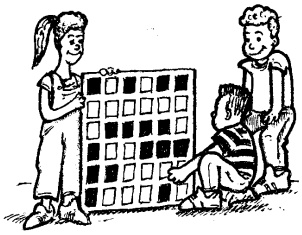
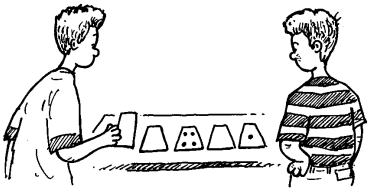
Computer Science Unplugged . . .
off-line activities and games
for all ages

Tim Bell

Ian H. Witten

Mike Fellows





`http://unplugged.canterbury.ac.nz`

Authors' contact details:

Tim Bell,
Department of Computer Science,
University of Canterbury,
Private Bag 4800,
Christchurch, New Zealand
`tim@cosc.canterbury.ac.nz`,
phone (+64 3) 364-2352

Ian H. Witten,
Department of Computer Science,
Waikato University,
Hamilton, New Zealand
`ihw@waikato.ac.nz`,
phone (+64 7) 838-4246

Mike Fellows,
Department of Computer Science,
University of Victoria,
Victoria, BC, Canada V8W 3P6
`mfellows@csr.UVic.ca`,
phone (604) 721-2399

About this book

Many important topics in computer science can be taught without using computers at all. This book unplugs computer science by providing twenty off-line activities, games and puzzles that are suitable for people of all ages and backgrounds, but especially for elementary school children. The activities cover a wide range of topics, from algorithms to artificial intelligence, from binary numbers to boolean circuits, compression to cryptography, data representation to deadlock. By avoiding the use of computers altogether, the activities appeal to those who lack ready access to computers, and are ideal for people who don't feel comfortable using them. The only materials needed are cards, string, crayons and other household items.

Full instructions are given for each activity, and reproducibles are provided wherever possible to minimize the effort required for class preparation. Each activity includes a background section that explains its significance, and answers are provided for all problems. All you need for most of these activities are curiosity and enthusiasm.

These activities are primarily aimed at the five to twelve year-old age group. They have been used in the classroom, in science center demonstrations, in the home, and even for community fun days in a park! But they are by no means restricted to this age range: they have been used to teach older children and adults too.

This book is principally for teachers who would like to give their classes something a bit different from the standard fare, teachers at the elementary, junior high, and high school levels. It is also written for computing professionals who would like to help out in their children's or grandchildren's classrooms, for parents who can use these as family activities, for homeschoolers, for science centers who run educational programs for children, for computer camps or clubs, and for course instructors—including university professors—who are looking for a motivational introduction to a computer science topic. It is designed for anyone who wants to introduce people to key concepts of the information age of which they have no knowledge themselves.

Topics include the Poor Cartographer (graph coloring), the Muddy City (minimal spanning trees), Treasure Hunt (finite-state machines), the Peruvian Coin Flip (cryptographic protocols), Magic Card Flips (error correcting codes), the Chocolate Factory (human–computer interaction), and many more.

So unplug your computer, and get ready to learn what computer science is really about!

Acknowledgments

Many children and teachers have helped us to refine our ideas. The children and teachers at South Park School (Victoria, BC), Shirley Primary School, and Ilam 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. The delightful illustrations were done by Malcolm Robinson and Gail Williams, and have also benefited from advice from Hans Knutson. Matt Powell has provided valuable assistance with the "Unplugged" project.

Special thanks go to Paul and Ruth Ellen Howard, who tested many of the activities and provided a number of helpful suggestions. Peter Henderson, Joan Mitchell, Nancy Walker-Mitchell, Jane McKenzie, 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: Judith, Pam, and Roberta 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 welcome comments and suggestions about the activities. Details about contacting the authors are given on page 227 in the conclusion.

[1] No relation to the first author.

[2] In fact, the text compression activity was invented by Michael.

Contents

Introduction

Computers are everywhere. Though you may not realize it, it's unlikely that you'll get through the day without using one. Even if you don't have one at home, and you don't use a bank, and you avoid the checkout at the supermarket—and the corner store—you'll probably end up using a computer disguised as a VCR, microwave, or video game. When you graduate from school, it will be hard to find a career that doesn't involve computers. They seem to be taking over! With computers around every corner, it makes sense to find out how they work, what they can do—and what they can't do. The activities in this book will give you a deeper understanding of what computers are about.

The reason this book is "unplugged" is that we are concerned about presenting the ideas and issues of computer science, and these are often easier to explain with paper and crayons, ordinary materials, and simple activities. We believe you will be surprised and delighted, as we are, with how much entertainment can be had with simple things animated by the ideas important in understanding computers. Rather than talking about chips and disks and ROM and RAM, we want to convey a feeling for the *real* building blocks of computer science: how to represent information in a computer, how to make computers do things with information, how to make them work efficiently and reliably, how to make them so that people can use them.

Pressing social issues are raised by computing technology, like how information can be kept private and whether computers will ever be as intelligent as us. There are performance issues: computers are mind-bogglingly fast, yet people are always complaining that their computer is too slow. And there are human issues: why do people get so frustrated using computers? Are computers getting out of control? Behind these issues lie important technical factors, and the activities in this book will help you understand what they are.

We have selected a wide range of topics from the field of computer science, and packaged them so that they can be learned without using a computer. They will give you a good idea of what computers can and can't do, what they might be able to do in the future, and some of the problems and opportunities that computer scientists face now.

But the main reason that we wrote this book is because computer science is *fun*. You don't believe us?—read on! The subject is bursting with fascinating ideas just waiting to be explored, and we want to share them with people who might not be tuned in to computers, but would be interested in the ideas in computer *science*. These activities will certainly give you something to think about.

The activities and how to use them

The activities are divided into six topics: data (*representing information*), putting computers to work (*algorithms*), telling computers what to do (*representing procedures*), really hard problems (*intractability*), sharing secrets and fighting crime (*cryptography*), and the human face of computing (*interacting with computers*). These areas are representative of the kinds of things that computer scientists study, although the list is far from exhaustive. Each topic has a brief introduction explaining where it fits into the wider picture, followed by several activities.

All the activities begin with a summary of materials needed. An age group is given, but it is merely indicative—the work can be adapted for anyone who has the basic skills, and a great deal depends on the children's background. There is no maximum age. Although nearly all the activities are designed for elementary school children, many have been used with older children, and even to introduce topics to university students. A time is indicated for each activity: this also is very approximate. The activities generally take about 30 to 40 minutes to complete, although they can be adapted to suit the time available. Some can serve as extended projects, and most can be cut down to a five-minute demonstration—perhaps as an illustration in a lecture or seminar. We indicate whether there is a minimum or maximum number of people required for an activity. Although the activities are described for a classroom situation, you will find that most can be undertaken individually or with the help of a second person such as a parent or friend.

Each activity description has the same structure. They begin with a *focus* section, which lists some of the key skills that are developed by the activity, and a *summary* section, which provides some background to the activity. There is a list of *technical terms* that might seem like jargon, but will be useful if you want to pursue the topic into the primary scientific literature. The *materials* part gives a checklist of what is required to undertake the activity with a class.

The next two sections take you through the activity. *What to do* is a step-by-step explanation of how to present it to children. The steps are the result of experimenting with several approaches, but you should feel free to adapt them to suit your group. *Variations and extensions* suggest alternative ways to present the activity, and ideas for extending it.

What's it all about? is intended to fill the teacher in on the wider significance of the activity. Older children might appreciate hearing about some of this—you might even have them read the section themselves. Younger children may not be able to comprehend the bigger picture, and it is often best to let them enjoy the activity in its own right, perhaps with a simplified account of its significance. This section is also intended for parents. Some will be curious, and others will want to talk about the activities with their child at home. It may also prove useful for justifying the activities to parents or supervisors who are not convinced of their value.

Further reading provides references to books and papers on the topic. Many of the references are fairly technical, but we have tried to include ones written for lay people where possible. A general reference that provides an accessible introduction to many of the topics in the activities is David Harel's book *Algorithmics: The Spirit of Computing* , which was published in a slightly less technical form as *The Science of Computing*. Complete bibliographic information about all material mentioned appears at the end of the book.

Many of the activities deserve a reward for good work. We have made the picture on page 6 into a stamp, which we dispense freely on worksheets and the backs of hands in return for

a good effort. Any rubber stamp shop will be able to make one for you from a copy of the picture. Another reason for using this particular stamp is that it reminds children that the work is about computers, which is important because there is not much mention of computers during the activities.

For the teacher

Don't be put off from using these activities just because you feel you don't know much about computers. Despite the technical nature of the topics, this book is intended specifically for teachers who have no background in computer science. We find that such people often enjoy the activities as much as the children. The activities are designed to provide opportunities for teachers and students to learn together about the principles of computer science. To assist you, every activity has a section that explains its background in non-technical terms. Answers to all the problems are provided so that you can confirm that you have things right.

Most of the activities can be used to supplement a mathematics program, but some (particularly Activity 19 about human interface design and Activity 20 about artificial intelligence) are also appropriate for social programs. The *focus* section identifies skills that the children will exercise, ranging from representing numbers in base two to coloring, from logical reasoning to interviewing. Topics and skills can be located using the index at the back of the book.

If you aren't experienced with working in a classroom ...

These activities can be used by computing professionals who would like to communicate computer science ideas to children or other general audiences. They are ideal for those occasions when you are asked to present something about your profession, but realize that the computers that you work with are probably not nearly as impressive as the video games that the children use every day.

If you are going to use these activities in a classroom situation, particularly with younger children, getting some help from a teacher about keeping order in the classroom can make the time more productive, and avoid frustration. There are some fairly safe techniques that will work in general (such as "I'll choose someone sitting quietly with their hand up"), although the best method varies from class to class. Some teachers have reward systems that you can use, or you can bring your own rewards—such as stickers, or the stamp on page 6. There may be a signal that the teacher uses to indicate that everyone must be quiet. This sort of information can be very valuable!

For somewhat exploratory and open-ended activities such as these, it is natural to expect a range of responses from individual students. One effective strategy is to employ those students who rapidly understand what's going on to explain things to others who are having more difficulty. Younger students may become quite animated—more so than the university students to whom you might be accustomed! The goal is to do something interesting and to have fun; a certain amount of chaos is not necessarily a bad thing.

Don't forget to check the list of materials required: it's easy to forget an important piece of equipment. Often the materials will be available from the school (such as balance scales,

crayons, and chalk). If you are not involved in the teaching profession, you may not realize that the probability of a photocopier breaking down is particularly high in the five minutes before a class for which you need a set of handouts.

