n). Is there a polynomial-time algorithm for finding the minimum set of locations?—no-one knows, although people have tried very hard to find one. And the same is true for the seemingly easier task of checking whether a particular set of locations is minimal: the brute-force algorithm of trying all possibilities for smaller sets of locations is exponential in the number of intersections, and polynomial-time algorithms have neither been discovered nor proved not to exist.

Does this remind you of map coloring (Activity 13)? It should. The ice-cream van question, which is officially called the “minimum dominating set” problem, is one of a large number—thousands—of problems for which it is not known whether polynomial-time algorithms exist, in domains ranging from logic, through jigsaw-like arrangement problems to map coloring, finding optimal routes on maps, and scheduling processes. Astonishingly, all of these problems have been shown to be equivalent in the sense that if a polynomial-time algorithm is found for one of them, it can be converted into a polynomial-time algorithm for all the others—you might say that they stand or fall together.

These problems are called *NP-complete*. NP stands for “non-deterministic polynomial.” This jargon means that the problem could be solved in a reasonable amount of time if you had a computer that could try out an arbitrarily large number of solutions at once (that’s the non-deterministic part). You may think this is a pretty unrealistic assumption, and indeed it is. It’s not possible to build this kind of computer, since it would have to be arbitrarily large! However, the concept of such a machine is important in principle, because it appears that NP-complete problems cannot be solved in a reasonable amount of time without a non-deterministic computer.

Furthermore, this group of problems is called *complete* because although the problems seem very different—for example, map-coloring is very different from placing ice-cream vans—it turns out that if an efficient way of solving one of them is found, then that method can be adapted to solve *any* of the problems. That's what we meant by "standing or falling together."

There are thousands of NP-complete problems, and researchers have been attacking them, looking for efficient solutions, for several decades without success. If an efficient solution had been discovered for just one of them, then we would have efficient solutions for them all. For this reason, it is strongly suspected that there is no efficient solution. But proving that the problems necessarily take exponential time is the most outstanding open question in theoretical computer science—possibly in all of mathematics—today.

Further reading

Harel's book *Algorithmics* introduces several NP-complete problems and discusses the question of whether polynomial-time algorithms exist. Dewdney's *Turing Omnibus* also discusses NP-completeness. The standard computer science text on the subject is Garey & Johnson's *Computers and Intractability*, which introduces several hundred NP-complete problems along with techniques for proving NP-completeness. However, it is fairly heavy going and is really only suitable for the computer science specialist.

Activity 16

Ice roads —*Steiner trees*

Summary

Sometimes a small, seemingly insignificant, variation in the specification of a problem makes a huge difference in how hard it is to solve. This activity, like the Muddy City problem (Activity 9), is about finding short paths through networks. The difference is that here we are allowed to introduce new points into the network if that reduces the path length. The result is a far more difficult problem that is not related to the Muddy City, but is algorithmically equivalent to the cartographer's puzzle (Activity 13) and Tourist Town (Activity 14).

Curriculum Links

- ✓ Mathematics – Position and orientation
- ✓ Mathematics – Logical reasoning

Skills

- ✓ Spatial visualization
- ✓ Geometric reasoning
- ✓ Algorithmic procedures and complexity

Ages

- ✓ 7 and up

Materials

Each group of students will need

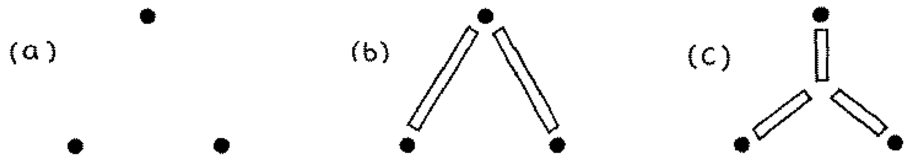
- ✓ five or six pegs to place in the ground (tent pegs are good, although a coat hanger cut into pieces which are then bent over is fine),
- ✓ several meters of string or elastic,
- ✓ a ruler or tape measure, and
- ✓ pen and paper to make notes on.

Ice Roads

Introduction

The previous activity, Tourist Town, took place in a very hot country; this one is just the opposite. In the frozen north of Canada (so the story goes), in the winter on the huge frozen lakes, snowplows make roads to connect up drill sites so that crews can visit each other. Out there in the cold they want to do a minimum of road building, and your job is to figure out where to make the roads. There are no constraints: highways can go anywhere on the snow—the lakes are frozen and covered. It's all flat.

The roads should obviously travel in straight stretches, since introducing bends would only increase the length unnecessarily. But it's not as simple as connecting all the sites with straight lines, because adding intersections out in the frozen wastes can sometimes reduce the total road length—and it's total length that's important, not travel time from one site to another.

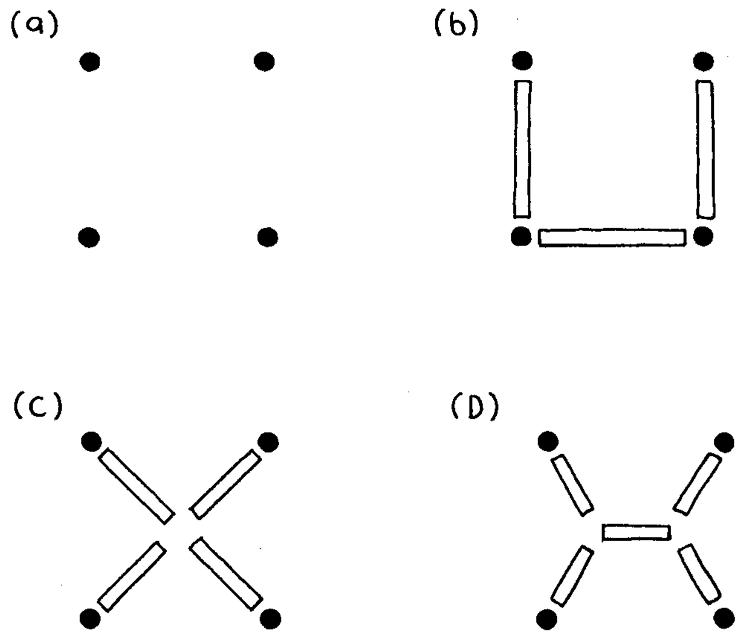


In this figure, (a) shows three drill sites. Connecting one of them to each of the others (as in (b)) would make an acceptable road network. Another possibility is to make an intersection somewhere near the center of the triangle and connect it to the three sites (c). And if you measure the total amount of road that has been cleared, this is indeed a better solution. The extra intersection is called a "Steiner" point after the Swiss mathematician

Jacob Steiner (1796–1863), who stated the problem and was the first to notice that the total length can be reduced by introducing new points. You could think of a Steiner point as a new, fictitious, drill site.

Discussion

1. Describe the problem that the students will be working on. Using the example above, demonstrate to the students that with three sites, adding a new one sometimes improves the solution by reducing the amount of road-building.



1. The students will be using four points arranged in a square, as illustrated in (a). Go outside and get each group to place four pegs in the grass in a square about 1 meter by 1 meter.

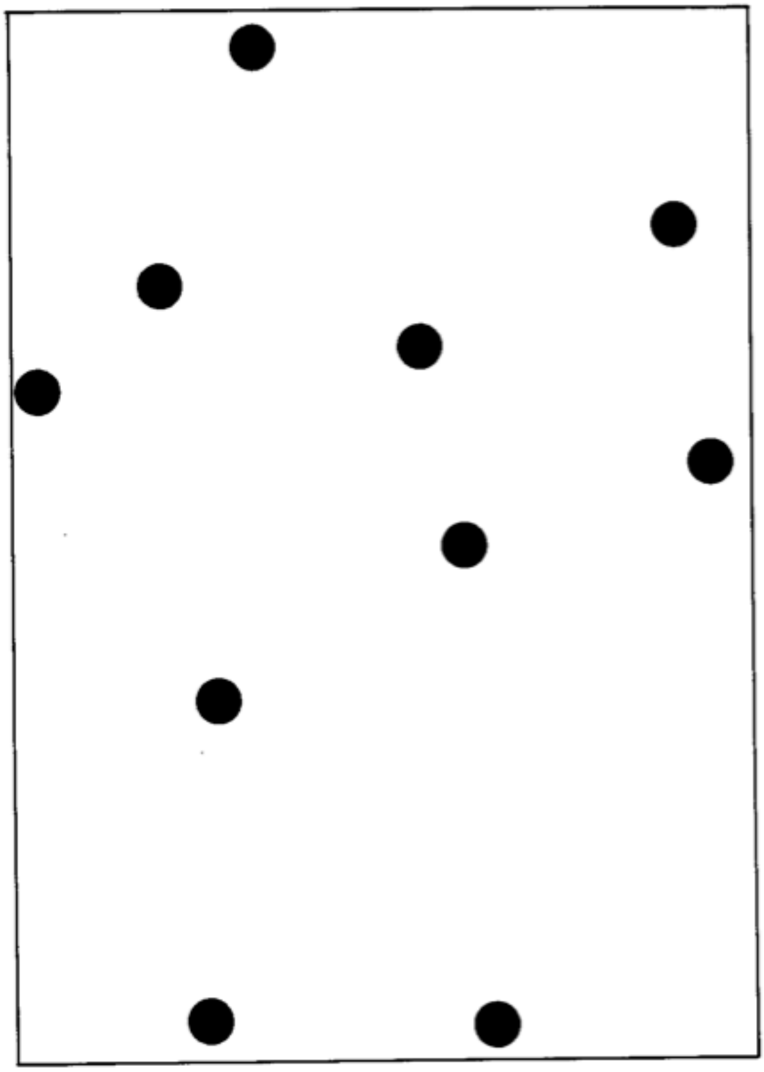
2. Get the students to experiment with connecting the pegs with string or elastic, measuring and recording the minimum total length required. At this stage they should not use any Steiner points. (The minimum is achieved by connecting three sides of the square, as in (b), and the total length is 3 meters.)

3. Now see if the students can do better by using one Steiner point. (The best place is in the center of the square, (c). Then the total length is $2\sqrt{2} = 2.83$ meters.) Suggest that they might do even better using two Steiner points. (Indeed they can, by placing the two points as in (d), forming 120 degree angles between the incoming roads. The total length is then $1 + \sqrt{3} = 2.73$ meters.)

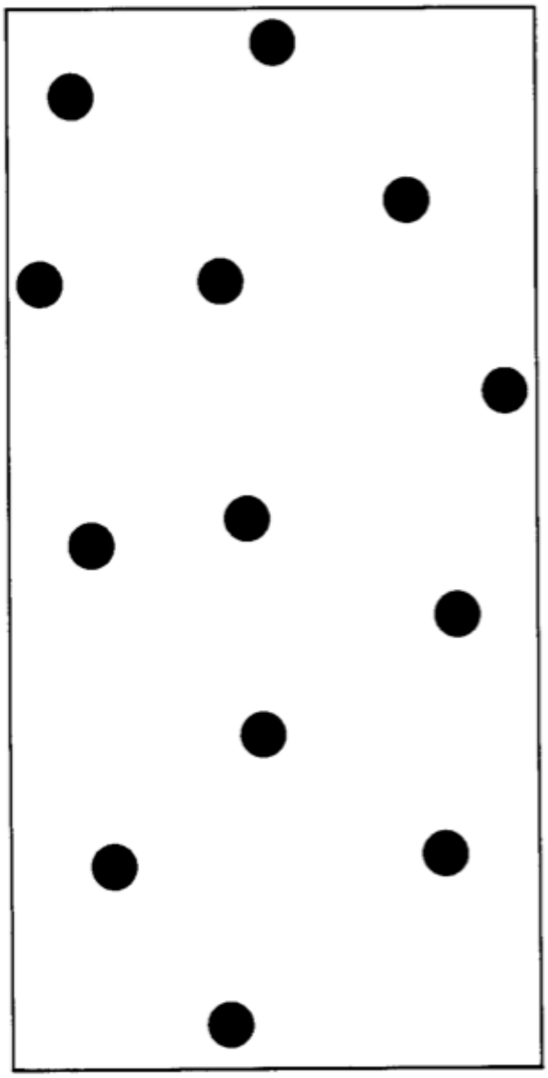
4. Can the students do better with three Steiner points? (No – two points are best, and no advantage is gained by using more.)

5. Discuss with the students why these problems seem hard. (It’s because you don’t know where to put the Steiner points, and there are lots of possibilities to try out.)

Worksheet Activity: Steiner Tree Example 1

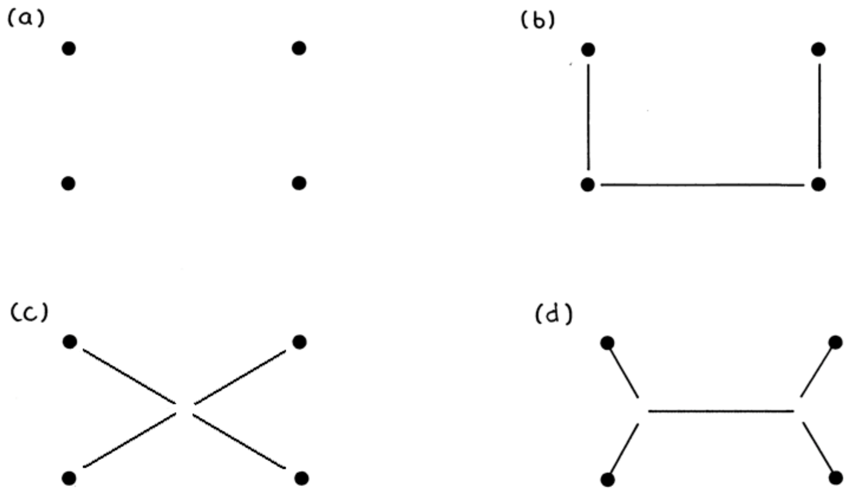


Worksheet Activity: Steiner Tree Example 2



page 132

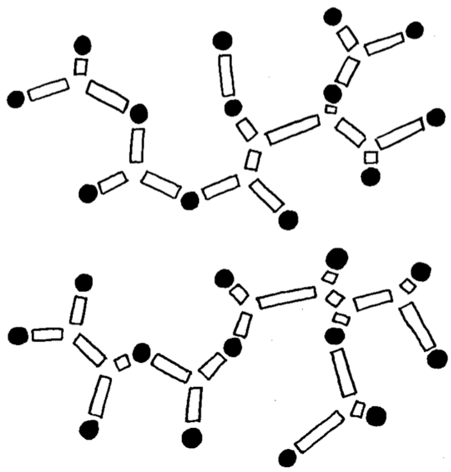
Variations and extensions



1. An interesting experiment for groups that finish the original activity early is to work with a rectangle about 1 meter by 2 meters (a). The students will find that adding one Steiner point makes things worse, but two give an improved solution. (The lengths are 4 meters for (b), $2\sqrt{5} = 4.47$ meters for (c), and $2 + \sqrt{3} = 3.73$ meters for (d).) See if they can figure out why the one-point configuration does so much worse for rectangles than for squares. (It's because when the square is stretched into a rectangle, the amount of stretch gets added just once into (b) and (d), but both diagonals increase in (c).)

2. Older students can work on a larger problem. Two layouts of sites to connect with ice roads are given in the worksheets. They can experiment with different solutions either using new copies of the worksheet, or by writing with removable pen on a transparency over the top of the sheet. Alternatively, the maps can be marked out on the ground using pegs. They can let the class know when they think they've set a new record for the shortest distance. (The figures on the right show the minimal solution for the first example and two possible solutions for the second, whose total length is quite similar.) The fact that there are two such similar solutions illustrates why these kinds of problem are so hard—there are so many choices about where to put the Steiner points!