Many of the activities involve solving problems. In most cases, the intention is *not* to explain how to solve the problem, but to allow the children to understand it and find their own method of solution. Knowing an algorithm that solves a problem has little intrinsic value, but the process of discovering an algorithm (even if it is not the best one) can be very instructive. In some cases there is no good algorithm; rather, the intention is for the children to learn about the inherent complexity of the problem. Sometimes there will be an opportunity to explain to the children how a particular problem applies to the real world, but often it will suffice for children to see the elegance of a solution, or appreciate that some problems don't have easy solutions.

Above all, there is no substitute for enthusiasm and being well prepared. Children appreciate both of these and respond accordingly. And if you are from outside the class you have the advantage that the children may be more attentive because you are offering a break from the daily routine.

For the technically-minded

As well as providing a taste of computer science for children, we have included some material for those who would like to delve a little deeper into the subject. Like this one, these sections are marked "for the technically-minded."

One of our goals is to communicate what computer science is really about. Computer *science* is a rich subject concerned with what computers can and cannot do, how to approach problems, and how to make computers more valuable to their users. Very little computer science is taught in elementary and high school. Most students will do some sort of "computing" work, but usually it is about how to *use* computers rather than how to design them for other people to use. A common misconception is that computer science is about programming. Programming is a fundamental tool, but it is not the end in itself. Computer science is about programming in the same way that astronomy is about telescopes—just as learning astronomy need not involve understanding how a telescope is built, learning about computer science does not need to involve programming. In fact, much of computer science would exist even if computers didn't (although it would probably have a different name!) None of the activities in this book require a computer, but the principles that they teach are widely used in modern computers.

Few youngsters—or even adults—are aware of what computers really can and can't do. For example, many real-life optimization problems, such as time-tabling teachers and classes, or finding the shortest route to make deliveries, take too long to solve optimally on computers—regardless of how fast the computer is. There are even problems, such as solving some equations (known as Diophantine equations), that we can prove will *never* be solved by a computer. There are lots of things that computers can't do.

Many things that computers *can* do are also not widely known. For example, many people use debit cards to pay for goods, where money is transferred directly from their bank account to the store's. However, in the process the bank finds out where the purchase is being made, and could build up a profile of the person's shopping habits. Despite this loss of privacy, debit

cards are widely accepted. Most people are unaware that cryptographic protocols exist which enable the transaction to be carried out reliably *without the bank being able to identify who the money is going to!* This seems incredible—literally unbelievable—to people who have never encountered public key cryptosystems and information hiding protocols. If more people know about such things, there may well be an outcry for systems that protect privacy better. (Activity 16 on information hiding protocols demonstrates a situation where it is possible to exchange information without losing any privacy). Understanding the technical issues involved goes a long way to making informed decisions on privacy issues, just as an understanding of biology goes a long way to making informed decisions on environmental issues.

We hope that this book will take some of the science fiction out of people's understanding of computers. The upcoming generation of computer users deserves a clear view of the technical issues that underpin the myriad of computerized systems that permeate our lives.

To find out more about the "Unplugged" project, visit the web site at

`http://unplugged.canterbury.ac.nz/`.

Instructions: *Have this picture made into a rubber stamp (about half this size) and use it as a reward for good work.*

Part I

Data: the raw material—*Representing information*

Data is the raw material that computers work on. People used to think of computers as giant electronic calculators. But nowadays it's better to think of a computer as a cross between an electronic filing cabinet, a library, and a TV. Calculators work with numbers. But computers work with data of any kind: baseball lore, sports facts, letters, mailing lists, accounts, payrolls, advertisements, magazines, books, encyclopedias, music, movie listings—even movies themselves. Calculators do arithmetic on numbers. Computers can do that too. But more importantly, they can manipulate all sorts of data, looking for high-scoring players, predicting the outcome of baseball games, helping to write letters, address envelopes, balance accounts, print checks, distribute advertisements, lay out magazine pages, find information in books and encyclopedias, play music, look through movie listings—they can even show movies. What is really amazing is that all of this wide range of facts, documents and images are stored on a machine that, at the lowest level, only works with two things: zero and one!

In this part of the book we look at how different kinds of information can be represented by a computer. Internally, all computers store data in an electronic form that is based on a very simple idea: that everything can be coded as a sequence of the digits zero and one. You, the user, do not normally see these things because the data is presented in a human-readable form—who wants to see a bunch of numbers instead of a movie? But to understand what computers do, you need to know what it is that they work with, and how numbers, letters, words, and pictures can be converted into zeros and ones.

For teachers

Representing information is absolutely fundamental to computing. Although the difference is hard to pin down precisely, data, which dictionaries define as "numerical information in a form suitable for processing by computer," is subtly different from information, or "knowledge derived from study, experience, or instruction." We refer to the stuff that is stored and manipulated by computers as *data*, and the real-world entities that are so represented as *information*. Thus data is the raw material of computing, while information is the raw material of computer applications.[3] Information is converted into data for the computer, and data is presented as information to the user.

The five activities in this section provide a broad introduction to information representation. The first is about binary numbers, disguised as a game that even very young children enjoy playing. The second, a kind of "paint by numbers" activity, shows how pictures can be represented as numbers, and embodies conventions used in fax machines for transmitting images over phone lines. The next two activities are about ways of representing information that are efficient and reliable. Data often consumes vast quantities of computer disk space—this is particularly true of multi-media material containing sound and images. Computer users will testify that they never seem to have quite enough disk space, and so computer scientists are concerned not just with how to store data, but how to store it efficiently. Activity 3 shows how ordinary text can be represented as numbers in an efficient way. The next activity is about maintaining the integrity of the data. The media

[3] We use *data* in the singular, because it usually seems like a large—often formidably large—entity in itself, rather than a collection of individual "datums."

on which data is stored (and transmitted) is susceptible to the occasional error, and it is important that the effect of errors is negligible. Disguised as a magic trick, we introduce a widely-used technique to detect and correct minor errors in computer data. The final activity in this section is about how information can be quantified. Because the idea of information is so central to computer science, a whole theory has been developed that enables us to quantify information and find limits on how efficiently it can be stored and transmitted.

Taken together, these activities will help children understand how different kinds of information can be stored on a computer, without taking up more space than necessary, and in a way that decreases the chance of loss of data due to defects in the storage medium. They will also learn about how to measure the information content of data. The activities can be performed in any order, though it is helpful if Activity 1 is done first. The first three are suitable for children in early elementary school, while for the last two the children need to be at the middle elementary level. However, these are lower bounds: even quite advanced children enjoy these activities and learn from them, as do adults.

For the technically-minded

Many computer professionals will be surprised at the content of these activities. Progressing from binary numbers, through picture representation, to text compression, error control, and foundations of information theory, constitutes a rather unusual introduction to computer science. Indeed, many students of the subject are unfamiliar with some of these ideas.

No-one would disagree that information representation is basic to computing. But the traditional fare of integers, floating-point numbers, and character strings conveys an impoverished and pedestrian view of information. Pictures and text are what users see, and in this sense they are the fundamental data types in computing today. Instead of working toward structured data—such as lists, trees, and object hierarchies—the way many computing technology and programming courses do, we pursue a user-oriented track. Efficient storage of information, and its integrity, are the major issues of concern. A feeling for how information is quantified underpins our understanding of what it is that computers work with. These topics provide a view of computing that young children can relate to, and they lend themselves naturally to simple but intriguing activities.

Activity 1

Count the dots—*Binary numbers*

Age group	Early elementary and up.
Abilities assumed	Counting up to 15 or 31, matching, sequencing.
Time	10 to 40 minutes.
Size of group	From individuals to the whole class.

Focus

Representing numbers in base two.

Patterns and relationships in powers of two.

Summary

All data in a modern digital computer is ultimately stored and transmitted as a series of zeros and ones. This activity demonstrates how numbers and text can be represented using just these two symbols.

Technical terms

Binary number representation; binary to decimal conversion; bits and bytes; character sets.

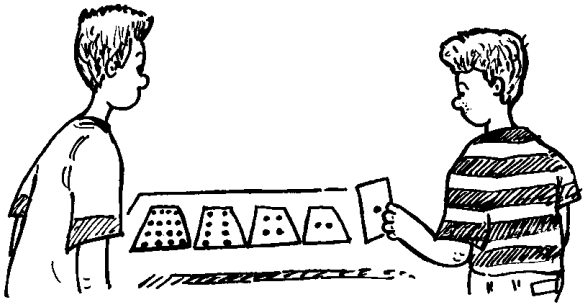


Figure 1.1: Initial layout of the binary cards

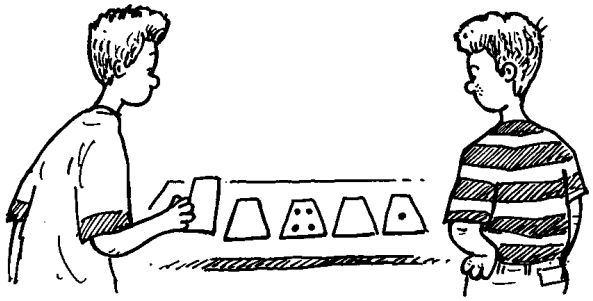


Figure 1.2: Flipping the cards to show five dots

Materials

Each child will need:

one set of five cards from the blackline master on page 17 (the blackline master has two sets),

a copy of the blackline master on page 18, and

a pen or pencil.

What to do

1. Seat the children where they can see you, and give each child a set of cards.
2. The children should lay their cards out, as in Figure 1.1, with the 16-dot card to their left.

   Some children will be tempted to put the cards in the opposite order, so you should check that they are in descending numeric order from left to right. For younger children, do not use the 16-dot card.

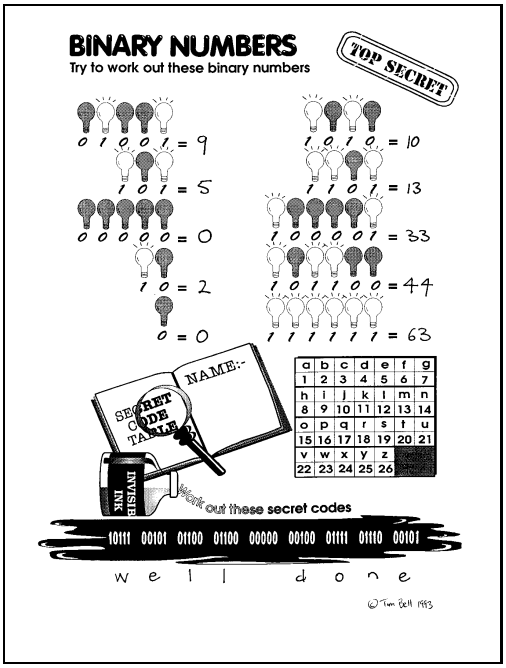


Figure 1.3: Solution to the worksheet on page 18

3. Have the children work out which cards to flip over so that exactly five dots are showing. The only (correct) way to do this is to have the 4-dot and 1-dot cards face up, and the rest face down (Figure 1.2). Each card must be either face up or face down, with all or none of its dots showing. Be prepared for some novel ways of getting five dots—it is not unusual for children to produce the requisite number by using spare cards to cover up three of the dots on the eight card!