Two possible Steiner trees for the second example

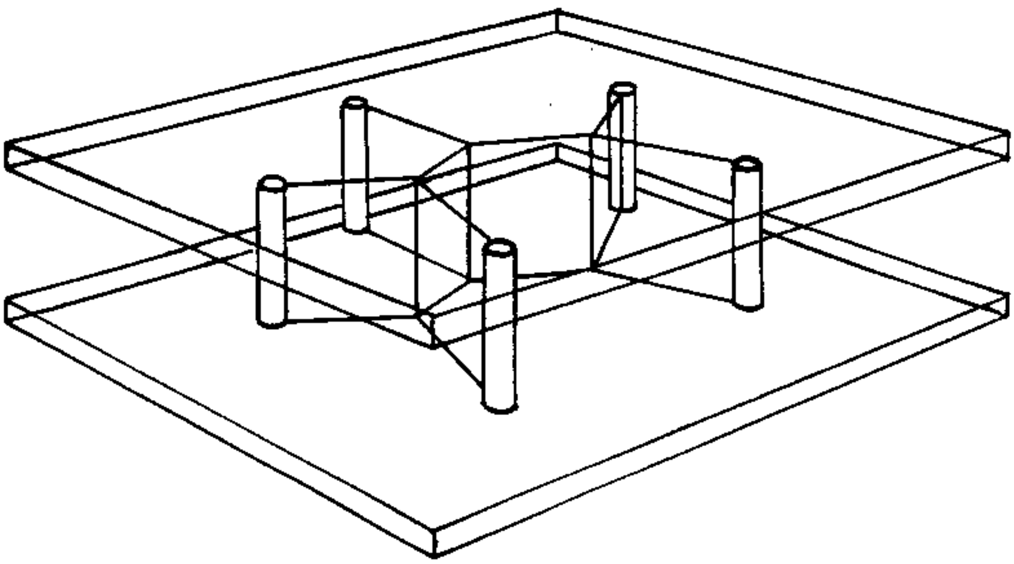


3. Ladder networks like this provide another way to extend the problem.

4. Some minimal Steiner trees for ladder networks are shown here.

   The one for a two-rung ladder is just the same as for a square. However, for a three-rung ladder the solution is quite different—as you will discover if you try to draw it out again from memory! The solution for four rungs is like that for two two-rung ladders joined together, whereas for five rungs it is more like an extension of the three-rung solution. In general, the shape of the minimal Steiner tree for a ladder depends on whether it has an even or odd number of rungs. If it is even, it is as though several two-rung ladders were joined together. Otherwise, it's like a repetition of the three-rung solution. But proving these things rigorously is not easy.

   Another interesting activity is to construct soap-bubble models of Steiner trees. You can do this by taking two sheets of rigid transparent plastic and inserting pins between them to represent the sites to be spanned, as shown here.



Now dip the whole thing into a soap solution. When it comes out, you will find that a film of soap connects the pins in a beautiful Steiner-tree network.

Unfortunately, however, it isn't necessarily a *minimal* Steiner tree. The soap film does find a configuration that minimizes the total length, but the minimum is only a local one, not necessarily a global one. There may be completely different ways of placing the Steiner points to give a smaller total length. For example, you can imagine the soap film looking like the first configuration in Extension 2 when it is withdrawn from the liquid on one occasion, and the second configuration on another.

What's it all about?

The networks that we've been working on are *minimal Steiner trees*. They're called "trees" because they have no cycles, just as the branches on a real tree grow apart but do not (normally) rejoin and grow together again. They're called "Steiner" trees because new points, Steiner points, can be added to the original sites that the trees connect. And they're called "minimal" because they have the shortest length of any tree connecting those sites. In the Muddy City (Activity 9) we learned that a network connecting a number of sites that minimizes the total length is called a minimal spanning tree: Steiner trees are just the same except that new points can be introduced.

It's interesting that while there is a very efficient algorithm for finding minimal spanning trees (Activity 14)—a greedy one that works by repeatedly connecting the two closest so-far-unconnected points—there is no general efficient solution to the minimal Steiner problem. Steiner trees are much harder because you have to decide where to put the extra points. In fact, rather surprisingly, the difficult part of the Steiner tree problem is not in determining the precise location of the Steiner points, but in deciding *roughly* where to put them: the difference between the two solutions to Example 2, for example. Once you know what regions to put the new points in, fine-tuning them to the optimum position is relatively simple. Soap films do that very effectively, and so can computers.

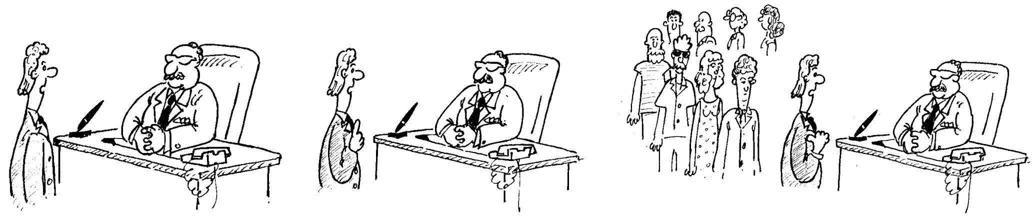
Finding minimal Steiner trees is part of a story that involved saving big money in the telephone business. Before 1967, when corporate customers in the US operated large private telephone networks, they leased the lines from a telephone company. The amount they are billed is not calculated on the basis of how the wires are actually used, but on the basis of the shortest network that would suffice. The reasoning is that the customer shouldn't have to pay extra just because the telephone company uses a round-about route. Originally, the algorithm that calculated how much to charge worked by determining the minimal spanning tree. However, around 1967 it was noticed by a customer—an airline, in fact, with three major hubs—that if they requested a fourth hub at an intermediate point then the total length of the network would be reduced. The telephone company was forced to reduce charges to what they would have been if there was a telephone exchange at the Steiner point! Although, for typical configurations, the minimal Steiner tree is only 5% or 10% shorter than the minimal spanning tree, this can be a worthwhile saving when large amounts of money are involved. The Steiner tree problem is sometimes called the "shortest network problem" because it involves finding the shortest network that connects a set of sites.

If you have tackled the two preceding activities, the cartographer's puzzle and tourist town, you will not be surprised to hear that the minimal Steiner tree problem is NP-complete. As the number of sites increases, so does the number of possible locations for Steiner points, and trying all possibilities involves an exponentially-growing search. This is another of the thousands of problems for which it simply isn't known whether exponential search is the best that can be done, or whether there is an as-yet-undiscovered polynomial-time algorithm. What is known, however, is that if a polynomial-time algorithm is found for this problem, it can be turned into a polynomial-time algorithm for graph coloring, for finding minimal dominating sets—and for all the other problems in the NP-complete class.

We explained at the end of the previous activity that the "NP" in NP-complete stands for "non-deterministic polynomial," and "complete" refers to the fact that if a polynomial-time algorithm is found for one of the NP-complete problems it can be turned into polynomial-time algorithms for all the others. The set of problems that are solvable in polynomial time is called P. So the crucial question is, do polynomial-time algorithms exist for NP-complete problems—in other words, is P = NP? The answer to this question is not known, and it is one of the great mysteries of modern computer science.

Problems for which polynomial-time algorithms exist—even though these algorithms might be quite slow—are called "tractable." Problems for which they do not are called "intractable," because no matter how fast your computer, or how many computers you use together, a small increase in problem size will mean that they can't possibly be solved in practice. It is not known whether the NP-complete problems—which include the cartographer's puzzle, tourist town, and ice roads—are tractable or not. But most computer scientists are pessimistic that a polynomial-time algorithm for NP-complete problems will ever be found, and so proving that a problem is NP-complete is regarded as strong evidence that the problem is inherently intractable.

What can you do when your boss asks you to devise an efficient algorithm that comes up with the optimal solution to a problem, and you can't find one?—as surely happened when the airline hit upon the fact that network costs could be reduced by introducing Steiner points. If you could *prove* that there isn't an efficient algorithm to come up with the optimal solution, that would be great. But it's very difficult to prove negative results like this in computer science, for who knows what clever programmer might come along in the future and hit upon an obscure trick that solves the problem. So, unfortunately, you're unlikely to be in a position to say categorically that no efficient algorithm is possible—that the problem is intractable. But if you can show that your problem is NP-complete, then it's actually true that thousands of people in research laboratories have worked on problems that really are equivalent to yours, and also failed to come up with an efficient solution. That may not get you a bonus, but it'll get you off the hook!



"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, because no such algorithm is possible."

"I can't find an efficient algorithm, but neither can all these famous people."

What to do when you can't find an efficient algorithm: three possibilities

Of course, in real life these problems still need to be solved, and in that case people turn to *heuristics* – algorithms that don't guarantee to give the best possible solution, but do give a solution within a very small percentage of the optimal. Heuristic algorithms can be very fast, and the wastage of not finding the best possible solution can be fairly small, so they are good enough to get on with the job. It's just frustrating to know that there might be a slightly better timetable, or a slightly better layout of a network or roads.

Further reading

The cartoon is based on one in Garey and Johnson's classic book *Computers and Intractability*.

The "Computer recreations" column of *Scientific American*, June 1984, contains a brief description of how to make Steiner trees using soap bubbles, along with interesting descriptions of other analog gadgets for problem solving, including a spaghetti computer for sorting, a cat's cradle of strings for finding shortest paths in a graph, and a light-and-mirrors device for telling whether or not a number is prime. These also appear in a section about analog computers in Dewdney's *Turing Omnibus*.

Part V

Sharing secrets and fighting crime-*Cryptography*

Sharing Secrets and Fighting Crime

You've heard of spies and secret agents using hidden codes or magic invisible writing to exchange messages. Well, that's how the subject of "cryptography" started out, as the art of writing and deciphering secret codes. During the Second World War, the English built special-purpose electronic code-breaking machines and used them to crack military codes. And then computers came along and changed everything, and cryptography entered a new era. Massive amounts of computation, that would have been quite unimaginable before, could be deployed to help break codes. When people began to share computer systems with each other, there were new uses for secret passwords. When computers were linked up in networks, there were new reasons to protect information from people who would have liked to have got hold of it. When electronic mail arrived, it became important to make sure that people who sign messages are really who they say they are. Now that people can do online banking, and buy and sell goods using computers, we need secure ways of placing orders and sending cash on computer networks. And the growing threat of a terrorist attacking a computer system makes computer security ever more important.

Cryptography probably makes you think of computers storing secret passwords, and jumbling up the letters of messages so that the enemy can't read them. But the reality is very different. Modern computer systems *don't* store secret passwords, because if they did, anyone who managed to get access to them would be able to break through all the security in the system. That would be disastrous: they could make phoney bank transfers, send messages pretending to be someone else, read everyone's secret files, command armies, bring down governments. Nowadays, passwords are handled using the "one-way functions" that we talked about in Activity 14. And encryption is *not* just jumbling up the letters of messages: it's done using techniques involving really hard problems—like the "intractable" ones introduced in Part IV.

Using cryptography, you can do things that you might think are impossible. In this section you will discover a simple way to calculate the average age of the people in a group without anyone having to let anyone else know what their age is. You will find out how two people who don't trust each other can toss a coin and agree on the outcome even though they are in different cities and can't both see the coin being tossed. And you will find a way to encode secret messages that can only be decoded by one person, even though everyone knows how to encode them.

For teachers

The activities that follow provide hands-on experience with modern cryptographic techniques—which are very different from what most people conjure up when they think of secrecy and computers.

There are two key ideas. The first is the notion of a "protocol," which is a formal statement of a transaction. Protocols may bring to mind diplomats, even etiquette, but computers use them too! Seemingly difficult tasks can be accomplished by surprising simple protocols. Activity 16, which only takes a few minutes, shows how a group of people, cooperating together, can easily calculate their average age (or income), without anyone finding out any individual's age (or income). The second key idea is the role that computational complexity—intractability—can play when interacting with others through computers. Activity 17 shows how two people who don't necessarily trust each other can agree on the outcome of a coin toss when they are connected only by telephone. (This activity also introduces, as an aside, the idea of Boolean logic circuits and how to work with them.) Activity 18 shows how people can use computational techniques to encrypt messages securely, even though the method for performing the encoding is public knowledge.

Some of these activities—particularly the last one—are hard work. You will have to motivate your class by instilling into the students a sense of wonder that such things can be done at all, for the activities really do accomplish things that most people would think were impossible. It is vital to create this sense of wonder, communicate it, and pause frequently to keep it alive throughout the activity so that students do not miss the (amazing!) forest for the (perhaps rather tiresome) trees. These activities are among the most challenging and technically intricate in the book. If they turn out to be too difficult, please skip to Part VI, which has a completely different, non-technical, character.

For the technically-minded

As computers encroach upon our daily lives, the application of cryptography is potentially rather tendentious. Most people simply don't realize what modern cryptographic protocols are capable of. The result is that when large institutions—both governmental and commercial—set up systems that involve personal information, it tends to be technocrats who make the key decisions on how things are to be handled, what is to be collected, what is to be made available, and to whom. If people had a better understanding of the possibilities opened up by modern technology, they would be able to participate more actively in such decisions, and society might end up with a different information infrastructure.

This material on information-hiding protocols, cryptographic protocols, and public-key encryption is generally considered to be pretty advanced. But the ideas themselves are not difficult. It's the technicalities, not the underlying concepts that are hard to understand. In practical situations involving electronic commerce, the technicalities are buried inside computer software, which renders the new technologies of encryption very easy to use. But it's also important to understand the ideas on which they are based, in order to gain insight into what can be done.

Cryptographic systems are of great interest to governments, not just because they want to keep official communications secure, but because of concerns that encrypted communication could be used by people involved in illegal activities such as drug trafficking and terrorism. If such people use encryption then wire-tapping becomes useless unless a decryption method is available. These concerns have created a lot of debate between people concerned with law enforcement, who want to limit the strength of cryptographic systems, and civil libertarians, who are uncomfortable with the government having access to the private communications. For a while the US government has restricted the use of some cryptographic methods by deeming them to be munitions—like bombs and guns, anyone can set up a secure communication system given the right information and some technical ability, but they are dangerous in the wrong hands. At one stage there was extensive debate over the "Clipper Chip," a system that has an extra password called a *key escrow*, which is held by a government agency that allows it to decode any message encrypted by the chip. The FBI and US Justice department wanted this chip to be widely used for communications, but this has drawn considerable opposition because of threats to privacy. All sorts of cryptographic systems are technically feasible, but they aren't necessarily politically acceptable!

Cryptographic ideas have many applications other than keeping messages secret. Like verifying that messages really were sent by the people who said they sent them—this is "authentication," and without it electronic commerce is impossible. There are ways to let people vote by computer without anyone else being able to find out who they voted for—even those who run the computer system—yet still prevent people from voting more than once. And you can even play cards over the phone—which may sound silly until you realize that making business deals is a lot like playing poker.

These things sound impossible. How could you even begin to shuffle a deck of cards over the phone if you're in competition with the person at the other end and so can't trust them? How could you possibly detect that someone has intercepted a message, modified it, and then passed it off as the original? Yet if you can't do those things, you can't conduct business electronically. You *have* to prevent technically-minded criminals from forging authorizations for withdrawals from bank accounts by intercepting the phone line between a point-of-sale terminal and the bank. You *have* to prevent business competitors from wreaking havoc by generating false orders or false contracts. With modern cryptographic techniques such miracles can be done, and these activities show how.

There are many interesting books about codes and code-breaking. *Codebreakers: the inside story of Bletchley Park* edited by Hinsley and Stripp, gives first-hand accounts of how some of the first computers were used to break codes during the Second World War, significantly shortening the war and saving many lives.

Activity 17

Sharing secrets—*Information hiding protocols*

Summary

Cryptographic techniques enable us to share information with other people, yet still maintain a surprisingly high level of privacy. This activity illustrates a situation where information is shared, and yet none of it is revealed: a group of students will calculate their average age without anyone having to reveal to anyone else what their age is.

Curriculum Links

- ✓ Mathematics – Sums and averages

Skills

- ✓ Calculating an average
- ✓ Random numbers
- ✓ Cooperative tasks

Ages

- ✓ 7 years and up

Materials

Each group of students will need:

- ✓ a small pad of paper, and
- ✓ a pen.

Sharing Secrets

Introduction

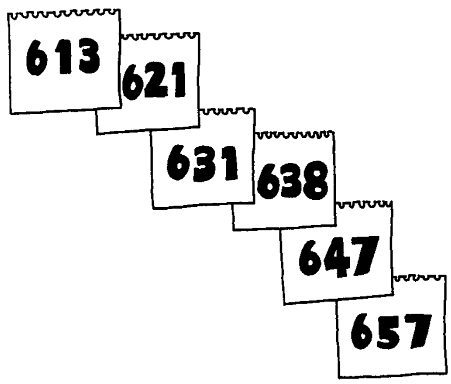
This activity involves finding the average age of a group of students, without anyone having to reveal what their age is. Alternatively, one could work out the average income (allowance) of the students in the group, or some similar personal detail. Calculating these statistics works particularly well with adults, because older people can be more sensitive about details like age and income.

You will need at least three students in the group.

Discussion

1. Explain to the group that you would like to work out their average age, without anyone telling anyone else what their age is. Ask for suggestions about how this might be done, or even whether they believe it can be done.

2. Select about six to ten students to work with. Give the pad and pen to the first student, and ask them to secretly write down a randomly chosen three-digit number on the top sheet of paper. In this example, 613 has been chosen as the random number.



3. Have the first student tear off the first page, add their age to the random number, and write it on the second sheet on the pad. The first student's age is 8, so the second sheet shows 621. They should keep the page that was torn off (and not show it to anyone.)

4. The pad is then passed to the second student, who adds their age to the number on the top, tears off the page, and writes the total on the next page. In the example, the second student is 10 years old.

5. Continue this process having a student tear off the top page and add their age to the number on it, until all the students have had the pad.

6. Return the pad to the first student. Have that student subtract their original random number from the number on the pad. In the example, the pad has been around five students, and the final number, 657, has the original number, 613, subtracted from it, giving the number 44. This number is the sum of the students' ages, and the average can be calculated by dividing by the number of students; thus the average age of our example group is 8.8 years old.