4. Now get the children to show other numbers of dots, so that they explore which numbers can be represented.

   Ask for numbers such as three (requires cards 2 and 1), twelve (8 and 4), nineteen (16, 2 and 1) and so on. For those who find the combination for a number quickly, ask if they can find another way to get the number (there is only one way to display each number, and they are likely to discover this eventually).

   Discuss what the biggest number is that can be made with the cards (it is 31 for five cards, 15 for four cards). The smallest? (Often the number one will be offered first, but the correct answer is zero.) Is there any number between the smallest and largest that can't be represented? (No—all numbers can be represented, and each has a unique representation.)

5. For older children, ask them to display the numbers 1, 2, 3, 4, ... in sequence, and see if they can work out a procedure for incrementing the number of dots displayed on the cards by one (the number of dots increases by one if you flip all cards from right to left until you turn one face up).

6. This part of the activity uses zeros and ones to represent whether a card is face up or not.

   Tell the children that we will use a 0 to show that a card is hidden, and a 1 if its face is showing. For example, the pattern in Figure 1.2 is represented by 00101. Give them some other numbers to work out (e.g. 10101 represents 21, 11111 represents 31). With some practice the children will be able to convert in both directions. You could ask children to take turns calling out the day of the month that they were born on using zeros and ones, and have the rest of the class interpret the date.

   This representation is called the binary system, also known as base two.

7. Use the worksheet on page 18 to extend the exercise. (A completed worksheet is shown in Figure 1.3.)

   The worksheet uses a light bulb that is switched on to represent a card that is showing, and a light bulb that is off to represent a hidden card. The first few patterns should be easy to work out. For example, the first pattern has the 8 and 1 cards showing, so the value represented is $8 + 1 = 9$. For the patterns with fewer than five light bulbs, the children should use only the smaller valued cards. For example, the second pattern has only three light bulbs, which correspond (from left to right) to the 4-, 2-, and 1-dot cards respectively. See if the children can work this out for themselves.

   The six-bulb questions are designed to make the children think about how many dots should be on a sixth card. The number of dots on each card is double the number on the previous one, so the sequence is 1, 2, 4, 8, 16, 32, 64 .... Thus a 32-dot card would be added (at the left) to solve a problem that needs six cards.

   The code at the bottom of the worksheet uses the numbers 1 to 26 to represent the letters of the alphabet. (A zero can be used to represent a space.) The children must work out what each number in the code is, and look up the corresponding letter in the table. This shows how a textual message can be converted to a series of zeros and ones. The children can then write coded messages for each other.

Variations and extensions

Instead of using cards with dots, the exercise can be done with the lengths of rods (a set of rods of lengths 1, 2, 4, 8 and 16 units can be used to create any length from 0 to 31) or with weights (a set of weights of 1, 2, 4, 8 and 16 units can be combined to produce any weight from 0 to 31).

Instead of calling out sequences like 01101, try using beeps—call out a high-pitched beep for a one and a low-pitched one for a zero. This activity is noisy in the classroom, but children find it memorable! Modems and fax machines use tones like this to transmit information, although the tones are sent so quickly that they blend to make a continuous screeching sound. If the children aren't familiar with this, they could try calling a fax machine number to hear what it sounds like.

Any objects that have two states can be used to represent numbers. Figure 1.4 shows some different ways of representing the number nine (01001). A particularly challenging method is to use your fingers. If a finger is up it represents a one; down represents zero. Counting on your

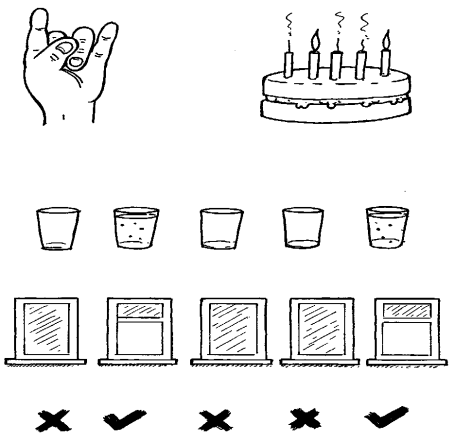


Figure 1.4: Some unusual ways of representing the number nine (01001 in binary)

fingers using the binary system enables you to go up to 31 on one hand, and 1023 on two hands. It requires some dexterity, and you have to watch out for rude gestures along the way! For a real challenge, try using your toes as well—this will allow you to count up to more than a million. (How many exactly? Two hands give 1024 possibilities, 0 through 1023. Hands and toes give $1024 \times 1024 = 1,048,576$ possibilities, 0 through 1,048,575.)

Older children will enjoy extending the sequence 1, 2, 4, 8, 16, 32 . . . The sequence contains an interesting relationship: if you add the numbers from the beginning from left to right, the sum will always be one less than the next number in the sequence.

Another property of binary numbers is that you can double the number by inserting a zero on the right-hand side of a number. For example, 1001 (9) doubled is 10010 (18). Older children should be able to explain why this happens. (All of the places containing a one are now worth twice their previous value, and so the total number doubles. The same effect occurs in base ten, where inserting a zero on the right of a number multiplies it by ten.)

Binary numbers are closely related to the guessing game in which one person thinks of a number and someone else tries to guess it by asking questions of the form "is it greater than or equal to x?" For example, suppose the number is known to be less than 32. A sensible first question would be "is it less than 16?" The yes/no answers to the questions are given by the zero/one bits in the binary representation of the number. This is explored in detail in Activity 5.

The five-bit code used for letters does not allow both upper- and lower-case letters to be represented. You could have the children work out how many different characters a computer has to represent (including digits, punctuation, and special symbols such as $), and consequently how many bits are needed to store a character. (With two lots of 26 letters, 10 digits, and a few punctuation marks, there are bound to be more than 64 codes needed, so at least seven bits are necessary. Seven bits allows for 128 characters, and this is more than sufficient.) Most current computers use a representation called ASCII (American Standard Code for Information Interchange), which is based on using seven bits per character. Longer codes that allow for the languages of non-English speaking countries are now becoming common.

What's it all about?

Modern digital computers almost exclusively use the system described above to represent information. The system is called binary because only two different digits are used. It is also known as base two (as opposed to base ten, which humans normally use). Each zero or one is called a "bit", the term being a contraction of *binary digit*. 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. On magnetic disks (floppy disks and hard disks), bits are represented by the direction of a magnetic field on a coated surface, either North-South or South-North. CD-ROMs store bits optically—the part of the surface corresponding to a bit either does or does not reflect light. When data must be transmitted over a telephone line or radio link, the ones and zeros are commonly represented by high- and low-pitched tones.

One bit on its own can't represent much, so they are usually grouped together as in the exercise above. It is very common to store bits in groups of eight, which can represent numbers from 0 to 255. A group of eight bits is called a byte.

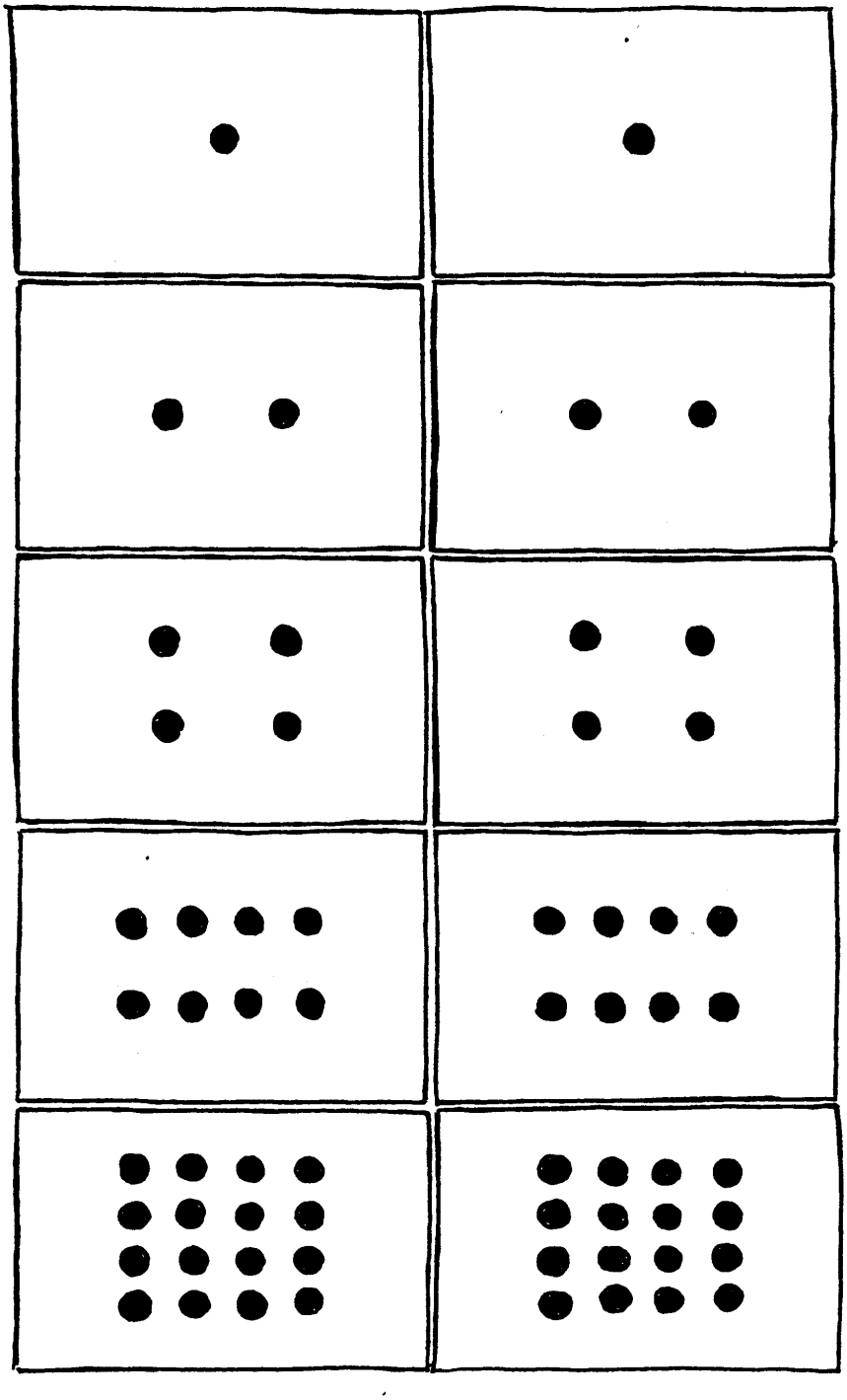
As well as representing numbers, this code can also represent the characters in a word-processor document. A byte is often used to represent a single character in a text—the numbers 0 to 255 are more than enough to encode all the upper- and lower-case letters, digits, punctuation, and many other symbols.

To represent larger numbers, several bytes are grouped together. Two bytes (16 bits) can represent 65,536 different values, and four bytes can represent over 4 billion values. The speed of a computer is affected by the number of bits it can process at once. For example, a 32-bit computer can perform arithmetic and manipulations on 32-bit numbers, whereas a 16-bit computer must break large numbers into 16-bit quantities, making it slower.

Normally we don't see the bits and bytes in a computer directly because they are automatically converted to characters and numbers when they are displayed, but ultimately bits and bytes are all that a computer uses to store numbers, text, and all other information.

Further reading

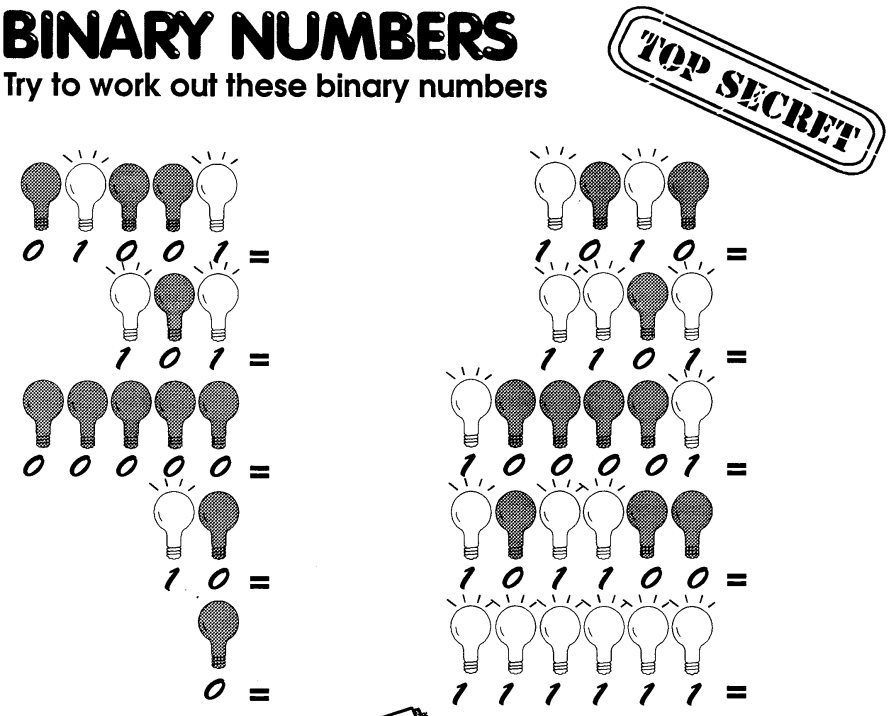
Most introductory computing texts discuss the binary number system. *My friend Arnold's book of Personal Computers* by Gareth Powell has a whole chapter on binary numbers.



Instructions: *Copy this page onto card, and cut out the boxes to make two sets of five cards.*

BINARY NUMBERS

Try to work out these binary numbers



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

Work out these secret codes

10111 00101 01100 01100 00000 00100 01111 01110 00101

Instructions: *Work out the numbers represented by the lightbulbs at the top of the page. Also, there is a message coded in binary at the bottom of the page; work out the numbers and look them up in the table to get the message.*

Activity 2

Color by numbers—*Image representation*

Age group	Early elementary and up.
Abilities assumed	Elementary counting, and some concentration. Activity 1 (Count the Dots) is helpful, but not essential, preparation. Experience with graphing is also useful.
Time	20 minutes or more.
Size of group	From individuals to the whole class.

Focus

Representation.

Coloring.

Pictures.

Summary

Computers are often used to store drawings, photographs and other pictures. This activity shows how pictures can be represented efficiently as numbers in a computer.

Technical terms

Raster images; pixels; image compression; run-length coding; facsimile machines.

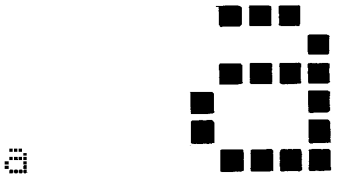


Figure 2.1: The letter “a” from a computer screen, and a magnified view showing the pixels that make up the image.

Materials

Each child will need:

a copy of the blackline master on page 25, and

a pencil and eraser.

You will need:

an overhead projector transparency of Figure 2.1 (or draw it on the classroom board).

What to do

1. Discuss what facsimile (fax) machines do (arranging for the children to send and/or receive faxes would be excellent preparation for this activity). Ask for other places where computers store pictures (for example, a drawing program, a game with graphics in it, or a multi-media system).

   Explain that computers can really only store numbers (if they have done the activity on binary numbers then they will already have some appreciation of this). Have the children suggest how a picture might be represented using only numbers.

2. Demonstrate how images are displayed on a computer screen, as follows:

   Computer screens are divided up into a grid of small dots called *pixels*. The word pixel is a contraction of “picture element.” On a black and white screen, each pixel is either black or white. Figure 2.1 shows a picture of a letter “a” that has been magnified so that the dots are visible. (This image should be shown on an overhead projector, or drawn on the board.) When a computer stores a picture, all that it needs to store is which dots are black and which are white.

3. Using the image of Figure 2.1, demonstrate how pictures can be represented by numbers on the blackboard, as follows:

   Begin by writing down the number of consecutive white pixels in the first line of the image (in Figure 2.1 there is only one white pixel at the start of the first line). Then write

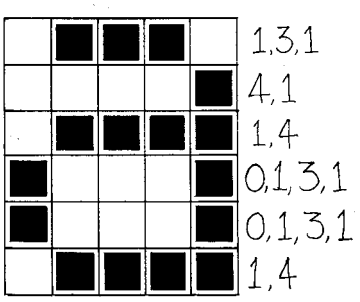


Figure 2.2: The image of Figure 2.1 coded using numbers

down the number of consecutive black pixels (three), and so on until the whole line has been coded. Thus the first line of Figure 2.1 will be represented as 1, 3, 1. Each row of pixels in the image is transcribed using this method. For example, the second line of Figure 2.1 is represented by 4, 1, since it has four white pixels followed by one black one. Figure 2.2 shows the final representation of Figure 2.1. Notice that lines beginning with a black pixel have a zero at the beginning to indicate that there are no white pixels at the start of the line.

There may be some confusion with this demonstration if you use a chalkboard, because coloring in a pixel makes it white rather than black!

4. Have the children "decode" the pictures on the worksheet on page 25.

   Hand out the worksheets and get the children to decipher the images there (see Figure 2.3 for the completed images; Figure 2.4 contains a reduced version which makes the pictures clearer). The "test" picture on the right is the easiest, and the "being" on the left is the most complex.[1] The squares on the grid should be colored in completely, and made as dark as possible. If the squares on the blackline master are too small, you can use quad-ruled paper. It is important to use pencil, as mistakes are easily made! Children should mark each line of numbers as they finish with it. Some may prefer to circle all the numbers that represent black pixels to help them keep track of where they are up to—it is very easy to get confused about which numbers are for black and which are for white.

   Note that each line always begins with the number of white pixels. If the first pixel is black, the line will begin with a zero.

Variations and extensions

The image will be clearer if the children draw on a sheet of tracing paper on top of the grid, so that the final image can be viewed without the grid.

Instead of coloring in the grid the class could use squares of sticky paper on a larger grid, or place any objects on a grid. The codes could even be used for a cross-stitch pattern.

[1] The images are taken from the excellent children's drawing program *KidPix*, by Brøderbund Software.

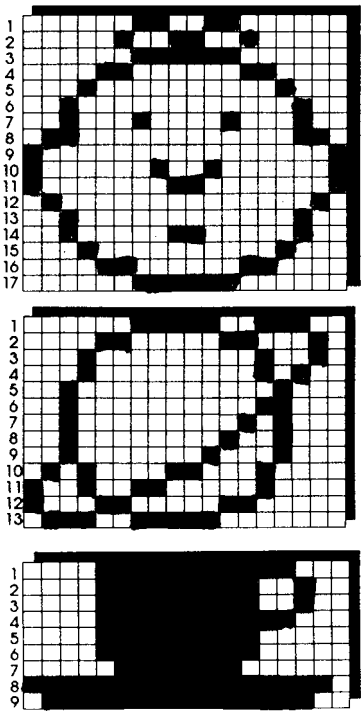


Figure 2.3: The completed images

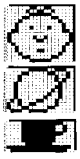


Figure 2.4: The completed images reduced

Children can try drawing their own pictures on a grid (or try to copy one from a computer screen). They can code the picture as numbers and give it to friends to decipher.

In practice there is usually a limit to the maximum length of a run of pixels because the length is being represented as a binary number. Ask the children how they would represent a run of twelve black pixels if they 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.)

The method can be extended to colored images by using a number to represent the color (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 color.

What's it all about?

The simplest way to represent a black and white picture on a computer is to represent white pixels with a zero (say) and black with a one. However, it is common for images to contain large blocks of white pixels (especially in the margins) and runs of black pixels (e.g. a horizontal line). Facsimile documents commonly have about 100 pixels per inch, so a row of white pixels at the top of a 7 inch wide page could take 700 bits of storage.