7. Point out to the students that so long as everyone destroys their piece of paper, no-one can work out an individual's age unless two people decide to cooperate.

Variations and extensions

This system could be adapted to allow secret voting by having each person add one if they are voting yes, and zero if they are voting no. Of course, if someone adds more than one (or less than zero) then the voting would be unfair, although they would be running the risk of arousing suspicion if everyone voted yes, since the number of yes votes would be more than the number of people.

What's it all about?

Computers store a lot of personal information about us: our bank balance, our social networks, how much tax we owe, how long we have held a driver's license, our credit history, examination results, medical records, and so on. Privacy is very important! But we do need to be able to share some of this information with other people. For example, when paying for goods at a store using a bank card, we recognize that the store needs to verify that we have the funds available.

Often we end up providing more information than is really necessary. For example, if we perform an electronic transaction at a store, they essentially discover who we bank with, what our account number is, and what our name is. Furthermore, the bank finds out where we have done our shopping. Banks could create a profile of someone by monitoring things like where they buy gas or groceries, how much they spend on these items each day, and when these places are visited. If we had paid by cash then none of this information would have been revealed. Most people wouldn't worry too much about this information being shared, but there is the potential for it to be abused, whether for targeted marketing (for example, sending travel advertisements to people who spend a lot on air tickets), discrimination (such as offering better service to someone whose bank usually only takes on wealthy clients), or even blackmail (such as threatening to reveal the details of an embarrassing transaction). If nothing else, people might change the way they shop if they think that someone might be monitoring them.

This loss of privacy is fairly widely accepted, yet cryptographic protocols exist that allow us to make electronic financial transactions with the same level of privacy as we would get with cash. It might be hard to believe that money can be transferred from your bank account to a store's account without anyone knowing where the money was coming from or going to. This activity makes such a transaction seem a little more plausible: both situations involve limited sharing of information, and this can be made possible by a clever protocol.

Further reading

A classic paper that highlights these issues was written by David Chaum, with the provocative title "Security without identification: transaction systems to make Big Brother obsolete." The paper is quite readable, and gives simple examples of information hiding protocols, including how completely private transactions can be made using "electronic cash." It can be found in *Communications of the ACM*, October 1985.

Activity 18

The Peruvian coin flip—*Cryptographic protocols*

Summary

This activity shows how to accomplish a simple, but nevertheless seemingly impossible task—making a fair random choice by flipping a coin, between two people who don't necessarily trust each other, and are connected only by a telephone.

Curriculum links

- ✓ Mathematics – logical reasoning
- ✓ Mathematics – Boolean logic

Skills

- ✓ Boolean Logic
- ✓ Functions
- ✓ Puzzle Solving

Ages

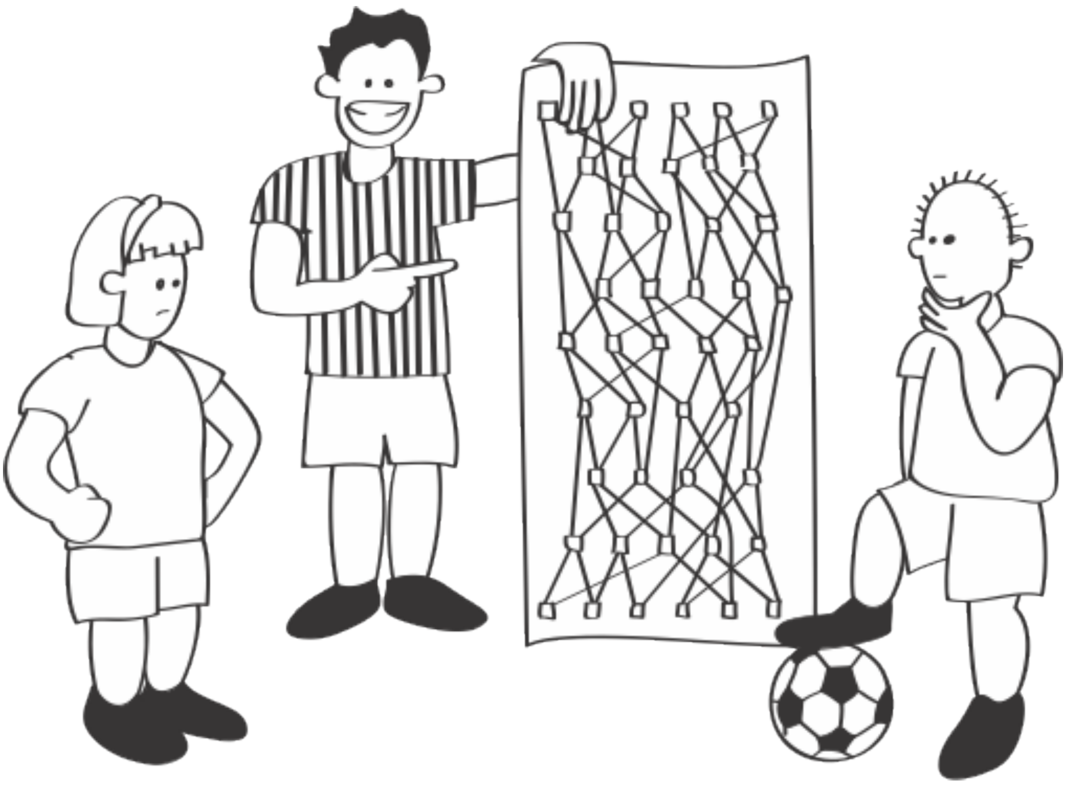
- ✓ 9 and up

Materials

Each group of students will need:

- ✓ a copy of the reproducible sheet *The Peruvian Coin Flip*
- ✓ about two dozen small buttons or counters of two different colors

The Peruvian Coin Flip



Introduction

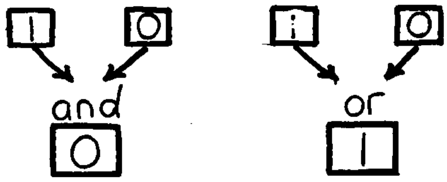
This activity was originally devised when one of the authors (MRF) was working with students in Peru, hence the name. You can customize the story to suit local conditions.

The soccer teams of Lima and Cuzco have to decide who gets to be the home team for the championship game. The simplest way would be to flip a coin. But the cities are far apart, and Alicia, representing Lima, and Benito, representing Cuzco, cannot spend the time and money to get together to flip a coin. Can they do it over the telephone? Alicia could flip and Benito could call heads or tails. But this won't work because if Benito called heads, Alicia can simply say "sorry, it was tails" and Benito would be none the wiser. Alicia is not naturally deceitful but this, after all, is an important contest and the temptation is awfully strong. Even if Alicia were truthful, would Benito believe that if he lost?

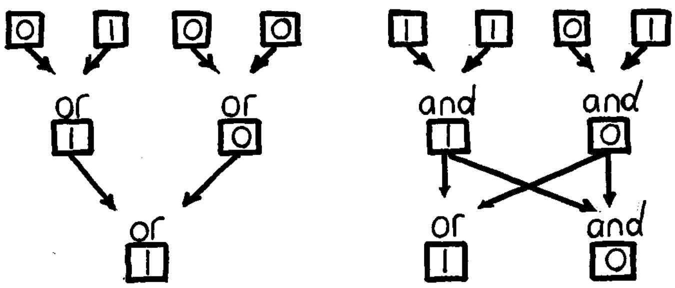
Students will get more out of this activity if they have learned binary number representation (see Activity 1, Count the dots), the concept of parity (see Activity 4, Card flip magic), and have seen the example of one-way functions in Activity 15, Tourist Town.

This is what they decide to do. Working together, they design a circuit made up of *and*-gates and *or*-gates, as explained below. In principle they can do this over the phone, although admittedly in practice it could turn out to be more than a little tedious (email would work too!) During the construction process, each has an interest in ensuring that the circuit is complex enough that the other will be unable to cheat. The final circuit is public knowledge.

Discussion

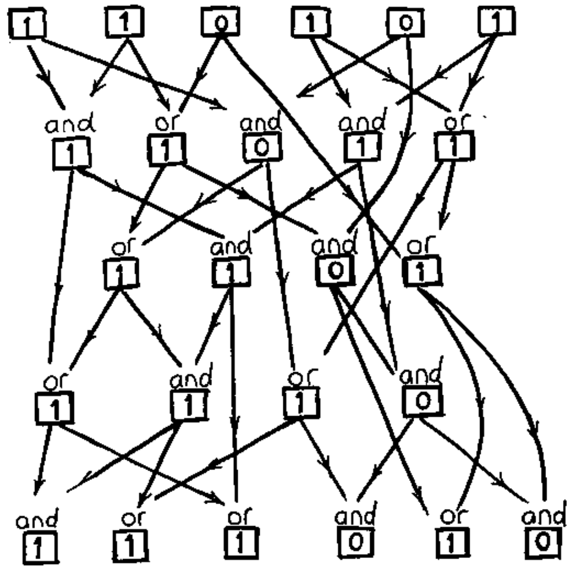


The rules of *and*-gates and *or*-gates are simple. Each "gate" has two inputs and one output. Each of the inputs can be either a 0 or a 1, which can be interpreted as *false* and *true*, respectively. The output of an *and*-gate is one (*true*) only if both inputs are one (*true*), and zero (*false*) otherwise. For example, the *and*-gate in has a one and a zero on its inputs (at the top), so the output (the square at the bottom) is a zero. The output of an *or*-gate is one (*true*) if either (or both) of the inputs is one (*true*), and zero (*false*) only if both the inputs are zero. Thus the output of the *or*-gate is a one for the inputs zero and one.



The output of one gate can be connected to the input of another (or several others) to produce a more complicated effect. For example, in the left-hand circuit the outputs from two *or*-gates are connected to the inputs of a third *or*-gate, with the effect that if any of the four inputs is a one then the output will be a one. In the right-hand circuit the outputs of each of the top two *and*-gates feeds into the lower two gates, so the whole circuit has two values in its output.

For the Peruvian coin flip we need even more complex circuits. The circuit on the worksheet has six inputs and six outputs. Here is a worked example for



one particular set of input values.

The way that this circuit can be used to flip a coin by telephone is as follows. Alicia selects a random input to the circuit, consisting of six binary digits (zeros or ones), which she keeps secret. She puts the six digits through the circuit and sends Benito the six bits of output. Once Benito has the output, he must try to guess whether Alicia's input has an even or an odd number of ones—in other words, she must guess the *parity* of Alicia's input. If the circuit is complex enough then Benito won't be able to work out the answer, and his guess will have to be a random choice (in fact, he could even toss a coin to choose!) Benito wins—and the playoff is in Cuzco—if his guess is correct; Alicia wins—and the playoff is in Lima—if Benito guesses incorrectly. Once Benito has told Alicia his guess, Alicia reveals her secret input so that Benito can confirm that it produces the claimed output.

1. Divide the students into small groups, give each group the circuit and some counters, and explain the story. The situation will probably be more meaningful to the students if they imagine one of their sports captains organizing the toss with a rival school. Establish a convention for the counter colors—red is 0, blue is 1, or some such—and have the students mark it on the legend at the top of the sheet to help them remember.

2. Show the students how to place counters on the inputs to show the digits that Alicia chooses. Then explain the rules of *and*-gates and *or*-gates,

which are summarized at the bottom of the sheet (consider getting the students to color these in).

3. Show how to work through the circuit, placing counters at the nodes, to derive the corresponding output. This must be done accurately and takes some care; The table (which should *not* be given to the students) shows the output for each possible input for your own reference in case of any doubt.

Input	000000	000001	000010	000011	000100	000101	000110	000111
Ouput	000000	010010	000000	010010	010010	010010	010010	010010
Input	001000	001001	001010	001011	001100	001101	001110	001111
Ouput	001010	011010	001010	011010	011010	011010	011010	011111
Input	010000	010001	010010	010011	010100	010101	010110	010111
Ouput	001000	011010	001010	011010	011010	011010	011010	011111
Input	011000	011001	011010	011011	011100	011101	011110	011111
Ouput	001010	011010	001010	011010	011010	011010	011010	011111
Input	100000	100001	100010	100011	100100	100101	100110	100111
Ouput	000000	010010	011000	011010	010010	010010	011010	011010
Input	101000	101001	101010	101011	101100	101101	101110	101111
Ouput	001010	011010	011010	011010	011010	011010	011010	011111
Input	110000	110001	110010	110011	110100	110101	110110	110111
Ouput	001000	011010	011010	011010	011010	111010	011010	111111
Input	111000	111001	111010	111011	111100	111101	111110	111111
Ouput	001010	011010	011010	011010	011010	111010	011010	111111

4. Now each group should elect an Alicia and a Benito. The group can split in half and each half side with Alicia or Benito respectively. Alicia should choose a random input for the circuit, calculate the output, and tell it to Benito. Benito guesses the parity of the input (whether it has an odd or even number of ones in it). It should become evident during this process that Benito's guess is essentially random. Alicia then tells everyone what the input was, and Benito wins if she guessed the correct parity. Benito can verify that Alicia's didn't change her chosen input by checking that it gives the correct output from the circuit.

At this point the coin toss has been completed.

Benito can cheat if, given an output, he can find the input that produced it. Thus it is in Alicia's interests to ensure that the function of the circuit is *one-way*, in the sense discussed in Activity 14, to prevent Benito cheating. A one-way function is one for which the output is easy to calculate if you know what the input is, but the input is very difficult to calculate for a given output.

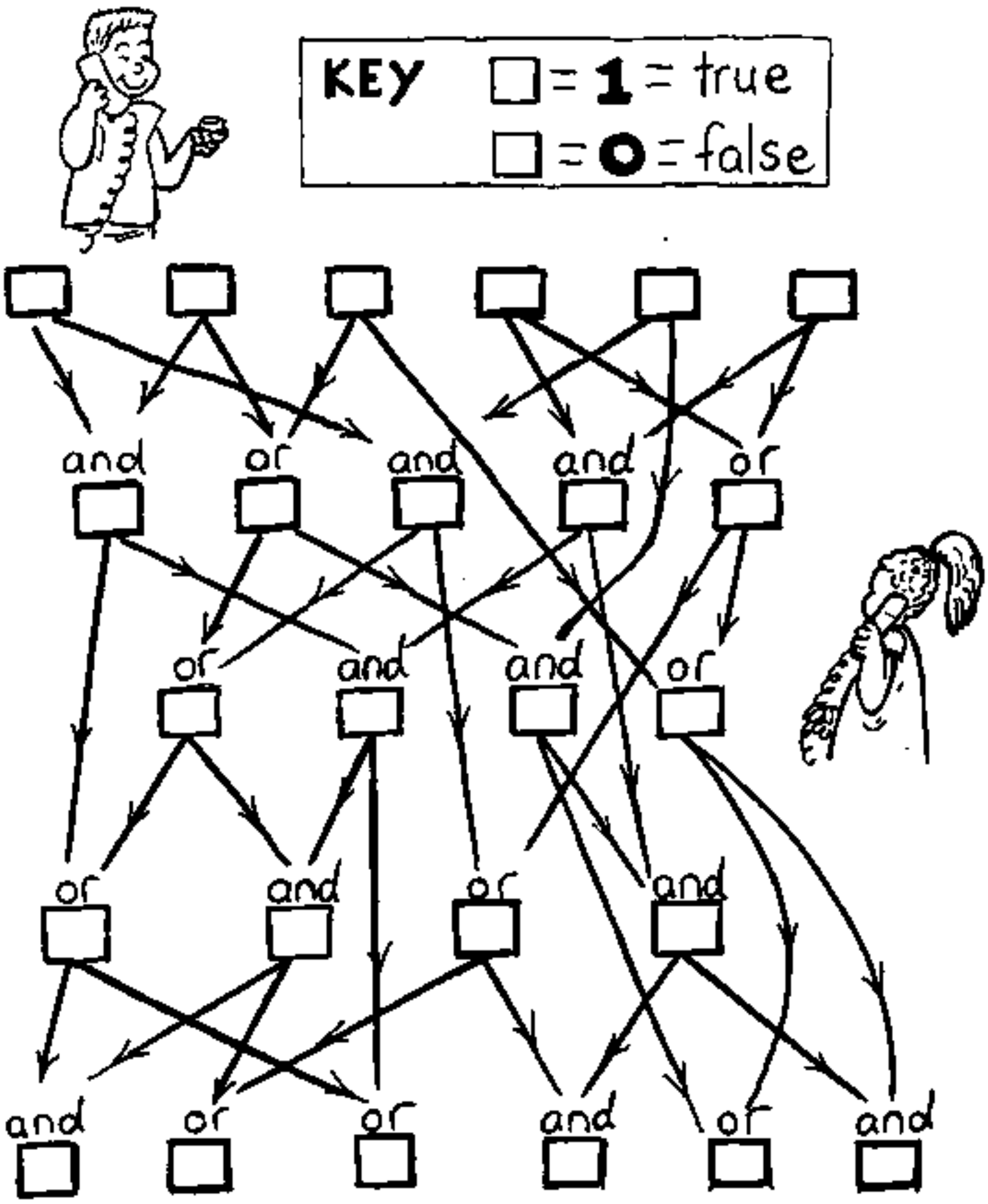
Alicia can cheat if she can find two inputs of opposite parity that produce the same output. Then, whichever way Benito guesses, Alicia can reveal the input that shows him to be wrong. Thus it is in Benito's interests to ensure that the circuit does not map many different inputs to the same output.

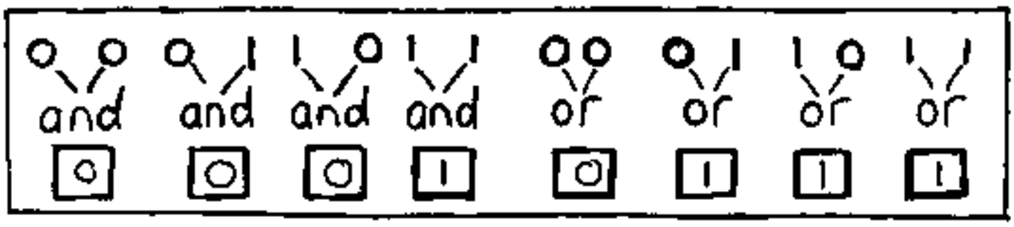
5. See if the students can find a way for Alicia or Benito to cheat. From the first line of the table you can see that several different inputs generate the output 010010—for example, 000001, 000011, 000101, etc. Thus if Alicia declares the output 010010, she can choose input 000001 if Benito guesses that the parity is even, and 000011 if he guesses that it is odd.

   With this circuit, it is hard for Benito to cheat. But if the output happens to be 011000, then the input must have been 100010—there is no other possibility (you can see this by checking right through the table). Thus if this is the number that Alicia happens to come up with, Benito can guess even parity and be sure of being correct. A computer-based system would use many more bits, so there would be too many possibilities to try (each extra bit doubles the number of possibilities).

6. Now ask the groups of students to devise their own circuits for this game. See if they can find a circuit that makes it easy for Alicia to cheat, and another that makes it easy for Benito to cheat. There is no reason why the circuit has to have six inputs, and it may even have different numbers of inputs and outputs.

Worksheet Activity: The Peruvian Coin Flip



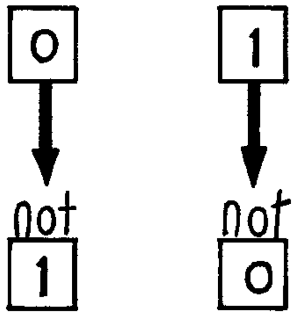


Choose some inputs for this circuit and work out what the outputs are.

Variations and extensions

1. An obvious problem in practice is the cooperation that is needed to construct a circuit acceptable to both Alicia and Benito. This might make the activity fun for the kids, but is likely to render the procedure inoperable in practice—particularly over the phone! However, there is a simple alternative in which Alicia and Benito construct their circuits independently and make them publicly available. Then Alicia puts her secret input through *both* circuits, and joins the two outputs together by comparing corresponding bits and making the final output a one if they are equal and zero otherwise. In this situation, neither participant can cheat if the other doesn't, for if just one of the circuits is a one-way function then the combination of them both is also a one-way function.

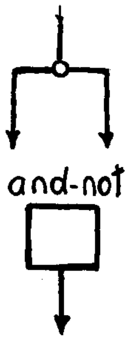
The next two variations relate not to cryptographic protocols or the coin-tossing problem *per se*, but rather to the idea of circuits constructed out of *and* and *or* gates. They explore some important notions in the fundamentals not only of computer circuits, but of logic itself. This kind of logic is called Boolean algebra named after the mathematician George Boole (1815-64).



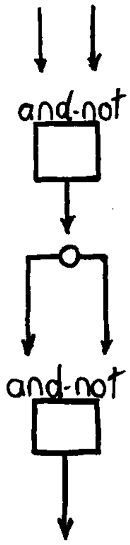
2. The students may have noticed that the all-zero input, 000000, is bound to produce the all-zero output, and likewise the all-one input 111111 is bound to produce the all-one output. (There may be other inputs that produce these outputs as well; indeed, there are for the example circuit—000010 produces all zeros, while 110111 produces all ones.) This is a consequence of the fact that the circuits are made up of *and* and *or* gates. By adding a *not*-gate, which takes just one input and produces the reverse as output (i.e. 0 → 1 and 1 → 0), the students can construct circuits that don't have this property.

3. Two other important kinds of gate are *and-not* and *or-not* (usually abbreviated to *nand* and *nor* respectively), which are like *and* and *or* but followed by a *not*. Thus a *and-not* b is *not* (a *and* b). These do not allow any functionally different circuits to be achieved, since their effect can always be obtained with the corresponding *and* or *or* gate, followed by *not*. However, they have the interesting property that all other gate types can be made out of *and-not* gates, and also out of *or-not* gates.

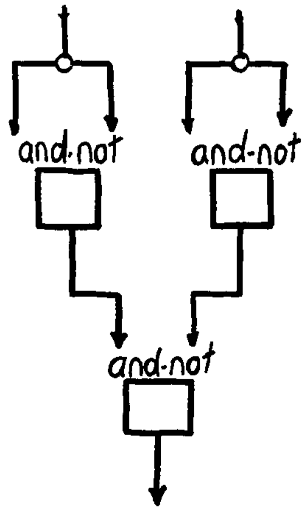
   Having introduced *and-not* and *or-not*, challenge the students to discover whether any of the gates can be made from other gates connected together, and further, if they can be made from just one type of gate connected together. The figure below shows how the three basic gates, *not*, *and*, and *or*, can be constructed from *and-not* gates, in the top row, and *or-not* gates, in the bottom row.



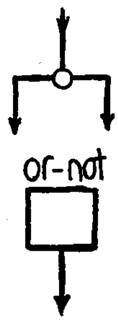
(a)



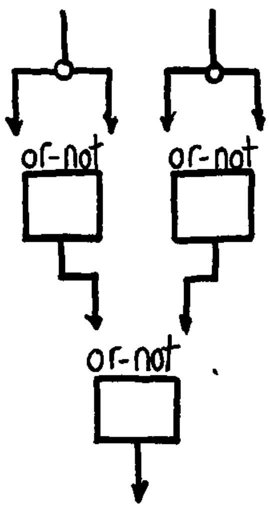
(b)



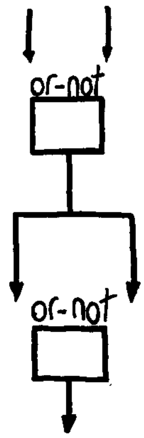
(c)



(d)



(e)



(f)

What's it all about?

Recent years have seen huge increases in the amount of commerce being conducted over computer networks, and it is essential to guarantee secure interchange of electronic funds, confidential transactions, and signed, legally binding, documents. The subject of *cryptography* is about communicating in secure and private ways. Several decades ago, computer science researchers discovered the counter-intuitive result that secrecy can be guaranteed by techniques that ensure that certain information is kept *public*. The result is the so-called "public key cryptosystem" of Activity 18, Kid Krypto, which is now widely used as the main secure way of exchanging information. For example, you may have seen settings such as SSL (Secure Sockets Layer) or TLS (Transport Layer Security) in your web browser; these systems are based on public key systems that enable your web browser to set up a secure connection to a website such as a bank, even if someone is eavesdropping on all the data being sent.

Cryptography is not just about keeping things secret, but about placing controls on information that limit what others can find out, and about establishing trust between people who are geographically separated. Formal rules or "protocols" for cryptographic transactions have been devised to allow such seemingly impossible things as unforgeable digital signatures and the ability to tell others that you possess a secret (like a password) without actually revealing what it is. Flipping a coin over the telephone is a simpler but analogous problem, which also seems, on the face of it, to be impossible.

In a real situation, Alicia and Benito would not design a circuit themselves, but acquire a computer program that does the work internally. Probably neither would be interested in the innards of the software. But both would want to rest assured that the other is unable to influence the outcome of the decision, no matter how good their computer skills and how hard they tried.

In principle, any disputes would have to be resolved by appeal to a neutral judge. The judge would be given the circuit, Alicia's original binary number, the output that she originally sent Benito, and the guess that Benito sent in return. Once the interchange is over, all this is public information, so both participants will have to agree that this is what the outcome was based on. The judge will be able to put Alicia's original number through the circuit and check that the output is as claimed, and therefore decide whether the decision has been made fairly. Needless to say, the very fact that there is a clear procedure to check that the rules have been followed makes it unlikely

that a dispute will arise. Compare with the situation where Alicia flips an actual coin and Benito calls heads or tails—no judge would take on that case!

A circuit as small as the one illustrated would not be much use in practice, for it is easy to come up with a table and use it to cheat. Using thirty-two binary digits in the input would provide better protection. However, even this does not *guarantee* that it is hard to cheat—that depends on the particular circuit. Other methods could be used, such as the one-way function introduced in Activity 14, Tourist Town. Methods used in practice often depend on the factoring of large numbers, which is known to be a hard problem (although, as we will learn at the end of the next activity, it is not NP-complete). It is easy to check that one number is a factor of another, but finding the factors of a large number is very time consuming. This makes it more complex for Alicia and Benito (and the judge) to work through by hand, although, as noted above, in practice this will be done by off-the-shelf software.

Digital signatures are based on a similar idea. By making public the output of the circuit for the particular secret input that she has chosen, Alicia is effectively able to prove that she is the one who generated the output—for, with a proper one-way function, no-one else can come up with an input that works. No-one can masquerade as Alicia! To make an actual digital signature, a more complex protocol is needed to ensure that Alicia can sign a particular message, and also to ensure that others can check that Alicia was the signatory even if she claims not to be. But the principle is the same.

Another application is playing poker over the phone, in an environment in which there is no referee to deal the cards and record both player's hands. Everything must be carried out by the players themselves, with recourse to a judge at the end of the game in the event of a dispute. Similar situations arise in earnest with contract negotiations. Obviously, players must keep their cards secret during the game. But they must be kept honest—they must not be allowed to claim to have an ace unless they actually have one! This can be checked by waiting until the game is over, and then allowing each player to inspect the other's original hand and sequence of moves. Another problem is how to deal the cards while keeping each player's hand secret until after the game. Surprisingly, it is possible to accomplish this using a cryptographic protocol not dissimilar to the coin-tossing one.

Cryptographic protocols are extremely important in electronic transactions, whether to identify the owner of an debit card, to authorize the use of a cellphone for a call, or to authenticate the sender of an email. The ability to do these things reliably is crucial to the success of electronic commerce.

Further reading

Harel’s book *Algorithmics* discusses digital signatures and associated cryptographic protocols. It also shows how to play poker over the phone, an idea that was first raised in 1981 in a chapter called “Mental poker”, in the book *The Mathematical Gardener*, edited by D.A. Klarner. *Cryptography and data security* by Dorothy Denning is an excellent computer science text on cryptography. Dewdney's *Turing Omnibus* has a section on Boolean logic that discusses the building blocks used for the circuits in this activity.

Activity 19

Kid Krypto—*Public-key encryption*

Summary

Encryption is the key to information security. And the key to modern encryption is that using only public information, a sender can lock up their message in such a way that it can only be unlocked (privately, of course) by the intended recipient.

It is as though everyone buys a padlock, writes their name on it, and puts them all on the same table for others to use. They keep the key of course—the padlocks are the kind where you just click them shut. If I want to send you a secure message, I put it in a box, pick up your padlock, lock the box and send it to you. Even if it falls into the wrong hands, no-one else can unlock it. With this scheme there is no need for any prior communication to arrange secret codes.

This activity shows how this can be done digitally. And in the digital world, instead of picking up your padlock and using it, I copy it and use the copy, leaving the original lock on the table. If I were to make a copy of a physical padlock, I could only do so by taking it apart. In doing so I would inevitably see how it worked. But in the digital world we can arrange for people to copy locks without being able to discover the key!

Sounds impossible? Read on.

Curriculum Links

✓ Technology – Public key encryption, secret codes

Skills

✓ Puzzle solving

Ages

✓ 11 years and up.

Materials

The students are divided into groups of about four, and within these groups they form two subgroups. Each subgroup is given a copy of the two maps on the worksheet *Kid Krypto Maps*. Thus for each group of students you will need:

✓ two copies of the *Kid Krypto Maps*.

✓

You will also need:

- ✓ an overhead projector transparency of *Kid Krypto Encoding*, and
- ✓ a way to annotate the diagram.

Kid Krypto

Introduction

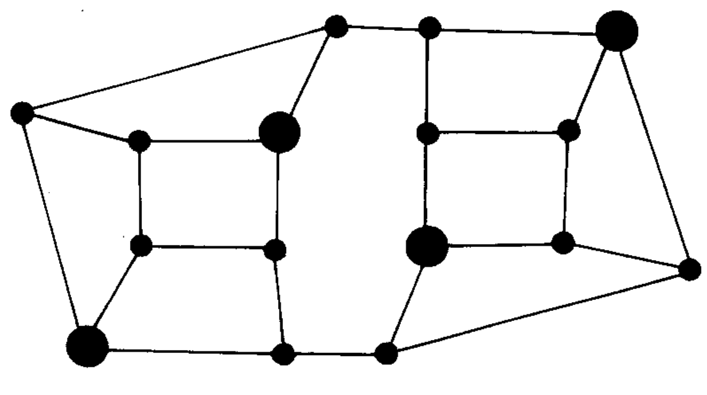
This is the most technically challenging activity in this book. While rewarding, it requires careful work and sustained concentration to complete successfully. Students should already have studied the example of one-way functions in Activity 14, Tourist Town, and it is helpful if they have completed the other activities in this section (Activity 16, Sharing Secrets, and Activity 17, the Peruvian coin flip). The activity also uses ideas covered in Activity 1, Count the dots, and Activity 5, Twenty guesses.

Amy is planning to send Bill a secret message. Normally we might think of secret messages as a sentence or paragraph, but in the following exercise Amy will send just one character — in fact, she will send one number that represents a character. Although this might seem like a simplistic message, bear in mind that she could send a whole string of such "messages" to make up a sentence, and in practice the work would be done by a computer. And sometimes even small messages are important —one of the most celebrated messages in history, carried by Paul Revere, had only two possible values. We will see how to embed Amy's number in an encrypted message using Bill's public lock so that if anyone intercepts it, they will not be able to decode it. Only Bill can do that, because only he has the key to the lock.

We will lock up messages using *maps*. Not Treasure Island maps, where X marks the spot, but street maps like the ones from Tourist Town (Activity 14), where the lines are streets and the dots are street corners. Each map has a public version—the lock—and a private version—the key.

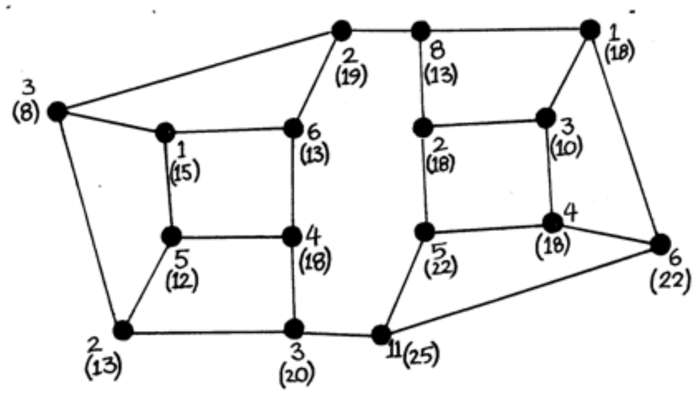
Discussion

Shown on the worksheet *Kid Krypto Encoding* is Bill's public map. It's not secret: Bill puts it on the table (or a web page) for everyone to see, or (equivalently) gives it to anyone who might want to send him a message. Amy has a copy; so has everyone else. The figure to the right shows Bill's private map. It's the same as his public map, except that



some of the street corners are marked as special by enlarging them. He keeps this version of the map secret.

This activity is best done as a class, at least to begin with, because it involves a fair amount of work. Although not difficult, this must be done accurately, for errors will cause a lot of trouble. It is important that the students realize how surprising it is that this kind of encryption can be done at all—it seems impossible (doesn't it?)—because they will need this motivation to see them through the effort required. One point that we have found highly motivating for school students is that using this method they can pass secret notes in class, and even if their teacher knows how the note was encrypted, the teacher won't be able to decode it.



1. Display Bill's public map (*Kid Krypto Encoding* worksheet). Decide which number Amy is going to send. Now place random numbers on each intersection on the map, so that the random numbers add up to the number that Amy wishes to send. This figure gives an example of such numbers as the upper (non-parenthesised) number beside each intersection. Here, Amy has chosen to send the number 66, so all the unbracketed numbers add up to 66. If necessary, you can use negative numbers to get the total down to the desired value.