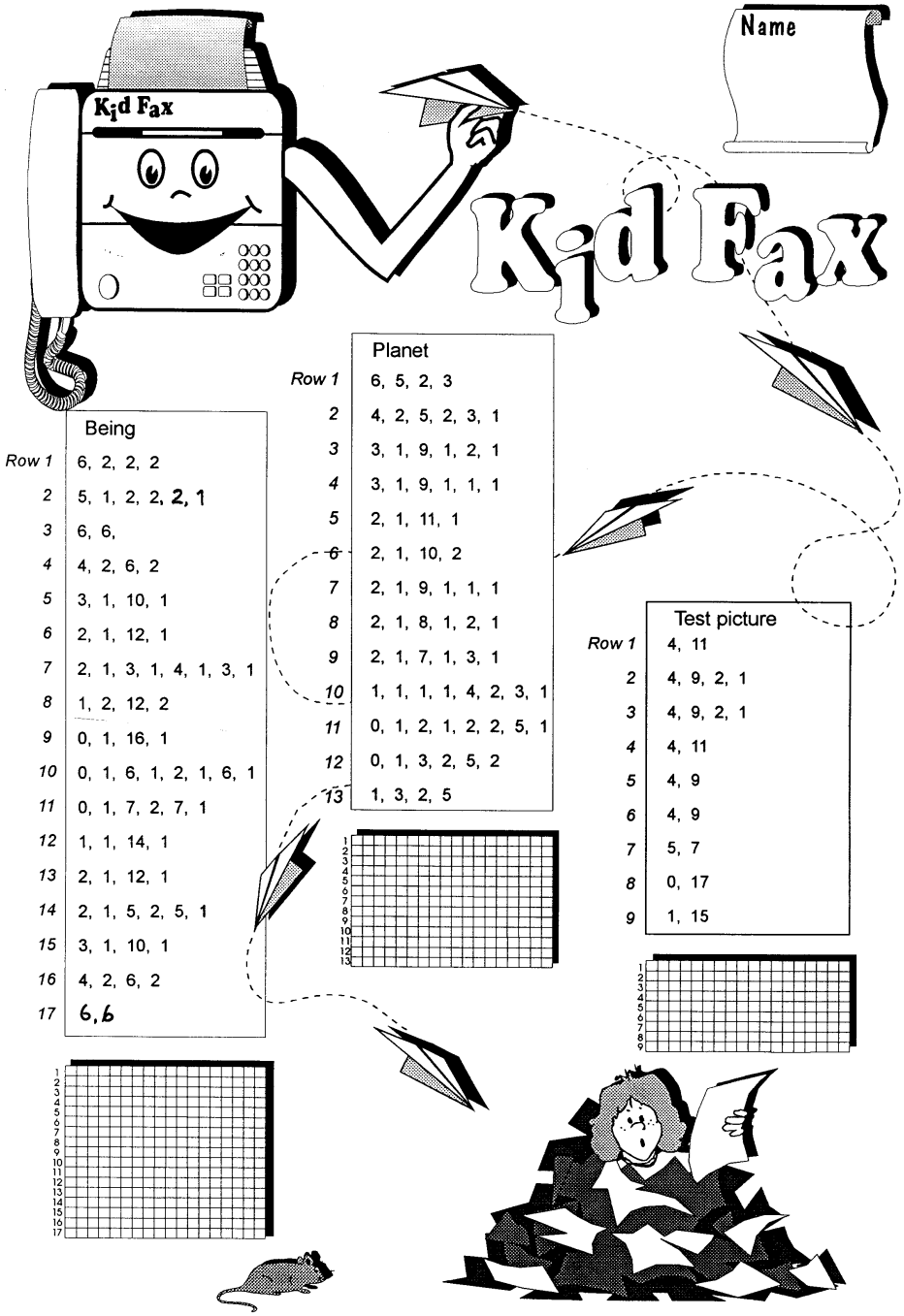
The "run-length coding" that we used in the activity above is much more efficient. It represents the run-length of 700 using the binary system (see Activity 1), which can be done in 10 bits. This example is a little extreme, but nevertheless significant savings can be made.

Saving space using techniques like this is called *compression*, and it is crucial to the viability of working with images on computers. 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, which would make fax a much less attractive means of communication. Images stored on computer disks are often compressed to a tenth or even a hundredth of their original size. Without compression there might be room for just a handful of images on the disk, but with compression many more images can be stored. The advantage is even greater when storing moving pictures, which normally involve 25 or more images each second. The compression method used in this activity is related to the method used on most common fax machines. A host of other methods are also available. The ones that give the best compression are "lossy"—they change the image very slightly so that it can be stored much more efficiently.

Is there a down-side?—can "compressing" an image ever *expand* it? Well, yes. A checkerboard image in which pixels alternate black and white would be much more efficient to store using one bit for each pixel than using one number for each pixel, because it will be necessary to represent the number as several bits to allow for the possibility of longer runs. Although this particular case is unlikely, some real images—such as halftone images, like illustrations in newspapers, that are made up of lots of tiny dots—do not compress well and may even expand. It is sometimes necessary to tailor the representation method to the kind of image being represented.

Further reading

Representing images on computers is discussed in depth by Netravali and Haskell in *Digital pictures: representation and compression*. The standard method for coding on fax machines is described by Hunter and Robinson in a paper published in 1978 entitled "International digital facsimile coding standards." A more recent standard for coding images is described in the book *JPEG: Still Image Data Compression Standard* by Pennebaker and Mitchell.



Instructions: *Use the numbers to color in the squares. There is a row of numbers for each line in the picture. For example, the line 4, 9, 2, 1 means to leave 4 squares empty, color in 9, leave 2 empty, and color in the next one.*

Activity 3

You can say that again!—*Text compression*

Age group	Early elementary and up.
Abilities assumed	Copying written text.
Time	10 minutes or more.
Size of group	From individuals to the whole class.

Focus

Writing.

Copying.

Repetition in written text.

Summary

Despite the massive capacity of modern computer storage devices, there is still a need for systems to store data as efficiently as possible. By coding data before it is stored, and decoding it when it is retrieved, the storage capacity of a computer can be increased at the expense of slightly slower access time. This activity shows how text can be coded to reduce the space needed to store it.

Technical terms

Text compression; Ziv-Lempel coding.

Materials

Each child will need:

a copy of the blackline master on page 32.

You will also need:

a copy of other poems or text for the children to encode.

What to do

1. Give each child a copy of the blackline master on page 32, and have them "decode" the message by filling in empty boxes with the contents of the box that their arrow points to. For example, the first empty box should contain the letter *e*. The first empty box on the second line refers to the phrase *Pease porridge* on the first line. The completed worksheet is shown in Figure 3.1.

2. Now have the children write their own representation of some text using boxes and arrows. The goal is to have as few of the original characters left as possible. The arrows should always point to an earlier part of the text to ensure that it can be decoded if the boxes are filled in from top to bottom, left to right. The children can either choose their own text, make some up, or encode some that has been provided. Many nursery rhymes and poems for children can be coded particularly effectively because they tend to have a lot of repetition, at least in the rhyming parts[1]. Books such as Dr. Seuss's "Green eggs and ham" also provide a good source of compressible text.

Variations and extensions

The arrow-and-box code requires that arrows always point to an earlier piece of text, although it is only the first letter pointed to that needs to be earlier. For example, Figure 3.2 shows a coded version of the word "Banana." Even though the missing text points to part of itself, it can be decoded correctly provided the letters are copied from left to right—each letter becomes available for copying before it is needed. This kind of self-referential representation is useful if there is a long run of a particular character or pattern.

On computers the boxes and arrows are represented by numbers. For example, the code in Figure 3.2 might be represented as "Ban(2,3)", where the 2 means count back two characters

[1] A good rhyme is "A dillar, a dollar / a ten o'clock scholar / what makes you come so soon / you used to come at ten o'clock / and now you come at noon."

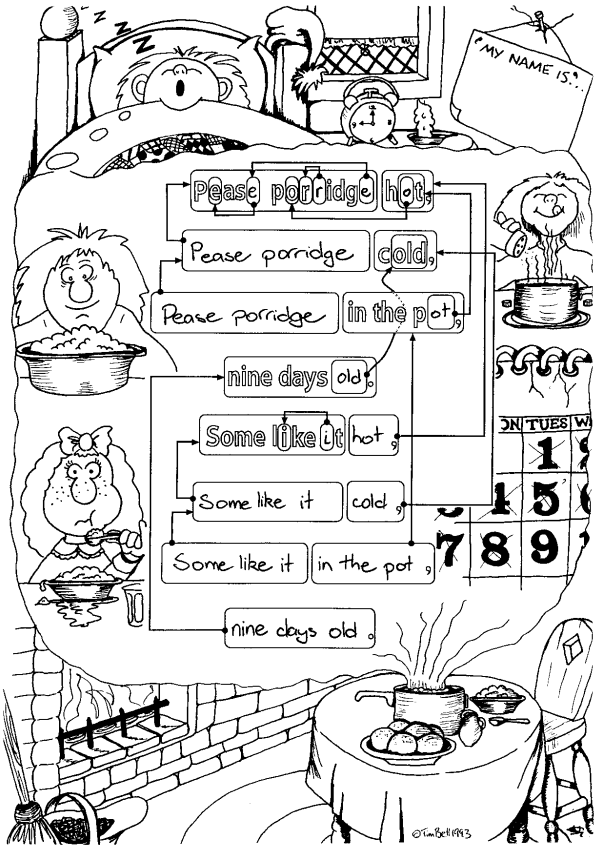


Figure 3.1: The completed worksheet

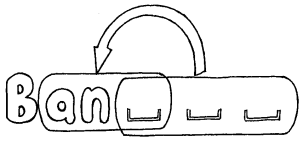


Figure 3.2: A self-referential code

to find the starting point for copying, and the 3 means copy three consecutive characters. On a computer the pair of numbers typically take up a similar amount of space to two letters, so matches of only one character are not normally encoded. Children could try encoding and decoding texts using the number representation.

To get a feel for how this technique would work on real text, children could be given a photocopied page containing a quantity of text, and starting about half-way down the page, cross out any text that could be replaced by a pointer to an earlier occurrence. The earlier occurrences should only be groups of two or more letters, since there is no saving if a single letter is substituted with two numbers. The goal is to get as many letters crossed out as possible.

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 instead of satisfying demand, the growth is fueling it. The ability of computers to store whole books suggests the possibility of storing libraries of books on computer; being able to show a high quality image on a computer suggests displaying movies; and the availability of high capacity CD-ROM simplifies the distribution of data and programs that had previously been limited by the size of a floppy disk. Anyone who owns a computer will have experienced the variation of Parkinson's law that states that the data always expands to fill the storage available for it.

An alternative to buying more storage space is to *compress* the data on hand. This activity illustrates the compression process: the representation of the data is changed so that it takes up less space. The process of compressing and decompressing the data is normally done automatically by the computer, and the user need not even be aware that it is being done except that they might notice that the disk holds more, and that it takes a little more time to get files from the disk. Essentially some computing time is being traded for storage space. Sometimes the system can even be faster using compression because there is less to read from the disk, and this more than compensates for the small amount of time needed to decompress the material.

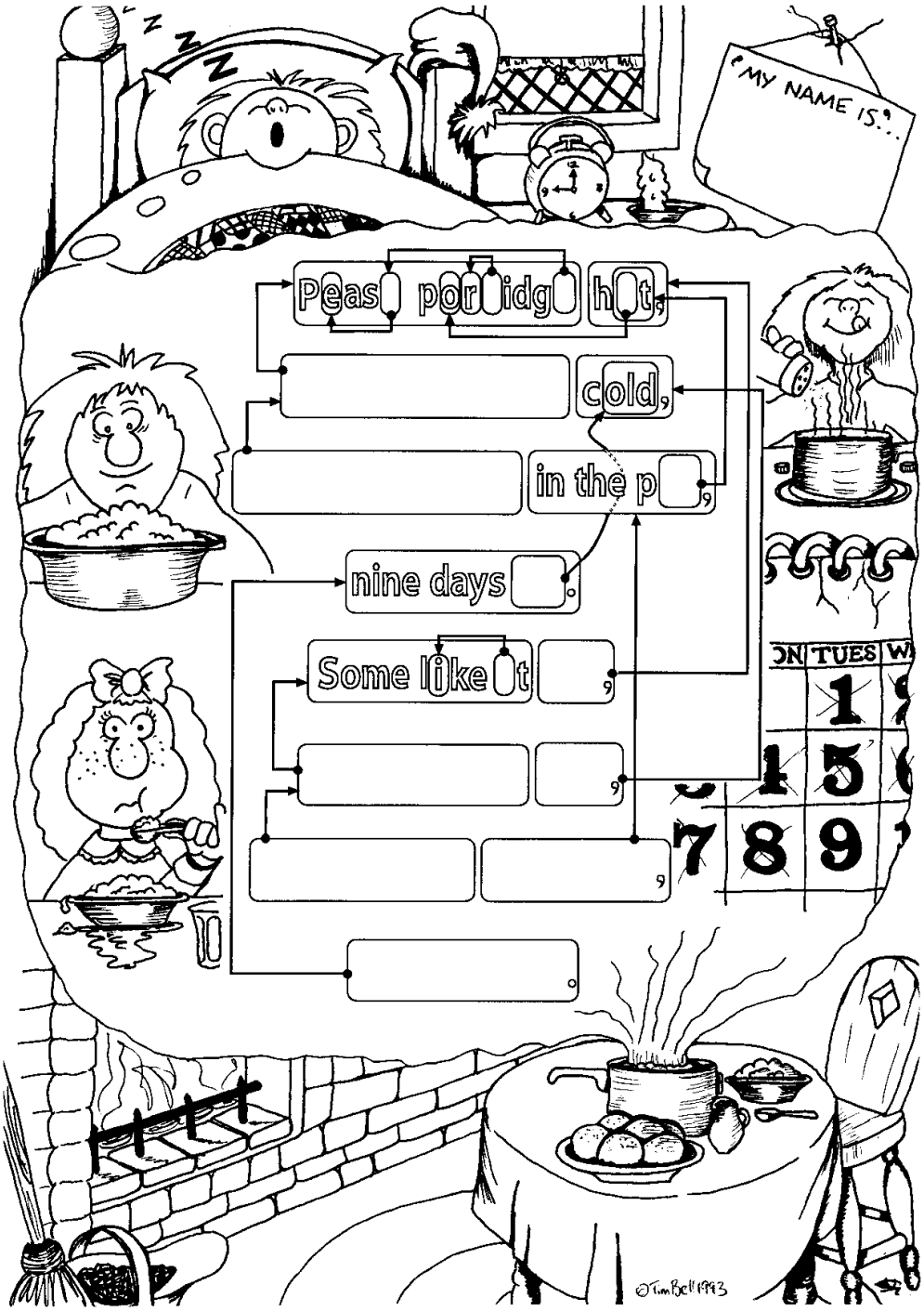
Many methods of compression have been invented. The principle of pointing to earlier occurrences of chunks of text is a popular technique, often referred to as "Ziv-Lempel coding," or LZ coding, after two Israeli professors who published several important papers in the 1970s and 1980s about this kind of compression. It is very effective because it adapts to whatever sort of data is being coded—it is just as suitable for Spanish, or even Japanese which uses a completely different alphabet, as it is for English. It even adapts to the subject of the text, since any word that is used more than once can be coded with a pointer. LZ coding will typically halve the size of the data being compressed. It is used in popular archiving programs with names like *Zip* and *ARC*, and in "disk doubling" systems. It is also used in high-speed modems, which are devices that allow you to interact with computers over ordinary telephone lines. Here it reduces the amount of data that needs to be transmitted over the phone line, noticeably increasing the apparent speed of transmission.

Another class of compression methods is based on the idea that frequently occuring letters should have shorter codes than rare ones (Morse code uses this idea.) Some of the best methods (which tend to be slower) use the idea that you can have a good guess at what the next letter will

be if you know the last few letters. We will encounter this principle in Activity 5.

Further reading

A comprehensive introduction to compression methods can be found in the books *Text compression* by Bell, Cleary and Witten, and *Managing Gigabytes: Compressing and indexing documents and images* by Witten, Moffat and Bell—although these are aimed at university-level computer science students rather than lay people. If you are interested in computer programming, you will enjoy Mark Nelson's *The data compression book*, which is a practically-oriented guide to the subject. Dewdney's *Turing Omnibus* discusses a technique called "Huffman coding" in a section about text compression.



Instructions: *Fill in each blank box by following the arrow attached to the box and copying the letters in the box that it points to.*

Activity 4

Card flip magic—*Error detection and correction*

Age group	Middle elementary and up.
Abilities assumed	Requires counting and recognition of (small) odd and even numbers. Children will get more out of it if they have learned binary number representation (see Activity 1, Count the Dots).
Time	About 30 minutes.
Size of group	From individuals to the whole class.

Focus

Odd and even numbers.

Patterns.

Magic tricks.

Summary

When data is transmitted from one computer to another, we usually assume that it gets through correctly. 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.

Technical terms

Error detecting codes, error correcting codes, parity.

Materials

Each pair of children will need:

a pile of approximately 25 identical cards, as described below.

To demonstrate to larger groups you will need:

a set of about 40 cards with magnets on them, and a metal board for the demonstration.

What to do

This activity is presented in the form of teaching the children a "magic" trick. Their interest is easily gained by first performing the trick, and then offering to show them how to do it. As with any magic trick, a certain amount of drama is helpful in making the presentation effective. The children will need to be sitting where they can see you.

The trick requires a pile of identical, two-sided cards. For example, the cards could be red on one side and white on the other. An easy way to make them is to cut up a large sheet of cardboard that is colored on one side only. A pack of playing cards is also suitable.

For the demonstration it is easiest if you have a set of cards that have magnets on them. It is possible to buy strips of magnetic material with a peel-off sticky back, and these are ideal. Make about forty cards, half with a magnet on the front and half with one on the back. Alternatively, fridge magnets can be used; glue them back to back in pairs, and paint one side. You can then lay the cards out vertically on a metal board (e.g. a whiteboard), which is easy for the class to see. When the children come to do the trick they can lay the cards out on the floor in front of them.

1. Have one or two children lay out the cards for you on the magnetic board, in a rectangular shape. Any size rectangle is suitable, but about five by five cards is good. (The larger the layout, the more impressive the trick.) The children can decide randomly which way up to place each card. Figure 4.1 shows an example of a five by five random layout.

2. Casually add another row and column to the layout "just to make it a bit harder" (Figure 4.2). Of course, these cards are the key to the trick. The strategy is to choose the extra cards to ensure that there is an even number of colored cards in each row and column (this is explained in more detail below).

3. Select a child, and while you cover your eyes and look away, have the child swap a card—just one card—for one of the opposite color (they only need to flip it over if it is on the floor). For example, in Figure 4.3, the third card in the fourth row has been flipped. You

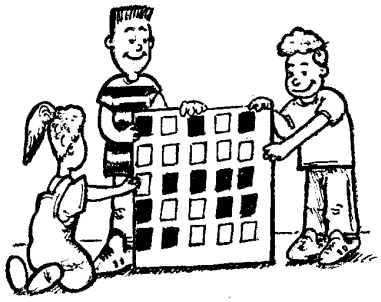


Figure 4.1: An initial random five by five layout of cards

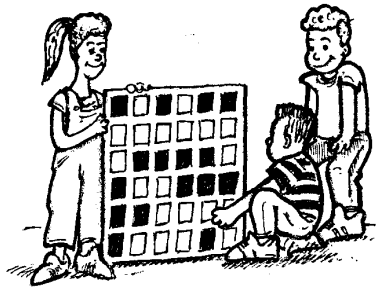


Figure 4.2: The cards with an extra row and column added

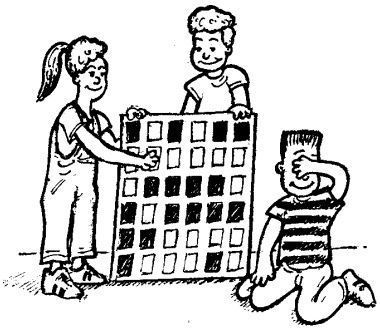


Figure 4.3: One card has been flipped

then uncover your eyes, study the cards, and identify which one was flipped. Because of the way the cards were set up, the row and column containing the changed card will now have an odd number of colored cards, which quickly identifies the offending card. Flip the card back, and have a couple more children choose a card to flip. Children will likely be quite impressed with your ability to repeatedly find the flipped card. The trick works with any number of cards, and is particularly impressive if a lot of cards are used, although it is important to rehearse the trick in this case.

4. Have the children try to guess how the trick is done. This is a good exercise in reasoning, and also helps to establish that the method is not obvious.

5. At this point you offer to teach the trick to the children. We have yet to see this offer refused!

   It is helpful to have the children work in pairs. Get each pair to lay out their own cards in a four by four square, choosing randomly whether each card is showing colored or white.

   Check that children can remember the concept of odd and even numbers, and that they appreciate that zero is an even number. Then get them to add a fifth card to each row and column, making sure that the number of colored cards is even (if it is already even then the extra card should be white, otherwise it should be colored). The technical name for the extra card is a *parity* card, and it can be helpful to teach the children the term at this point.

   Point out what happens if a card is flipped—the row and column of the flipped card will have an odd number of colored cards, and so the flipped card is the one where the offending row and column intersect. Each member of a pair can now take turns at doing the "trick."

   Once they are comfortable with the trick they may wish to get together with another pair and make up a larger rectangle of cards. You might even try pooling all the children's cards together to make one huge square.

Variations and extensions

Instead of cards, you can use just about any object at hand that has two "states." For example, you could use coins (heads or tails), sticks (pointing left-right or up-down), or cups (upside-down or right-way-up). If the activity is to be related to binary representation (Activity 1), the cards could have a zero on one side and a one on the other. This makes it easier to explain the wider significance of the exercise—that the cards represent a message in binary, to which parity bits are added to protect the message from errors.