2. Now Amy must calculate what to send to Bill. If she sent the map with the numbers on, that would be no good, because if it fell into the wrong hands anybody could add them up and get the message.

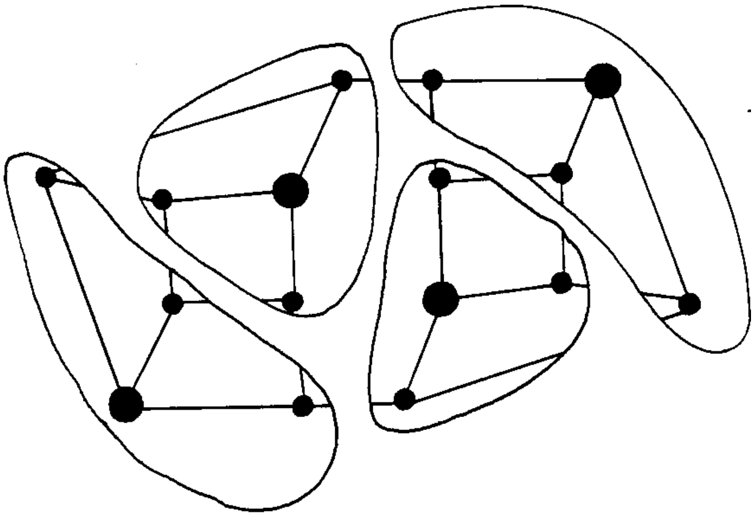
   Instead, choose any intersection, look at it and its three neighbors—four intersections in all—and total the numbers on them. Write this number at the intersection in parentheses or using a different color pen. For example, the rightmost intersection in the example public map is connected to three others, labeled 1, 4, 11, and is itself labeled 6. Thus it has a total of 22. Now repeat this for all the other intersections in the map. This should give you the numbers in parentheses.

3. Amy will send to Bill his map, with only the parenthesised numbers on it.

Erase the original numbers and the counts, leaving only the numbers that Amy sends; or write out a new map with just those numbers on it. See if any of the students can find a way to tell from this what the original message was. They won't be able to.

4. Only someone with Bill's private key can decode the message to find the message that Amy originally wanted to send. On the coded message, mark the special enlarged nodes in Bill's private map.

To decode the message, Bill looks at just the secret marked intersections and adds up the numbers on them. In the example, these intersections are labeled 13, 13, 22, 18, which add up to 66, Amy's original message.



5. How does it work? Well, the map is a special one. Suppose Bill were to choose one of the marked intersections and draw around the intersections one street distant from it, and repeat the procedure for each marked intersection. This would partition the map into non-overlapping pieces, as illustrated here. Show these pieces to the students by drawing the boundaries on the map. The group of intersections in each partition is exactly the ones summed to give the transmitted numbers for the marked intersections, so the sum of the four transmitted numbers on those intersections will be the sum of all the original numbers in the original map; that it, it will be the original message!

Phew! It seems a lot of work to send one letter. And it *is* a lot of work to send one letter—encryption is not an easy thing to do. But look at what has been accomplished: complete secrecy using a public key, with no need for any prior arrangement between the participants. You could publish your key on a noticeboard and *anyone* could send you a secret message, yet no-one could decrypt it without the private key. And in real life all the calculation is done by a software package that you acquire (typically built into your web browser), so it's only a computer that has to work hard.

Perhaps your class would like to know that they have joined the very select group of people who have actually worked through a public-key encryption example by hand—practising computer scientists would consider this to be an almost impossible task and few people have ever done it!

Now, what about eavesdropping? Bill's map is like the ones in the Tourist Town activity (Activity 14), where the marked intersections are a minimal way of placing ice-cream vans to serve all street corners without anyone having to walk more than one block. We saw in Tourist Town that it's easy for Bill to make up such a map by starting with the pieces shown in his private map, and it's very hard for anyone else to find the minimal way to place ice-cream vans except by the brute-force method. The brute-force method is to try every possible configuration with one van, then every configuration with two vans, and so on until you hit upon a solution. No-one knows whether there is a better method for a general map—and you can bet that lots of people have tried to find one!

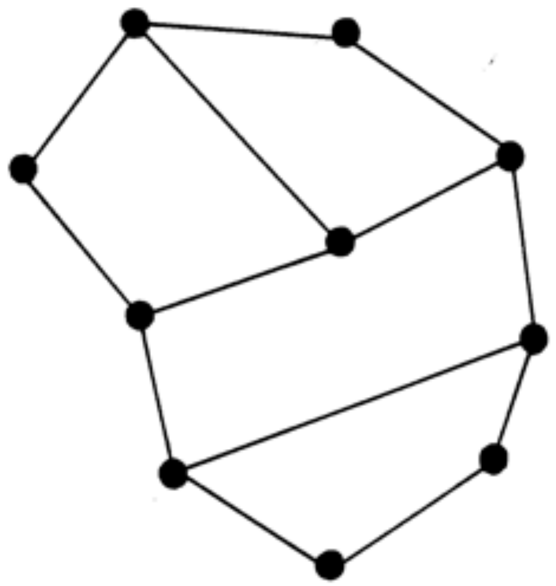
Providing Bill starts with a complicated enough map with, say, fifty or a hundred intersections, it seems like no-one could ever crack the code—even the cleverest mathematicians have tried hard and failed. (But there is a caveat: see below under *What's it all about?*)

6. Having been through one example with the whole class, divide the students into groups of, say, four. Give each pair of each group the public map on the *Kid Krypto Maps*. Each pair should choose a "message" (any integer), encode it with the public key, and give the resulting map to the other group. The other group can try to decode it, but they are unlikely to be successful until they are given (or work out!) the private map. Then give out the private map and see if they can now decode it correctly.

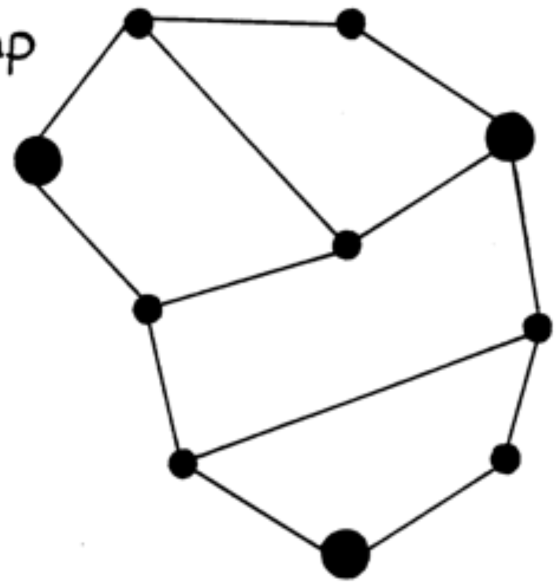
7. Now each pair can design their own map, keeping the private version secret and giving the public version to the other pair—or indeed "publishing" it on the classroom board. The principle for designing maps is just the same as was discussed in the Tourist Town activity, and extra streets can be added to disguise the solution. Just be careful not to add extra streets into any of the "special" points. That would create an intersection from which *two* ice-cream vans could be reached in one hop, which is all right for the tourist town situation but would cause havoc when encrypting. That is because the special points no longer decompose the map into *non-overlapping* pieces, as illustrated in the private map, and this is essential for the trick to work.

Worksheet Activity: Kid Kyrpto Maps

public Map



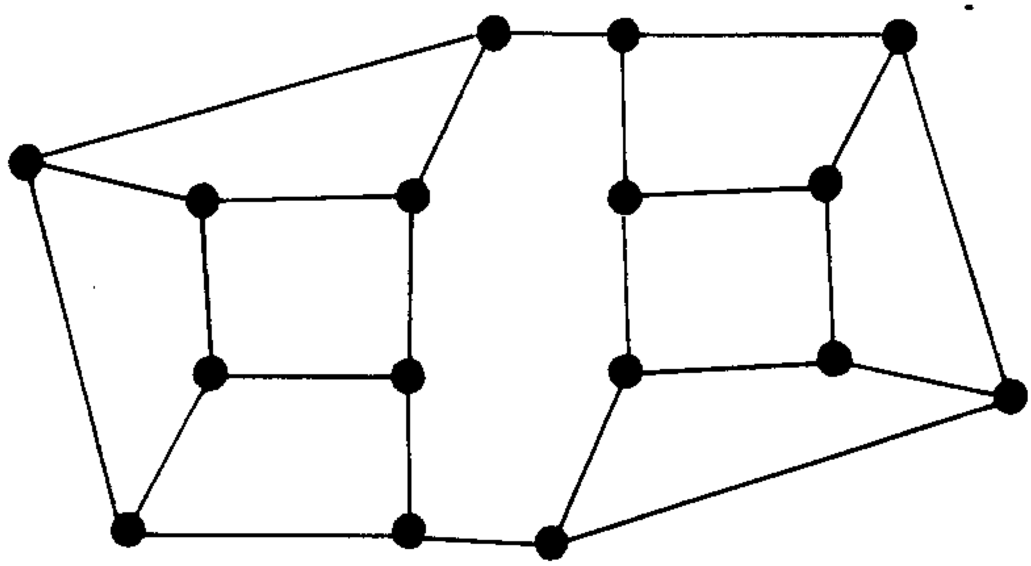
private Map



Use these maps as described in the text to encrypt and decrypt messages.

Worksheet Activity: Kid Krypto Encoding

Display this “map” to the class and use it to demonstrate the encoding of a message.



What’s it all about?

It’s clear why you might want to send secret messages over computer networks that no-one but the intended recipient could decode, no matter how clever they were or how hard they tried. And of course there are all sorts of ways in which this can be done *if* the sender and receiver share a secret code. But the clever part of public-key encryption is that Amy can send Bill a secure message without any secret prior arrangement, just by picking up his “lock” from a public place like a web page.

Secrecy is only one side of cryptography. Another is *authentication*: When Amy receives a message from Bill, how does she know that it really comes from him and not from some imposter? Suppose she receives electronic mail that says, “Darling, I’m stuck here without any money. Please put $100 in my bank account, number 0241-45-784329 -- love, Bill.” How can she know whether it really comes from Bill? Some public-key cryptosystems can be used for this, too. Just as Amy sends Bill a secret message by encoding it with his public key, he can send her a message *that only he could have generated* by encoding it with his *private* key. If Amy can decode it with Bill’s public key, then it must have come from him. Of course, anyone else could decode it too, since the key is public, but if the message is for her eyes only, Bill can then encode it a second time with Amy’s public key. This dual encoding provides both secrecy and authentication with the same basic scheme of public and private keys.

Now is the time to admit that while the scheme illustrated in this activity is very similar to an industrial-strength public-key encryption system, it is *not* in fact a secure one—even if quite a large map is used.

The reason is that although there is no known way of finding the minimal way to place ice-cream vans on an arbitrary map, and so the scheme is indeed secure from this point of view, there happens to be a completely different way of attacking it. The idea is unlikely to occur to school students, at least up to high school level, but you should at least know that it exists. You might say that the scheme we have been looking at is schoolstudent secure, but not mathematician-secure. Please ignore the next paragraph if you are not mathematically inclined!

Number the intersections on the map 1, 2, 3, ... Denote the original numbers that are assigned to intersections by b_1, b_2, b_3, ..., and the numbers that are actually transmitted by t_1, t_2, t_3, ... Suppose that intersection 1 is connected to intersections 2, 3, and 4. Then the number that is transmitted for that intersection is

$t_1 = b_1+b_2+b_3+b_4$.

Of course, there are similar equations for every other intersection—in fact, there are the same number of equations as there are unknowns b_1, b_2, b_3, ... An eavesdropper knows the public map and the numbers t_1, t_2, t_3, ... that are transmitted, and can therefore write down the equations and solve them with an equation-solving computer program. Once the original numbers have been obtained, the message is just their sum—there is actually no need ever to discover the decryption map. The computational effort required to solve the equations directly using Gaussian elimination is proportional to the cube of the number of equations, but because these equations are sparse ones—most of the coefficients are zero—even more efficient techniques exist. Contrast this with the exponential computational effort that, as far as anyone knows, is the best one can do to come up with the decryption map.

We hope you don't feel cheated! In fact, the processes involved in real public-key cryptosystems are virtually identical to what we have seen, except that the techniques they use for encoding are different—and really are infeasible to do by hand. The original public-key method, and still one of the most secure, is based on the difficulty of factoring large numbers.

What are the factors of the 100-digit number 9,412,343,607,359,262,946,971,172,136, 294,514,357,528,981,378,983,082,541,347,532,211,942,640,121,301,590,698,634,089, 611,468,911,681? Don't spend too long!

They are 86,759,222,313,428,390,812,218,077,095,850,708,048, 977 and 108,488,104,853,637,470,612,961,399,842,972,948,409,834,611,525,790,577,216,753. There are no other factors: these two numbers are prime. Finding them is quite a job: in fact, it's a several-month project for a supercomputer.

Now in a real public-key cryptosystem, Bill might use the 100-digit number as his public key, and the two factors as the private key. It would not be too difficult to come up with such keys: all you need is a way of calculating large prime numbers. Find two prime numbers that are big enough (that's not hard to do), multiply them together, and—hey presto, there's your public key. Multiplying huge numbers together is no big deal for a computer. Given the public key, no-one can find your private key, unless they have access to several months of supercomputer time. And if you're worried that they might, use 200-digit primes instead of 100-digit ones—that'll slow them down for years! The main thing is that the cost of cracking the key is higher than the value of the information it would unlock. In practice, 512-bit or larger keys are common for setting up secure connections, which is equivalent to about 155 decimal digits or more.

We still haven't been given a way to encode a message using a prime-number based public key in such a way that it can't be decoded without the private key. In order to do this, life is not quite as simple as we made out above. It's not the two prime numbers that are used as the private key and their product as the public key, instead it's numbers derived from them. But the effect is the same: you can crack the code by factoring the number. Anyway, it's not difficult to overcome these difficulties and make the scheme into a proper encryption and decryption algorithm, but let's not go into that here. This activity has already been enough work!

How secure is the system based on prime numbers? Well, factoring large numbers is a problem that has claimed the attention of the world's greatest mathematicians for several centuries, and while methods have been discovered that are significantly better than the brute-force method of trying all possible factors, no-one has come up with a really fast (that is, polynomial-time) algorithm. (No-one has proved that such an algorithm is impossible, either.) Thus the scheme appears to be not just school-student secure, but also mathematician-secure. But beware: we must be careful. Just as there turned out to be a way of cracking Bill's code without solving the Tourist Town problem, there may be a way of cracking the prime-number codes without actually factoring large numbers. People have checked carefully for this, and it seems OK.

Another worry is that if there are just a few possible messages, an interloper could encrypt each of them in turn using the public key, and compare the actual message with all the possibilities. Amy's method avoids this because there are many ways of encrypting the same message, depending on what numbers were chosen to add up to the code value. In practice, cryptographic systems are designed so that there are just too many possible messages to even begin to try them all out, even with the help of a very fast computer.

It is not known whether a fast method for solving the prime factorization problem exists. No one has managed to devise one, but also it has not been proven that a fast method is impossible. If a fast algorithm for solving this problem is found, then many currently used cryptographic systems will become insecure. In Part IV we discussed *NP-complete* problems, which stand or fall together: if one of them is efficiently solvable then they all must be. Since so much (unsuccessful) effort has been put into finding fast algorithms for these problems, they would seem like excellent candidates for use in designing secure cryptosystems. Alas, there are difficulties with this plan, and so far the designers of cryptosystems have been forced to rely on problems (such as prime factorization) that might in fact be easier to solve than the NP-complete problems—maybe a lot easier. The answers to the questions raised by all this are worth many millions of dollars to industry and are regarded as vital to national security. Cryptography is now a very active area of research in computer science.

Further reading

Harel's book *Algorithmics* discusses public-key cryptography; it explains how to use large prime numbers to create a secure public-key system. The standard computer science text on cryptography is *Cryptography and data security* by Dorothy Denning, while a more practical book is *Applied cryptography* by Bruce Schneier. Dewdney's *Turing Omnibus* describes another system for performing public key cryptography.

Part VI

The human face of computing-
Interacting with computers

The Human Face of Computing

Why are computers so hard to get along with? Many people have stories about how difficult computers are to use, how they never seem to do what you really want them to, how they keep going wrong and make ridiculous mistakes. Computers seem to be made for wizards, not for ordinary people. But they *should* be made for ordinary people, because computers are everyday tools that help us to learn, work, and play better.