There are lots of other things that children can discover by experimenting with the cards. Get them to think about what happens if two cards are flipped. In this case it is not possible to determine exactly which two cards were flipped, although it is always possible to tell that something has been changed. Depending on where the cards are in relation to each other, it can be possible to narrow it down to one of two pairs of cards. A similar thing happens with three card flips: you can always tell that the pattern has been changed, but it's not possible to determine exactly which cards have been flipped. With four flips it is possible that all the parity bits will be correct afterwards, and so the error could go undetected.

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, which is fortunate as it saves having to remember whether it applies to the row or the column. Children can probably "prove" this to themselves by trying to find a counter-example.)

The description above uses *even parity*—it requires an even number of colored cards. It is also possible to use *odd parity*, where each row and column has an odd number of colored cards. However, the lower right-hand card only works out the same for its row and column if the number of rows and columns of the layout are either both odd or both even. For example, a 5 by 9 layout or a 12 by 4 layout will work out fine, but a 3 by 4 layout won't. Children could be asked to experiment with odd parity and see if they can discover what is happening to the corner bit.

An everyday use of a related kind of error checking occurs in the International Standard Book Number (ISBN) given to published books. This is a ten-digit code, usually found on the back cover of a book, that uniquely identifies it. The final (tenth) digit is not part of the book identification, but is a check digit, like the parity bits in the exercise. It can be used to determine if a mistake has been made in the number. For example, if you order a book using its ISBN and get one of the digits wrong, this can be determined using only the checksum, so that you don't end up waiting for the wrong book.

The checksum is calculated using some straightforward arithmetic. You 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 to form a value which we will call s. For example, the ISBN 0-13-911991-4 gives a value

$$s = (0\times 10)+(1\times 9)+(3\times 8)+(9\times 7)+(1\times 6)+(1\times 5)+(9\times 4)+(9\times 3)+(1\times 2) = 172.$$

You then take the remainder after dividing s by 11, which is 7 for the example. If the remainder is zero then the checksum is zero, otherwise subtract the remainder from 11 to get the checksum. For the example, the checksum (at the end of the ISBN), is therefore $11-7=4$.

Figure 4.4: A bar code (UPC) from a grocery item

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

With this formula for a checksum it is possible to come up with the value 10, which requires more than one digit. This is solved in an ISBN by using the character X for a checksum of 10. It isn't too hard to find a book with an X for the checksum—one in every 11 should have it.

Have the children try to check some real ISBN checksums. Now get them to see if they can detect whether one of the following common errors has been made in a number:

- 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.

Have the children think about what sort of errors could occur without being detected. For example, if one digit increases and another decreases then the sum might still be the same.

A similar kind of check digit (using a different formula) is used on bar codes (universal product codes or UPCs) such as those found on grocery items (Figure 4.4). If a bar code is misread, the final digit is likely to be different from its calculated value, in which case the scanner beeps and the checkout operator must re-scan the code.

What's it all about?

Imagine that 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 to be added to your balance. 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 it is in the bank's interest to make sure that the message gets through correctly!

This is just one example of where it is important to detect errors in transmitted data. Just about anywhere that data is transmitted between computers, it is crucial to make sure that a receiving computer can check that the incoming data has not been corrupted by some sort of electrical interference on the line. The transmission might be the details of a financial transaction, a fax of a document, some electronic mail, or a file of information. And it's not just transmitted data that we are concerned about: another situation where it is important to ensure

that data has not been corrupted is when it is read back from a disk or tape. Data stored on this sort of medium can be changed by exposure to magnetic or electrical radiation, by heat, or by physical damage. In this situation, not only do we want to know if an error has occurred, but if possible we want to be able to reconstruct the original data—unlike the transmission situation, we can't ask for corrupted disk data to be re-sent! Another situation where retransmission is not feasible is when data is received from a deep space probe. It can take many minutes, even hours, for data to come in from a remote probe, and it would be very tedious to wait for retransmission if an error occurred. Often it is not even possible to retransmit the data, since space probes generally don't have enough memory to store images for long periods.

People have come to expect that when they store a document on their word processor, or send a message by electronic mail, they won't find the occasional character changed. Sometimes major errors happen, as when a whole disk gets wiped out, but very minor errors just don't seem to occur. The reason is that computer storage and transmission systems use techniques that ensure that data can be retrieved accurately.

Being able to recognize when the data has been corrupted is called *error detection*. Being able to reconstruct the original data is called *error correction*. A simple way to achieve error detection and correction is to transmit the same data three times. If an error occurs then one copy will be different from the other two, and so we know to discard it . . . or do we? What if, by chance, two of the copies happen to be corrupted in the same way? What's more, the system is very inefficient because we have to store or send three times as much data as before. All error control systems require some sort of additional data to be added to our original data, but we can do better than the crude system just described.

All computer data is stored and transmitted as sequences of zeros and ones, called *bits*, so the crux of the problem is to make sure that we can deliver these sequences of bits reliably. The "card flip" game uses the two sides of a card to represent a zero or one bit respectively, and adds parity bits to detect errors. The same technique is used on computers. Adding a single parity bit to a row of bits will allow us to detect whether a single bit has been changed. By putting the bits into imaginary rows and columns (as in the game), and adding parity bits to each row and column to ensure that it has an even number of ones, 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.

In practice computers often use more complex error control systems that are able to detect and correct multiple errors. Nevertheless, they are closely related to the parity scheme. Even with the best error control systems there will always be a tiny chance that an error goes undetected, but it can be made so remote that it is less likely than the chance of a monkey hitting random keys on a typewriter producing the complete works of Shakespeare.

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.

Further reading

The parity method of error detection is relatively crude, and there are many other methods around that have better properties. Some of the more important ideas in error coding are *Hamming distances, CRC (cyclic redundancy check), BCH (Bose-Chaudhuri-Hocquenghem) codes, Reed-Solomon Codes,* and convolutional codes. Details about these sorts of codes can be found in Richard Hamming's book *Coding and Information Theory*, and Benjamin Arazi's *A commonsense approach to the theory of error correcting codes.* The ISBN of Hamming's book is 0-13-139139-9, an interesting number for a book about coding. Less comprehensive, but more accessible, information about error correction can be found in *The Turing Omnibus* by Dewdney, and in *Computer Science: An Overview* by Brookshear.

Activity 5

Twenty guesses—*Information theory*

Age group	Can be simplified to suit middle elementary and up.
Abilities assumed	Comparing numbers and working with ranges of numbers.
Time	20 to 30 minutes.
Size of group	From two people to the whole class.

Focus

Deduction.

Ranges of numbers.

Asking questions.

Summary

Computer science is very much concerned with information. Computer systems store information, retrieve it, analyze it, summarize it, and exchange it with other computers. Since information is so fundamental to the subject, it is important to know how to quantify it. Information theory provides a way of measuring information, and includes laws that define limits on how efficiently it can be stored or transmitted. This activity studies an important method of measuring information content.

Technical terms

Information theory; Shannon theory; coding; data compression; error detection and correction; entropy.

Materials

You will need:

a writing surface, such as blackboard and chalk, and

cards with answers to questions written on them (shown in Table 5.1), as described below.

What to do

1. Have a discussion with the children about *information*. Ask for their definition of information. Ask them how we might measure how much information there is in, say, a book. They might suggest the number of pages, or the number of words. But what if it is a particularly boring book, or particularly interesting—does one have more information than the other? What if the book contained nothing but the phrase "Blah blah blah" repeated over and over? Would 400 pages of that contain more information than a 400 page telephone directory?

   It soon becomes apparent that although we have an intuitive notion of information content, it is hard to quantify. Explain to the group 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 today in a helicopter," that *would* be surprising, and would therefore convey a lot of information.

   How can the surprise value of a message be measured? 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 traveled by spaceship.

   The aim of this following activity is to get a feel for how much information messages contain, based on how difficult they are to guess. It is really just a sophisticated version of the game of twenty questions, although the children are allowed to ask more than twenty questions if necessary.

2. Select a child to stand in front of the group and answer questions about some information that is on a card which you give them. The group is initially told what kind of information it is—whether it is a number, a sequence of numbers, or a sentence; and if it is a number, what range it lies in. The group then asks questions of the child who has the card, but the child can only give yes/no answers. A good starter is to have the group guess a number

Type of message	Sample message	Notes
A number between 1 and 100	67	
A number between 1 and 1000	387	For older children, use numbers between 1 and 1,000,000.
Any whole number	145	For older children, use larger numbers.
The age of the child answering the questions	10	Or use your own age if you aren't easily offended!
A sequence of numbers	{2,4,6,8,10,12}	Explain that there are six numbers, and that they follow a pattern. Encourage the children to guess them one at a time, from first to last.
	{1,3,2,4,3,5,4,6,5,7}	These numbers go up two, down one, up two, down one, etc.
A sentence	The switch is on.	The sentence should be guessed one letter at a time, from left to right.

Table 5.1: Suggested messages for *Twenty Questions* (see text for notes about each)

between 1 and 100. They will typically ask questions like "Is it more than 50?" or "Is it between 20 and 60?" In the latter case, you should clarify whether "between" is inclusive or exclusive *before* an answer is given. Check that the child is answering the questions correctly, because one erroneous answer can be very misleading.

Record the questions and answers on the chalkboard (at least in shorthand) so that the group can refer back to them. It can be convenient to have the child with the card choose who can ask the next question.

Once the number has been guessed, count up the number of questions that were asked, and point out that this is a measure of the amount of "information" that the number was worth.

3. Now try the same game with some of the "messages" suggested in Table 5.1, taking note of how many questions are needed for each one.

   It may become boring for the children if you do *all* the message types in Table 5.1, depending on the age group, but they will probably enjoy this game in small doses.

   After they have been guessing numbers within a range, discuss the strategies that they used. The best strategy (which they will probably latch on to) is to halve the range of

possibilities with each question. You are always able to identify a number between 1 and 100 in just seven guesses this way, although children will be tempted to make wild guesses that do not usually pay off.

When the range is increased to 1,000, the natural reaction is to think that it will take ten times the effort, but in fact only three more questions are needed. Numbers between 1 and 1,000,000 can be found in just twenty questions. Every time the range doubles, just one extra question is needed—the number of questions increases in proportion to the logarithm of the size of the range.

For larger numbers, the children may devise other strategies—such as guessing a digit at a time. This is quite reasonable, for it is often a good idea to break a large "message" down into simple parts—although the halving strategy is always the most efficient in terms of the number of guesses taken.