The part of a computer system that you interact with is called its "user interface." It's the most important bit! Although you might think of what the program actually *does* as the main thing and the user interface as just how you get into it, a program is no good at all if you can't interact with it and make it do what you want. User interfaces are very difficult to design and build, and it has been estimated that when writing programs, far more effort goes into the interface than into any other part. Some software has excellent user interfaces, interfaces that need no complicated instructions and become almost invisible as you are drawn into using the application. But countless software products which are otherwise very good have been complete flops because they have strange user interfaces. And whole industries have been built around a clever interface idea—like the word processor or smartphones—that promotes access to computational functions which are really quite elementary in themselves.

But why do we have to have user interfaces at all? Why can't we just talk to our computers the way we do to our friends? Good question. Maybe someday we will; maybe not. But certainly not yet: there are big practical limitations on how "intelligent" computers can be today. The activities that follow will help you understand the problems of user interface design, and help you to think more clearly about the limitations of computers and be wary of the misleading hype that is often used to promote computer products.

For teachers

Computing is not so much about calculation as it is about *communication*. Computing *per se* really has no intrinsic value; it is only worthwhile if the results are somehow communicated to the world outside the computer, and have some influence there. Perhaps surprisingly, this means that computer science is less about computers and more about people – in the end, a computer is no use unless it helps people in some way. All the ideas we've looked at about how to make computers work fast and efficiently are needed only because people need computers to respond quickly, and to be economical to use.

The interface is how the computer and human communicate. And a lot of the activities in this book are about communication. *Representing data* (Part I) shows how different kinds of information can be communicated to a computer or between computers. *Representing processes* (Part III) is about how to communicate processes to a computer to tell it how to accomplish certain tasks—after all, "programming" is really only explaining to a computer, in its own language! *Cryptography* (Part V) is about how to communicate in secret, or to communicate bits of secrets without revealing all.

The activities that follow are about how people communicate with computers. While the rest of the book is based on well understood technical ideas, this part is not. That makes it both easier, in that no special knowledge is required of the students, and more difficult, in that a certain level of maturity is needed to understand what the activities are about and relate them to a broader context. These activities contain more detailed explanations than most of the others because it is necessary to give you, the teacher, enough background material to be in a position to help draw out some of the implications in class discussion.

There are two activities in this section. The first is about the area known as the "human–computer interface," commonly abbreviated to HCI. In order to "unplug" this aspect of computing without depending on prior knowledge of a particular example of a computer system, we have invented a design exercise that does not really involve computers—but does introduce fundamental principles that are used in the design of human–computer interfaces. Because human interface design is culture-dependent, there are not necessarily any "right" answers in this activity, which may frustrate some students. The second activity is about the area known as "artificial intelligence," or AI. It involves a guessing game that stimulates students into thinking about what computers can and can't do.

For the technically-minded

Human–computer interaction has become one of the hottest research areas in computer science as people realize how much the success of a software product depends on its user interface. The subject draws heavily on a wide range of disciplines outside computer science, such as psychology, cognitive science, linguistics, sociology—even anthropology. Few computer scientists have training in these areas, and HCI represents an important growth area for people who are interested in the "softer" side of the subject.

Artificial intelligence is a topic that often raises hackles and causes disputes. In this book we have tried to steer a middle path between AI afficionados who believe that intelligent machines are just around the corner, and AI sceptics who believe that machines cannot in principle be intelligent. Our goal is to encourage students to think independently about such issues, and to promote a balanced view.

The activities here draw heavily on two eminently readable books, Don Norman's *The design of everyday things* and John Haugeland's *Artificial intelligence: the very idea*, which we enthusiastically recommend if you are interested in pursuing these issues further.

Computers involve another important kind of communication, one that is not touched upon in this book: communication between people who are building a computer system. Students who learn about computers and make their way into the job market—perhaps having graduated in computer science from university—are invariably surprised by how much interpersonal communication their job entails. Computer programs are the most complex objects ever constructed by humankind, with millions or perhaps billions of intricately interlocking parts, and programming projects are tackled by close-knit teams that work together and spend a great deal of their time communicating. Once the product is complete, there is the job of communicating with customers through user manuals, courses, “help” phonelines, online support, and the like—not to mention the problem of communicating with potential customers through demonstrations, displays, and advertising. We haven’t yet found a way to realistically “unplug” for students the interpersonal communication aspect of computing, so this book doesn’t address it. But it is the kind of thing that computer professionals who are visiting a classroom may be able to describe from their own experience and bring out in discussion.

Activity 20

The chocolate factory—*Human interface design*

Summary

The aim of this activity is to raise awareness of human interface design issues. Because we live in a world where poor design is rife, we have become accustomed (resigned?) to putting up with problems caused by the artifacts we interact with, blaming ourselves ("human error," "inadequate training," "it's too complicated for me") instead of attributing the problems to flawed design. The issue is greatly heightened by computers because they have no obvious purpose—indeed, they are completely general purpose—and their appearance gives no clues about what they are for, nor how to operate them.

Curriculum Links

- ✓ Technology: Understand that technology is purposeful intervention through design.

Skills

- ✓ Design.
- ✓ Reasoning.
- ✓ Awareness of everyday objects.

Ages

- ✓ 7 and up

Materials

Each group of students will need:

- ✓ a copy of the sheets *How do you open doors?* and *Stove top*, and
- ✓ a copy of the images on the worksheet *Icons*, either displayed on a projector, shown on overhead projector transparency or on cards that can be displayed to the class, and
- ✓ one or more of the six cards on the *Icon cards* page. Cut the sheet into individual cards and divide them between the groups.

The Chocolate Factory

Introduction

The great chocolate factory is run by a race of elf-like beings called Oompa-Loompas[3]. These Oompa-Loompas have terrible memories and no written language. Because of this, they have difficulty remembering what to do in order to run the chocolate factory, and things often go wrong. Because of this, a new factory is being designed that is supposed to be very easy for them to operate.

Discussion

1. Explain the story to the students and divide them into small groups.

2. The first problem the Oompa-Loompas face is getting through the doors carrying steaming buckets of liquid chocolate. They cannot remember whether to push or pull the doors to open them, or slide them to one side. Consequently they end up banging into each other and spilling sticky chocolate all over the place. The students should fill out the "doors" worksheet *How do you open doors*. More than one box is appropriate in each case. For some of the doors (including the first one) it is not obvious how to open them, in which case the students should record what they would try first. Once they have filled out their own sheets, have the whole group discuss the relative merits of each type of door, particularly with regard to how easy it is to tell how it works, and how suitable it would be to use if you are carrying a bucket of hot chocolate. Then they should decide what kind of doors and handles to use in the factory.

3. Follow this activity with a class discussion. The table below comments briefly on each door in the worksheet. Real doors present clues in their frames and hinges as to how they open, and there are conventions about whether doors open inwards or outwards. Identify the kinds of door handles used in your school and discuss their appropriateness (they may be quite *in*appropriate!) Can you think of a door that often confuses you? Why? Do doors normally open inwards or outwards into corridors?—and why? (Answer: They open into rooms so that when you come out you won't bash the door into people walking along the corridor, although in

[3] With apologies to Roald Dahl. You'll know about the Oompa-Loompas if you've read his wonderful tale Charlie and the Chocolate Factory. If not, never mind: the plot is not relevant to this activity.

some situations they open outwards to make evacuation easier in an emergency.)

4. The key concept here is what is called the *affordances* of an object, which are its visible features—both fundamental and perceived—whose appearance indicates how the object should be used. Affordances are the kinds of operation that the object permits, or "affords." For example, it is (mostly) clear from their appearance that chairs are for sitting, tables are for placing things on, knobs are for turning, slots are for inserting things into, buttons are for pushing. On a computer interface the affordances are the shapes of buttons, text boxes, menus and so on, which give the user a clue as to how they should be used. If a button is made to look like something else, then people won't realise they can push it. This might seem obvious, but these problems aren't hard to find on digital devices.

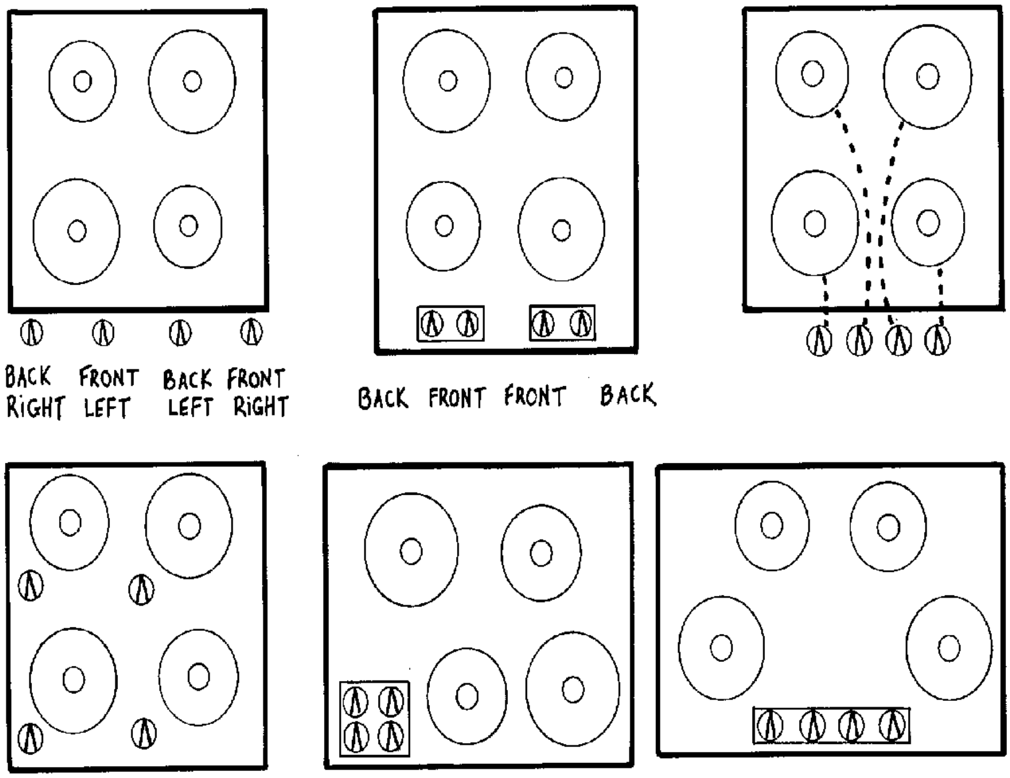
Plain door	Can't see how to open this one at all, except that since it has no handle, it must require pushing rather than pulling.	Labeled door	The label is like a tiny user manual. But should a door need a user manual? And the Oompa Loompas can't read.
Hinge door	At least you can see which is the side that opens.	Bar door	It's fairly clear that you are supposed to push the bar, but which side? Or should you pull?
Handle door	Handles like this are usually for pulling—or sliding.	Knob door	The knob shows what to grasp, but not whether to push or pull; it probably doesn't slide.
Panel door	It's clear that you push this. What else could you do?	Glass door	The small vertical bar on this side signals "pull"; the longer horizontal one on the other signals "push".
Sliding door	This one's only for sliding.		

Doors are very simple objects. Complex things may need explaining, but simple things should not. When simple objects need pictures, labels, or instructions, then design has failed.

5. The pots containing different kinds of chocolate have to cook at different temperatures. In the old chocolate factory the stoves were as shown in the *Stove top* sheet. The left-hand knob controlled the rear left heating element, the next knob controlled the front left element, the next one controlled the front right, and the right-hand knob controlled the rear right element. The Oompa-Loompas were always making mistakes, cooking the chocolate at the wrong temperature, and burning their sleeves when reaching across the elements to adjust the controls.

6. The students should recall how the controls are laid out on their cookers at home and come up with a better arrangement for the new factory.

   Follow this activity with a class discussion. This picture below shows some common arrangements. All but the one at the lower left have the controls at the front, to avoid having to reach across the elements. In the design at the top left, there are so many possible mappings from controls to



   burners (24 possibilities, in fact) that eight words of labeling are needed. The “paired” arrangement in the top center is better, with only four possible mappings (two for the left cluster and two for the right); it requires just four labeling words. The design at the top right specifies the control–burner relationship diagrammatically rather than linguistically (which is good for the Oompa-Loompas!) The lower three designs need no labels. The left-hand one has a control by each burner, which is awkward and dangerous. The other two involve relocating the burners slightly, but for different reasons: in the center design they are moved to leave room for the controls, while in the right-hand one they are rearranged to make the correspondence clear.

The key concept here is the *mapping* of controls to their results in the real world. Natural mapping, which takes advantage of physical analogies and cultural standards, leads to immediate understanding. The spatial correspondences at the bottom of the picture are good examples—they are easily learned and always remembered. Arbitrary mappings, as in the top arrangements, need to be labeled, or explained and memorized.

7. The factory is full of conveyer belts carrying pots of half-made chocolate in various stages of completion. These conveyer belts are controlled manually by Oompa-Loompas, on instructions from a central control room. The people in the control room need to be able to tell the Oompa-Loompa to stop the conveyer belt, or slow it down, or start it up again.

   In the old factory this was done with a voice system: the control room person's voice came out of a loudspeaker by the conveyer belt controls. But the factory was noisy and it was hard to hear. The groups should design a scheme that uses visual signals.

   One possibility is to put in lights to signal *Stop!*, *Slow down* and *Start up*. Students will probably work out that these should follow the normal traffic-light convention by using red for *Stop!*, yellow for *Slow down* and green for *Start up*. They should be arranged just like traffic lights, with red at the top and green at the bottom.

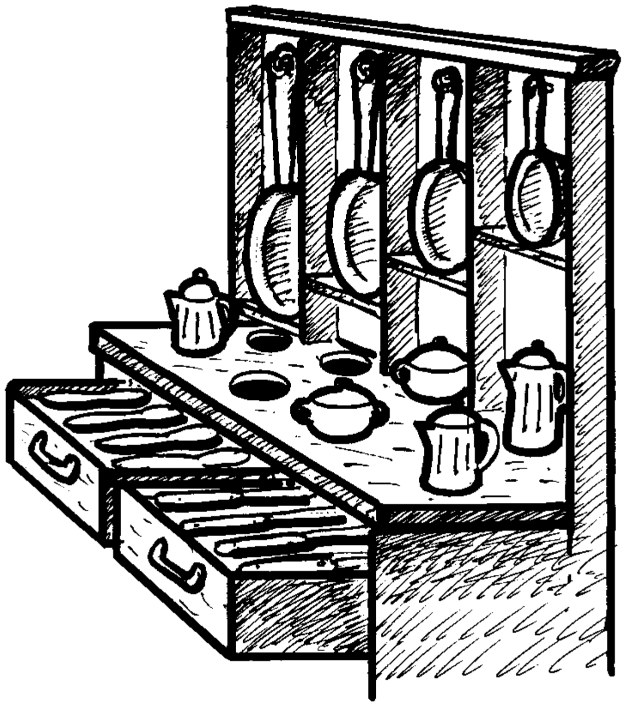
   But now reveal to the class that in Oompa-Loompa land, traffic lights work differently from the way they do for us: yellow means stop, red means go, and lights go green to warn people that they will soon have a stop light. How does this affect things? (Answer: the factory should follow the Oompa-Loompa's traffic-light convention—we should not try to impose our own.)

   The key concepts here are those of *transfer effects*—people transfer their learning and expectations of previous objects into new but similar situations—and *population stereotypes*—different populations learn certain behaviours and expect things to work in a certain way. Although the traffic light scenario may seem far-fetched (though nothing is all *that* farfetched in Oompa-Loompa land), there are many examples in our own world: in America light switches are on when they are up and off when they are down, whereas in Britain the reverse is true; calculator keypads and touchtone phones are laid out in different ways; and number formats (decimal point or comma) and date formats (day/month/year or month/day/year) vary around the world.

8. When one shift of Oompa-Loompas finishes work in the chocolate factory, they must clean up and put away pots and pans and jugs and spoons and stirrers ready for the next shift. There is a cupboard with

shelves for them to put articles on, but the next shift always has trouble finding where things have been put away. Oompa-Loompas are very bad at remembering things and have trouble with rules like "always put the pots on the middle shelf," "put the jugs to the left."

The groups of students should try to come up with a better solution.



The diagram on the right shows a good arrangement (which is sometimes used—but for rather different reasons—on yachts and other places where it is necessary to stop things sliding around). The key concept here is to use *visible constraints* to make it obvious where everything is supposed to go. It is clear from the size and shape of each hole which utensil it is intended for: the designer has made the constraints visible and used the physical properties of the objects to avoid the need to rely on arbitrary conventions.

9. In the main control room of the chocolate factory there are a lot of buttons and levers and switches that operate the individual machines. These need to be labeled, but because the Oompa-Loompas can't read, the labels have to be pictorial—iconic—rather than linguistic.

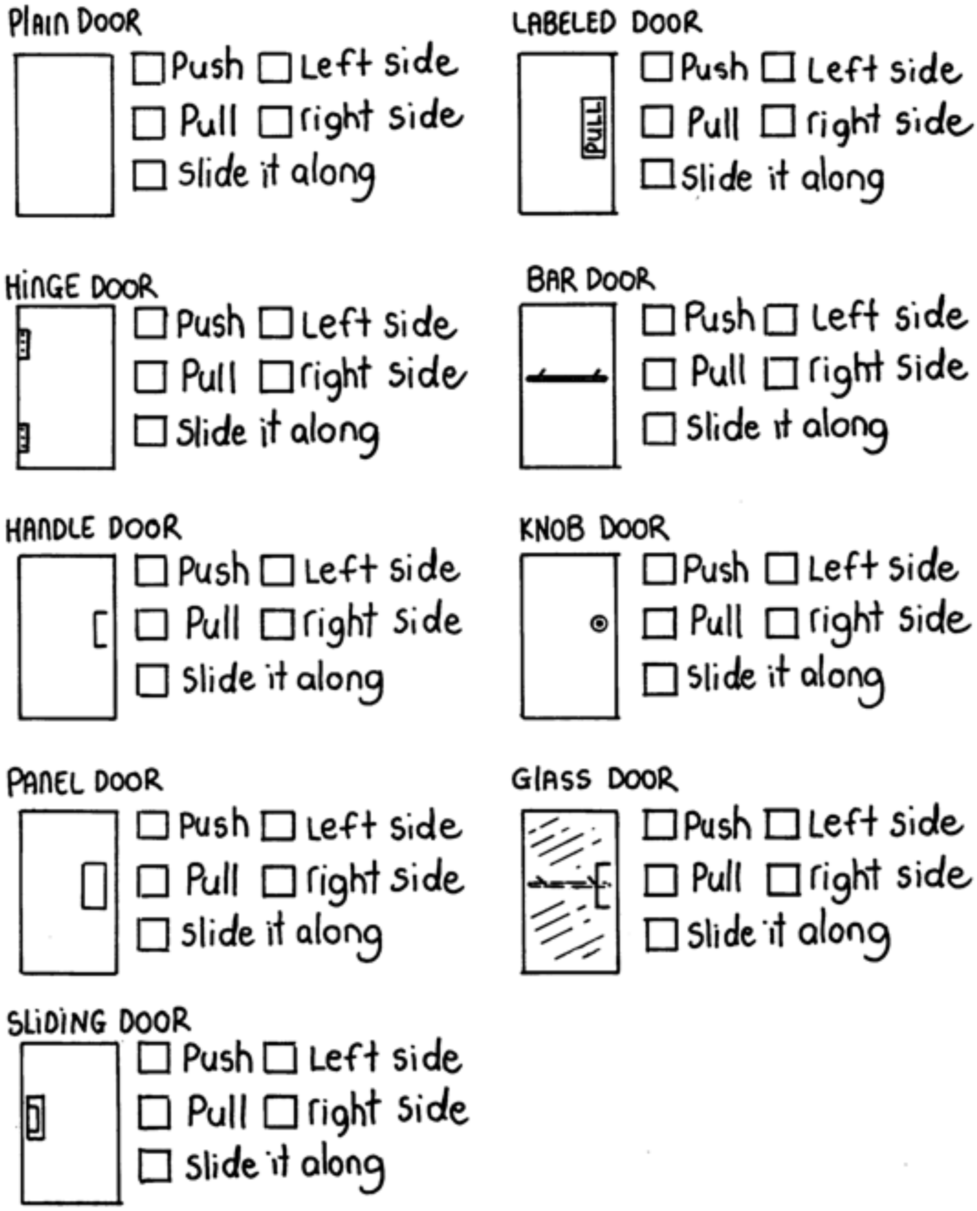
To give the students a feeling for icons, the worksheet *Icons* shows some examples. The students should identify what the icons might mean (for example, the letter going into a mailbox might represent sending a message). There are no "correct" answers to this exercise; the idea is simply to identify possible meanings.

10. Now let's design icons for the chocolate factory. The cards on worksheet *Icon cards* specify clusters of related functions, and each group of students receives one or more cards without the other groups knowing what they are. A control panel is to be designed for the function clusters that contains individual icons for each of the five or six operations. The groups then show their work to the other students, without saying what

the individual operations are, to see if they can guess what the icons mean. Encourage the use of imagination, color, and simple, clear icons.

Worksheet Activity: How do you open doors?

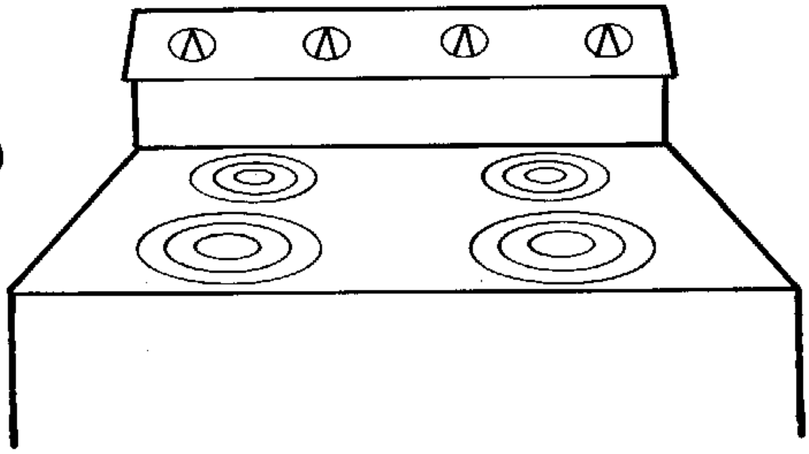
Fill out the worksheet to show how you think each type of door opens.



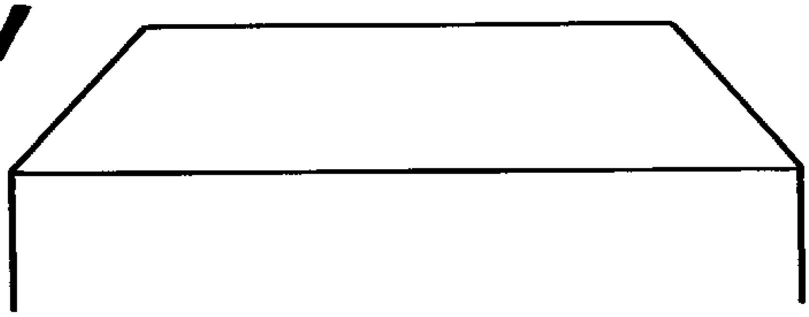
Worksheet Activity: Stove Top

Redesign the stove so that the controls are easy to use. Front or back panels can be added to the design if desired.

OLD



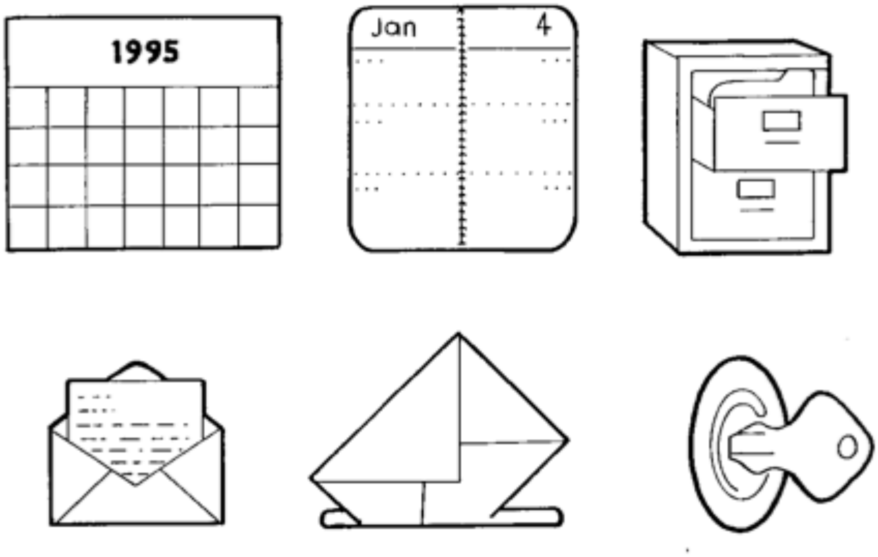
NEW



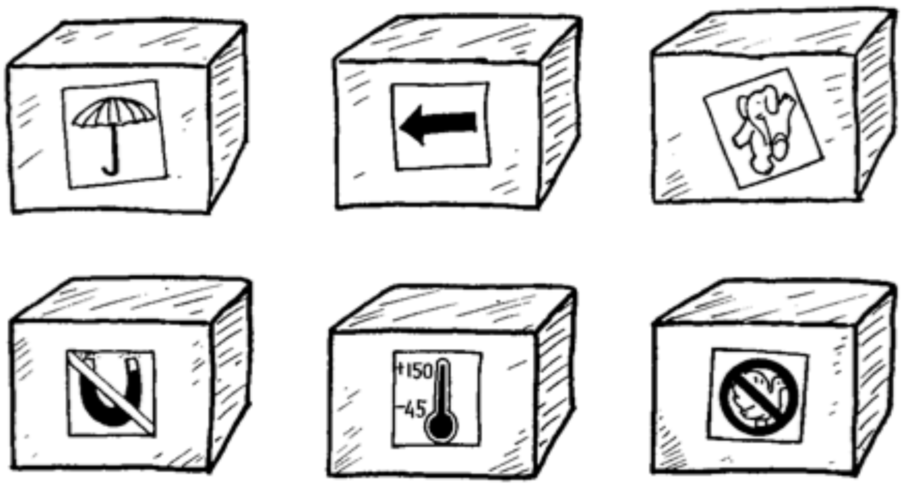
Worksheet Activity: Icons

What do you think each of the icons (symbols) means?

In an Office...



On a box...



Worksheet Activity: Icon Cards

Cut out the cards and give one to each group. Have each group design icons (symbols) to put on a control panel to represent each instruction.

Ingredients

add
- cocoa
- milk
- sugar
- extra sugar
- butter

Extras

add
- nuts
- caramel
- ginger
- raisins
- coconut

Making

- start mixing
- stop mixing
- start heating
- stop heating
- pour into moulds
- stamp a pattern (lots of different ones!)

Tasting

- taste it
- wonderful!-premium grade
- ok-regular grade
- yech-cooking chocolate
- yech,yech-throw out

Sizing

- small bar
- medium bar
- large bar
- humungous bar
- set bar size (in squares)
- make chocolate chips

Packing

- wrap with foil
- wrap with paper
- put into bag
- put into box
- start conveyer belt
- stop conveyer belt

Variations and extensions

Can the students set the time on a digital wristwatch or microwave oven? The mappings involved in the cooker layouts were simple because there were four controls for four burners. More difficulty occurs whenever the number of actions exceeds the number of controls. The controls on wristwatches or microwaves are often exceedingly complex, not because of the number of buttons (often there are only a few), but because of the number of states the device can get in to. ("You would need an engineering degree from MIT to work this," someone looking at his new wristwatch once told Don Norman, a leading user interface psychologist. Don *has* an engineering degree from MIT, and, given a few hours, he *can* figure out the watch. But why should it take hours?)

Students should keep an eye out for places where people get confused or frustrated using digital devices – mobile phones, video recorders, computers, remote controls – all these devices provide opportunities for frustrating users! Students should ask themselves, what is it about the device that confuses the users, and how might it have been designed better?

What’s it all about?

Human–computer interaction is about designing, evaluating, and implementing computer systems that allow people to carry out their activities productively and safely. In the old days, computers were for specialists and the users could be expected to be highly educated and specially trained in their use. Later people thought it was perfectly normal to buy a “dummies” book to find out how to use their computer. But now computers are everyday tools that we all must use, and far greater attention must be paid to the human interface.

Many disasters, some involving loss of life, have occurred because of inadequate interfaces: airplane crashes and even shoot-downs of civilian airplanes, freeway pile-ups because of errors in switching remotely-operated highway signs, nuclear power station disasters. On a smaller scale, most people experience frustration—often extreme frustration (a police officer once fired bullets into his computer screen)—with computers and other high-tech devices every day in the workplace. And it is not just computers: what about those shrink-wrapped packages that you could only open if you had sharp claws or a hooked beak, doors that hurt your wrist as you try to push your way through, milk cartons that always splash you when you open them, elevators where you can’t see how you’re supposed to push the button, home entertainment systems whose advertisements claim to do everything, but make it almost impossible to do anything?

We are becoming used to “human error” and to thinking of ourselves as somehow inadequate; people often blame themselves when things go wrong. But many so-called human errors are actually errors in design. People have limitations in how much information they can process, and designers need to account for these; bad design cannot be rectified by producing a detailed and complicated user manual and expecting people to study it intensively and remember it forever. Also, humans are fallible and design needs to take this into consideration.

Interface *evaluation* is an essential part of the design process. The present activity has involved some evaluation when the students tested their icon designs on others. A more thorough evaluation would test the design on real Oompa-Loompas (who may perceive icons differently) in a carefully-controlled psychology-style experiment.

Although the problems caused by technology form the butt of many jokes, human interface design is by no means a laughing matter. Inadequate interfaces cause problems ranging from individual job dissatisfaction to stock-market disasters, from loss of self-esteem to loss of life.

Further reading

Don Norman's book *The design of everyday things* is a delightful—and liberating—account of the myriad design problems in everyday products. Jeff Johnson's *Designing with the mind in mind* is a thought-provoking insight into how people think, and how interfaces should be designed to take account of the human element.

Activity 21

Conversations with computers—*The Turing test*

Summary

This activity aims to stimulate discussion on the question of whether computers can exhibit “intelligence,” or are ever likely to do so in the future. Based on a pioneering computer scientist’s view of how one might recognize artificial intelligence if it ever appeared, it conveys something of what is currently feasible and how easy it is to be misled by carefully-selected demonstrations of “intelligence.”

Curriculum Links

- ✓ Technology – Technological systems. Understand that technological systems are represented by symbolic language tools and understand the role played by the black box in technological systems.

Skills

- ✓ Interviewing.
- ✓ Reasoning.

Ages

- ✓ 7 years and up

Materials

- ✓ A copy of the questions in the *Turing Test Questions* sheet that each student can see (either one for each pair of students, or a copy displayed on a projector/overhead projector), and
- ✓ one copy of the answers in the *Turing Test Answers* sheet.

Conversations with Computers

Discussion

This activity takes the form of a game in which the students must try to distinguish between a human and a computer by asking questions and analyzing the answers. The game is played as follows.

There are four actors: we will call them Gina, George, Herb and Connie (the first letter of the names will help you remember their roles). The teacher coordinates proceedings. The rest of the class forms the audience. Gina and George are *go-betweens*, Herb and Connie will be answering questions. Herb will give a human's answers, while Connie is going to pretend to be a computer. The class's goal is to find out which of the two is pretending to be a computer and which is human. Gina and George are there to ensure fair play: they relay questions to Herb and Connie but don't let anyone else know which is which. Herb and Connie are in separate rooms from each other and from the audience.

What happens is this. Gina takes a question from the class to Herb, and George takes the same question to Connie (although the class doesn't know who is taking messages to whom). Gina and George return with the answers. The reason for having go-betweens is to ensure that the audience doesn't see how Herb and Connie answer the questions.

Before the class begins this activity, select people to play these roles and brief them on what they should do. Gina and George must take questions from the class to Herb and Connie respectively, and return their answers to the class. It is important that they don't identify who they are dealing with, for example, by saying "*She* said the answer is..." Herb must give his own short, accurate, and honest answers to the questions he is asked. Connie answers the questions by looking them up on a copy of the *Turing Test Answers* sheet. Where the instructions are given in italics, Connie will need to work out an answer.

Gina and George should have pencil and paper, because some of the answers will be hard to remember.

1. Before playing the game, get the students' opinions on whether computers are intelligent, or if the students think that they might be one day. Ask for ideas on how you would decide whether a computer was intelligent.

2. Introduce the students to the test for intelligence in which you try to tell the difference between a human and a computer by asking questions. The computer passes the test if the class can't tell the difference reliably. Explain that Gina and George will communicate their questions to two people, one of whom will give their own (human) answers, while the other will give answers that a computer might give. Their job is to work out who is giving the computer's answers.

3. Show them the list of possible questions in the *Turing Test Questions* sheet. This can either be copied and handed out, or placed on a projector.

   Have them choose which question they would like to ask first. Once a question has been chosen, get them to explain why they think it will be a good question to distinguish the computer from the human. This reasoning is the most important part of the exercise, because it will force the students to think about what an intelligent person could answer that a computer could not.

   Gina and George then relay the question, and return with an answer. The class should then discuss which answer is likely to be from a computer.

   Repeat this for a few questions, preferably until the class is sure that they have discovered who is the computer. If they discover who is the computer quickly, the game can be continued by having Gina and George toss a coin to determine if they will swap roles so the class no longer know which role the two have.