When the "message" can be *any* whole number, the children will need to find an upper bound first. A typical sequence of questions begins "Is it less than 100?", "less than 1000?" etc. Discuss the strategy with them. To determine where they were headed, find out what questions they would have asked had the answers to these initial questions all been "no." Any increasing sequence of numbers will eventually find a bound. For example, one could simply say "Is it greater than or equal to 1?", then "2?", "3?" and so on, but this would take a long time for large numbers. A better strategy is to multiply the bound by 10 (or some other number) on each question, giving the questions "Is it less than 10?", "100?", "1,000?", and so on.

When the message is someone's age, it will probably only take a couple of guesses to find the answer. Discuss with the class why this is the case, even though an age can be any number between 1 and 100 or more. (The reason is that some ages are more likely than others. This relates to the "information content" discussed earlier—it would be very surprising if a grade 2 child was a hundred years old, or if the teacher was twelve years old.)

A similar effect occurs with the sequence of numbers. Once the beginning of the sequence is known (e.g. $\{2,4,6\}$), the next one is very predictable. Once they figure out the relationship, there is very little information in the latter part of the sequence.

Guesses for the letters of the sentence should be of the form "Is the next letter an *A*?", "Is the next letter a vowel?", or "Is the next letter one of *B*, *D*, or *P*?". Note that spaces between words should also count as "letters." Discuss with the children which letters were the hardest to guess. (Usually the first letter of a word is the hardest. Many of the letters are very predictable, and will be guessed first time.) The letters that are guessed quickly contain very little information; in fact, many of them could be left out and the message would still be intelligible—consider the sentence: "Ths sntnc hs th vwls mssng."

Variations and extensions

If the strategy for asking questions is pre-determined, it is possible to transmit a message without having to ask the questions.

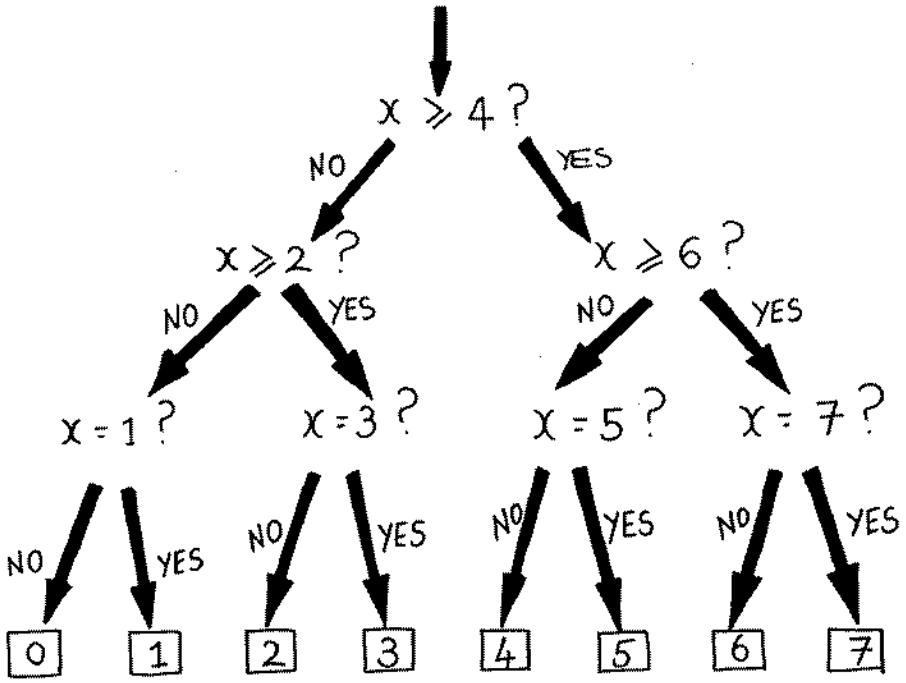


Figure 5.1: A strategy for guessing a number between 0 and 7

For example, Figure 5.1 shows a strategy for guessing a number between 0 and 7. The first question is "Is it greater than or equal to four?" If the answer is yes, we follow the right-hand path, and ask if it is greater than or equal to six. If that answer is no, we follow the left-hand path from there, and ask if the number is five. If that answer is no, the number must be four; otherwise it is five.

If the person answering the questions knew that this strategy was being followed, they need only give the sequence of answers and we would be able to figure out the message. For example, the answers "No, yes, yes" would represent the number three.

The structure shown in Figure 5.1 is called a *decision tree.* Children could design and use a decision tree for guessing numbers between 0 and 15. Have them think about the kind of tree they would use for someone's age (it might be biased towards smaller numbers). What about letters in a sentence? (In this case the decision might depend upon what the previous letter was.)

What's it all about?

We encountered the idea of a binary digit or "bit" of information in Activity 1, and 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?" Indeed, 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 tree of Figure 5.1, which is the same if we write no for 0 and yes for 1. The binary notation is discussed in detail in Activity 1.

The celebrated American mathematician (and juggler, and unicyclist) Claude Shannon defined the concept of "entropy" in 1948 as, roughly, the information content of a message. The amount of information in a message is a slightly slippery concept because it depends on your knowledge of what the message might be. 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 seven times out of eight, 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 that 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 just described, the answer to this will be "yes" 49/64, or about 77%, of the time, and on average you will be asking less than one question per coin toss! So the entropy depends not just on the *number* of possible outcomes—in the case of a coin toss, two—but also on their *probabilities*. Improbable events convey more information, just like helicoptering versus walking to school, but probable ones happen more frequently (by definition!), keeping the average information low.

Shannon was fascinated by the question of how much information is contained in messages written in a natural language like English. He performed an experiment in which people had to guess sentences letter by letter. He showed subjects text up to a certain point, and asked what they thought the next letter was. If they were wrong, they were told so and asked to guess again, and again, and again, until eventually they guessed correctly. For example, one of Shannon's texts began "there is no reverse on a motorcycle a friend of mine found this out rather dramatically the other day . . . " (he omitted punctuation and capitalization, to make life easier). A typical result of the experiment is:

t	h	e	r	e	•	i	s	•	n	o	•	r	e	v	e	r	s	e	•	o	n
1	1	1	5	1	1	2	1	1	2	1	1	15	1	17	1	1	1	2	1	3	2

•	a	•	m	o	t	o	r	c	y	c	l	e	•	a	•	f	r	i	e	n	d
1	2	2	7	1	1	1	1	4	1	1	1	1	1	3	1	8	6	1	3	1	1

where bullets represent spaces and the numbers below the letters record how many guesses it took the subject to get that letter correct. On the basis of no information about the sentence, this subject guessed that its first letter would be *t*—and in fact was correct. Knowing this, the next letter was guessed correctly as *h* and the following one as *e*. The fourth letter was not guessed first time. Seeing *the*, the subject probably guessed space; then, when told that was wrong, tried letters such as *n*, *i* and *s* before getting the *r*, which is correct, on the fifth attempt. Out of 44 symbols the first guess was correct 28 times, the second six times, the third three times, the fourth, fifth, sixth, seventh and eighth once each, while on only two occasions were more than eight guesses necessary.

Results like this are typical of prediction by a good subject with ordinary literary prose. In fact, we ourselves have tried this very sentence on many adult audiences and found that, in an astonishingly high proportion of cases, the number of guesses taken are *exactly* the same as those given above. From experiments such as these, Shannon showed that the entropy of

ordinary English text is between 0.6 and 1.3 bits per letter; more recent experiments indicate that the upper bound is more typical of people's performance. One bit per letter would mean that on average there are two possibilities for each letter, while two bits means four possibilities. With 1.3 bits, on average there are two and a half possibilities for each letter.

In principle, the decision tree idea of Figure 5.1 is a way of removing the interaction from the experiment by getting subjects to consider all possibilities in advance. You can imagine eliciting from a subject a decision tree for each possible context. The one for no context would say "guess *t*, then, if incorrect, guess *a*, then *o*, then *s*, . . . ," because *t* is the most likely letter to begin a sentence, then *a*, and so on. The one to use when the context was *the* says something like "guess space, then, if incorrect, guess *n*, *i*, *s*, *r*, . . . ". The one to use when the context was *q* says "guess *u*, then, if incorrect, guess space (for phrases like *coq a vin*), *a* (for words like *Qantas*), . . . " In practice, it would be impossibly tedious to elicit decision trees like this, even for a few contexts, let alone a complete set of them (and what is a "complete set"?—language is infinite, for it has limitless possibilities).

The concept of entropy relates directly to both compression (Activity 2 and Activity 3) and error detection/correction (Activity 4). You cannot compress a message to occupy less space than its entropy, at least on average. And although people's models of English text seem to indicate an inherent entropy of around 1.3 bits per letter, in practice computer models, which effectively embody decision trees as described above, are far less sophisticated: the best ones compress English text to around 2 bits per letter. Since letters are usually stored in 8-bit bytes, compression can reduce text files to around a quarter of their original size—which is certainly a worthwhile saving.

Compression *removes* redundancy from a message, whereas error detection and correction *insert* it. For example, adding the extra row and column of cards in Figure 4.2 increases the number of cards from 25 to 36. It is this extra information that allows changes in the cards—more generally, "errors"—to be detected. A good way of compressing the resulting array of cards would be to remove the extra row and column entirely, for they can always be regenerated! But this is the point: compression and error detection/correction work in opposite directions. In practice, communication engineers find it best to compress a message first to remove all redundancy from it, and then add just the right kind of redundancy that is needed to combat the kind of error, or "noise," that occurs in the communication channel.

There are other applications of prediction, aside from compression. For example, using the kind of decision tree that might be used to compress text, you can build a computer interface that predicts what the user is going to type next! Obviously predictions cannot always be correct—otherwise users could go away and leave the computer to do their work for them—but they might be correct often enough to support useful communication. The *Reactive Keyboard* is a device that displays several different predictions on a computer "menu," and lets the user select the correct one by pointing at it. While it is unlikely to help a skilled typist, except perhaps when entering documents such as legal contracts that contain a lot of standard "boilerplate" paragraphs, some physically disabled people, for whom regular keyboards are difficult to use, find it a boon.

As we have seen, information theory has a variety of applications in computer science, from finding limits on how much a file can be compressed, to predicting what people are going to type. It is fundamental for a machine whose modern day application is primarily to store, retrieve, and

transmit information.

Further reading

Bell, Cleary and Witten's book on *Text compression* gives a good introduction to the subject of information theory and entropy, although it is aimed at university-level computer science students rather than lay people. Shannon's original work was published in 1949 in a little book entitled *The mathematical theory of communication*, which also includes a non-mathematical, and very readable, account of the subject by Warren Weaver. An article by Donald Knuth called "Supernatural numbers" discusses strategies for guessing arbitrarily large numbers; it appears in the book *The Mathematical Gardner*, edited by D.A. Klarner. *The Reactive Keyboard* and other "keyboard acceleration" devices for the disabled are described in a book by Darragh and Witten.