The answers that Connie is reading from are not unlike the ones that some "intelligent" computer programs can generate. Some of the answers are likely to give the computer away quickly. For example, no-one is likely to recite the square root of two to 20 decimal places, and most people (including, perhaps, the students in the class) would not be able to answer that question at all. Some questions will reveal the computer when their answers are combined. For example, the "Do you like..." answers sound plausible on their own, but when you encounter more than one it becomes apparent that a simple formula is being used to generate the answers from the questions. Some of the answers indicate that the question was misinterpreted, although the class might reason that the person could have made the mistake.

Many of the answers are very bland, but safe, and a follow-up question would probably reveal that the computer doesn't really understand the subject. Answering "I don't know" is reasonably safe for the computer, and might even make it seem more human—we might expect a student to answer "I don't know" to some of the questions too, such as the request for the square root of two. However, if a computer gives this answer too often, or for a very simple question, then again it would reveal its identity.

Since the goal of the computer is to make the questioners think that they are dealing with a person, some of the answers are deliberately misleading—such as the delayed and incorrect answers to the arithmetic problem. The questions and answers should provide plenty of fuel for discussion.

Worksheet Activity: Turing Test Questions

Choose questions from this list to ask the hidden human and “computer”.

1. What is the name of Bart Simpson’s baby sister?
2. What do you think of Roald Dahl?
3. Are you a computer?
4. What is the next number in the sequence 3, 6, 9, 12, 15?
5. What do you think of nuclear weapons?
6. What is 2 × 78?
7. What is the square root of two?
8. Add 34957 to 70764.
9. Do you like school?
10. Do you like dancing?
11. What day is it today?
12. What time is it?
13. How many days are there in February in a leap year?
14. How many days are there in a week?
15. For which country is the flag a red circle on a white background?
16. Do you like to read books?
17. What food do you like to eat?

Worksheet Activity: Turing Test Answers

Choose questions from this list to ask the hidden human and "computer".

1. What is the name of Bart Simpson's baby sister?

I can't remember.

2. What do you think of Roald Dahl?

He writes funny books.

3. Are you a computer?

Are *you* a computer?

4. What is the next number in the sequence 3, 6, 9, 12, 15?

18.

5. What do you think of nuclear weapons?

Nuclear weapons are very dangerous and should not be used.

6. What is 2 × 78?

166 *(This is deliberately incorrect!)*

7. What is the square root of two?

1.41421356237309504878

8. Add 34957 to 70764.

Wait for about 20 seconds before giving the answer

...105621.

9. Do you like school?

Yes, I like school.

10. Do you like dancing?

Yes, I like dancing.

11. What day is it today?

Give the correct day of the week.

12. What time is it?

Give the correct time.

13. How many days are there in February in a leap year?

2000 and 2004 are leap years. ***(This is deliberately incorrect!)***

14. How many days are there in a week?

Seven.

15. For which country is the flag a red circle on a white background?

I don't know.

16. Do you like to read books?

Yes, I like to read books.

17. What food do you like to eat?

I'm not hungry, thanks.

Variations and extensions

The game can be played with as few as three people if Gina also takes the role of George and Connie. Gina takes the question to Herb, notes his answer, and also notes the answer from the *Turing Test Answers* sheet. She returns the two answers, using the letters A and B to identify who each answer came from.

In order to consider whether a computer could emulate a human in the interrogation, consider with the class what knowledge would be needed to answer each of the questions on *Turing Test Answers*. The students could suggest other questions that they would have liked to ask, and should discuss the kind of answers they might expect. This will require some imagination, since it is impossible to predict how the conversation might go.

By way of illustration, here are two example conversations. The former illustrates "factual" questions that a computer might be able to answer correctly, while the latter shows just how wide-ranging the discussion might become, and demonstrates the kind of broad knowledge that a computer might need to call upon.

Question	Please write me a sonnet on the subject of the Forth Bridge.
Answer:	Count me out on this one. I never could write poetry.
Question:	Add 34957 to 70764.
Answer:	pause for about 30 seconds ... 105621.
Question:	Do you play chess?
Answer:	Yes.
Question:	My King is on the K1 square, and I have no other pieces. You have only your King on the K6 square and a Rook on the R1 square. Your move.
Answer:	after a pause of about 15 seconds ... Rook to R8, checkmate.

Question:	In the first line of the sonnet which reads "Shall I compare thee to a summer's day," would not "a spring day" do as well or better?
Answer:	It wouldn't scan.
Question:	How about "a winter's day"? That would scan all right.
Answer:	Yes, but nobody wants to be compared to a winter's day.
Question:	Would you say Mr. Pickwick reminded you of Christmas?
Answer:	In a way.
Question:	Yet Christmas is a winter's day, and I don't think Mr. Pickwick would mind the comparison.
Answer:	I don't think you're serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas.

There is a system called "Eliza" that is widely available on the web (it is a kind of "chatbot", which is a system that you can have typed conversations with). Eliza simulates a session with a psychotherapist, and can generate remarkably intelligent conversation using some simple rules. Some sample sessions with Eliza are discussed below. Students might try out Eliza, or other chatbots, although be warned that some have been trained using language and subjects that might not be appropriate for school students.

What's it all about?

For centuries philosophers have argued about whether a machine could simulate human intelligence, and, conversely, whether the human brain is no more than a machine running a glorified computer program. This issue has sharply divided people. Some find the idea preposterous, insane, or even blasphemous, while others believe that artificial intelligence is inevitable and that eventually we will develop machines that are just as intelligent as us. (As countless science fiction authors have pointed out, if machines do eventually surpass our own intelligence they will themselves be able to construct even cleverer machines.) Artificial Intelligence (AI) researchers have been criticized for using their lofty goals as a means for attracting research funding from governments who seek to build autonomous war machines, while the researchers themselves decry the protests as a Luddite backlash and point to the manifest benefits to society if only there was a bit more intelligence around. A more balanced view is that artificial intelligence is neither preposterous nor inevitable: while no present computer programs exhibit "intelligence" in any broad sense, the question of whether they are capable of doing so is an experimental one that has not yet been answered either way.

The AI debate hinges on a definition of intelligence. Many definitions have been proposed and debated. An interesting approach to establishing intelligence was proposed in the late 1940s by Alan Turing, an eminent British mathematician, wartime counterspy and long-distance runner, as a kind of "thought experiment." Turing's approach was operational—rather than define intelligence, he described a situation in which a computer could demonstrate it. His scenario was similar to the activity described above, the essence being to have an interrogator interacting with both a person and a computer through a teletypewriter link (the very latest in 1940s technology!) If the interrogator could not reliably distinguish one from the other, the computer would have passed Turing's test for intelligence. The use of a teletypewriter avoided the problem of the computer being given away by physical characteristics or tone of voice. One can imagine extending the exercise so that the machine had to imitate a person in looks, sound, touch, maybe even smell too—but these physical attributes seem hardly relevant to intelligence.

Turing's original test was a little different from ours. He proposed, as a preliminary exercise, a scenario where a man and a woman were being interrogated, and the questioner had to determine their genders. The man's goal was to convince the questioner that he was the woman, and the woman's was to convince the questioner that she was herself. Then Turing imagined—for this was only proposed as a thought experiment—a computer being substituted for one of the parties to see if it could be just as successful at this "imitation game" as a person. We altered the setup for this classroom activity, because the kind of questions that students might ask to determine gender would probably not be appropriate, and besides, the exercise promotes sexual stereotyping—not to mention deception.

Imitating intelligence is a difficult job. If the roles were reversed and a person was trying to pass themselves off as a computer, they would certainly not be able to do so: they would be given away by their slow (and likely inaccurate) response to questions like "What is 123456 × 789012?"

However, it is surprisingly easy for a computer to acquire a thin veneer of conversational ability. A famous program developed in the early 60s, Eliza (mentioned earlier), simulated a non-directive psychotherapist in a dialog with the user. On the right is an example of a conversation with this system. Something approaching a natural conversation is achieved by devices such as

Person:	Men are all alike.
Computer:	In what ways?
Person:	They're always bugging us about something or other.
Computer:	Can you think of a specific example?
Person:	Well, my boyfriend made me come here.
Computer:	Your boyfriend made you come here.
Person:	He says I'm depressed much of the time.
Computer:	I'm sorry to hear that you are depressed.
Person:	It's true. I'm unhappy.
Computer:	Do you think coming here will help you not to be unhappy?
Person:	I need some help, that much seems certain.
Computer:	What would it mean to you if you got some help?
Person:	Perhaps I could learn to get along with my mother.
Computer:	Tell me more about your family.
	...
Computer:	What else comes to mind when you think of your father?
Person:	Bullies.
Computer:	Does that have anything to do with the fact that your boyfriend made you come here?
	....

1. generating canned responses (e.g. *Can you think of a specific example* in response to something beginning *They're always …*; *I'm sorry to hear that …* in response to *He says …*);

2. echoing the user's statements (e.g. *Your boyfriend made you come here* in response to *My boyfriend made me come here)*;

3. recognizing key words (*Tell me more about your family* in response to *… mother …*);

4. some stock phrases (*What would it mean to you if …*);

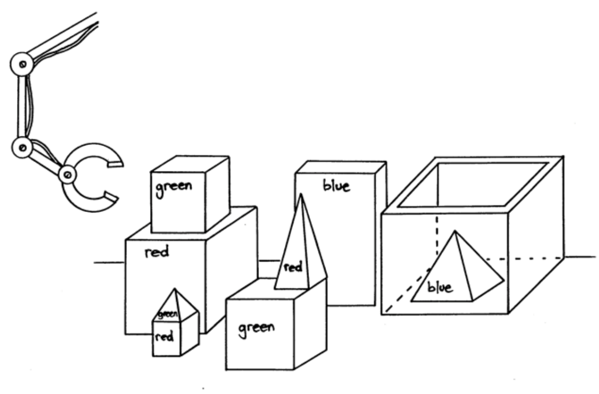
5. issues taken up from earlier parts of the conversation (*Does that have anything to do with the fact that …*).

Of course, this does not mean that the Turing test has been passed, because the person is not asking questions with the aim of identifying whether the respondent is a machine or not: if they were, it would not be long before all was revealed.

A widely-quoted incident, which may be apocryphal but has now passed into the folklore of artificial intelligence, occurred when a computer sales vice-president allegedly came across a terminal that was normally connected directly to a particular person's house, but on this occasion happened to be running the "psychotherapist" program. This conversation shows what happened. Although the vice-president was evidently taken in, it is clear that the situation predisposed him to believe that he was talking to a person. If he had suspected otherwise, he would soon have found out!

Another system that appears to be able to hold intelligent conversations is a program called "SHRDLU", developed in the late 60s that accepted instructions to manipulate students' blocks on a table top. In fact, the blocks and the table were simulated and presented in the form of a picture on a computer screen, although it would not have been too difficult to make things work with a real robot (at least in principle). The picture depicts the robot's micro-world.

SHRDLU could carry on surprisingly fluent conversations, as shown in the dialog below. Amongst other things, this illustrates how it can obey orders, disambiguate instructions, work out the referents of pronouns, and understand new terms ("steeple").



Person: Pick up a big red block.
Robot: OK.

The system answers "OK" when it carries out a command. In order to pick up the red block, it had to clear it off by finding a space for the green one and moving the green one away

Person: Grasp the pyramid.
Robot: I don't know which pyramid you mean.

There are three pyramids in the scene, so the program does nothing.

Person: Find a block which is taller than the one you are holding and put it into the box.

The system has to figure out what is meant by "one" and "it", as well as using deduction to decide which block it is holding.

Robot: By "it", I assume you mean the block which is taller than the one I am holding.

Printed out to keep us informed.

Robot: OK.

The hand has to set down the red block to pick up the taller blue one.

Person: What does the box contain?
Robot: The blue pyramid and the blue block.
Person: What is the pyramid supported by?

Earlier the system responded to "the pyramid" by saying "I don't know which pyramid you mean." Now, it knows that we must mean the one which was just mentioned.

Robot: The box.
Person: Does a steeple ...
Robot: Sorry, I don't know the word "steeple".

The system interrupts immediately when it sees a word it can't figure out.

Person: A "steeple" is a stack which contains two green cubes and a pyramid.

The quote marks indicate that we are defining a new word.

Robot: I understand.
Person: Are there any steeples now?
Robot: No.

The new definition is used as a description to recognize steeples.

Person: Build one.
Robot: OK.

The description given in the definition is enough to deduce a plan for building new objects as well as recognizing them. The word "one" means "a steeple", and the plan is carried out.

However, the system is very fragile and the illusion it creates is easily shattered.

Person:	Trade you a water pistol for the big red block.
Robot:	Sorry, I don't know the word “trade”.
Person:	A “trade” is a free exchange of property.
Robot:	Sorry, I don't know the word “free”.
Person:	A “free” act is done willingly, without being forced.
Robot:	Sorry, I don't know the word “act”.
Person:	“Acts” are what people do on purpose, and not by accident.
Robot:	Sorry, I don't know the word “people”.
Person:	Sorry, I thought you were smarter than you are.
Robot:	Sorry, I don't know the word “sorry”.

Here is a continuation of the conversation—which was not actually generated by the program, but certainly could have been. Although it was possible to define the new term *steeple* in the last conversation, only a very limited kind of term can be described because the robot has an extremely restricted “intelligence.”

Ultimately, it has to admit that it doesn’t even understand the words it uses itself (“Sorry, I don’t know the word ‘sorry’”)!

Person:	Trade you a water pistol for the big red block.
Robot?	I already have a water pistol, but I'll give you two blocks and a pyramid for your slimy old frog.

Now consider an alternate, completely imaginary, continuation of the conversation, which might have been generated by a student. Although tiny, this fragment demonstrates a wealth of world knowledge.

1. The “robot” appreciates that a water pistol is less valuable if you already have one.

2. Furthermore, it *expects the person to know that too* since it doesn’t bother to explain that this is a reason for declining the offer.

3. Still, it sees that the person wants a block badly enough to part with something important.

4. Also, it appreciates that—unlike water pistols—blocks are valuable in quantity.

5. It tries to soften the person up by demeaning the frog.

6. It implies that it is prepared to haggle.

Certainly this degree of artfulness is beyond today’s computers!

The story on the right was generated by another computer program (late 80s). Although it is immediately given away as computer-generated by its scarcity and terseness, it is not hard to imagine how it could be dressed up by adding all sorts of extra detail. What is interesting is not the superficial aspects of the story but the plot that it embodies. While this is a long way from any human-generated plot, it does seem to capture some human elements of conflict. These days there are a number of systems around for automatically generating stories, although the challenge in evaluating them is to determine how much of the material is just standard patterns with the gaps filled in, and how much is a plot that has been constructed creatively as above.

> Once upon a time there was an Arctic tern named Truman. Truman was homeless. Truman needed a nest. He flew to the shore. Truman looked for some twigs. Truman found no twigs. He flew to the tundra. He met a polar bear named Horace. Truman asked Horace where there were some twigs. Horace concealed the twigs. Horace told Truman there were some twigs on the iceberg. Truman flew to the iceberg. He looked for some twigs. He found no twigs. Horace looked for some meat. He found some meat. He ate Truman. Truman died.

There is an annual competition for the *Loebner prize*, in which computer programs compete to pass the Turing test by fooling judges into thinking that they are human. As of 2012, no computer has yet won the gold or silver prizes, which involve consistently fooling the judges, but a bronze prize is awarded each year for the one judged to be the most human. In the first year of the competition (1991) a program managed to win a bronze award by, amongst other tricks, making typing mistakes to appear to be more human!

No artificial intelligence system has been created that comes anywhere near passing the full Turing test. Even if one did, many philosophers have argued that the test does not really measure what most people mean by intelligence. What it tests is behavioral equivalence: it is designed to determine whether a particular computer program exhibits the symptoms of intellect, which may not be the same thing as genuinely possessing intelligence. Can you be humanly intelligent without being aware, knowing yourself, being conscious, being capable of feeling self-consciousness, experiencing love, being ... alive?

The AI debate is likely to be with us for many more decades.

Further reading

Artificial intelligence: the very idea by the philosopher John Haugeland is an eminently readable book about the artificial intelligence debate, and is the source of some of the illustrations in this activity (in particular, the SHRDLU conversations, and the discussion of them).

The original Turing test was described in an article called "Computing machinery and intelligence," by Alan Turing, published in the philosophical journal *Mind* in 1950, and reprinted in the book *Computers and thought*, edited by Feigenbaum and Feldman. The article included the first two conversations.

The psychotherapist program was described in "ELIZA—A computer program for the study of natural language communication between man and machine," by J. Weizenbaum, published in the computer magazine *Communications of the Association for Computing Machinery* in 1966.

The blocks-world robot program is described in a PhD thesis by Terry Winograd which was published as a book entitled *Understanding natural language* (Academic Press, New York, 1972).

The program that generated the story of Truman and Horace is described in "A planning mechanism for generating story text," by Tony Smith and Ian Witten, published in the *Proceedings of the 10th International Conference on Computing and the Humanities* in 1990.