Activity 17

The Peruvian coin flip—*Cryptographic protocols*

Age group	Older elementary and up.
Abilities assumed	Requires counting, and recognition of odd and even numbers. Some understanding of the concepts *and* and *or* is helpful. Children will get more out of this activity if they have learned binary number representation (see Activity 1, Count the dots), the concept of *parity* (see Activity 4, Card flip magic), and have seen the example of one-way functions in Activity 14, Tourist Town.
Time	About 30 minutes.
Size of group	From individuals to the whole classroom.

Focus

Boolean logic.

Functions.

Puzzle solving.

Summary

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

©Bell, Witten, and Fellows, 1998

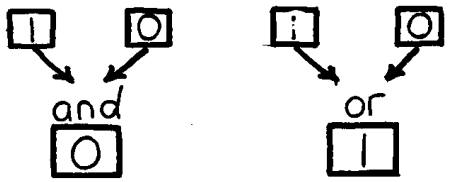


Figure 17.1: An *and*-gate and an *or*-gate

Technical terms

Distributed coin-tossing, computer security, cryptography, cryptographic protocol, *and*-gate, *or*-gate, combinatorial circuit.

Materials

Each group of children will need:

a copy of the reproducible sheet on page 183, and

about two dozen small buttons or counters of two different colors.

What to do

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

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

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

The rules of *and*-gates and *or*-gates are simple. Each "gate" has two inputs and one output (Figure 17.1). Each of the inputs can be either a 0 or a 1, which can be interpreted as *false* and *true*, respectively. The output of an *and*-gate is one (*true*) only if both inputs are one (*true*), and zero (*false*) otherwise. For example, the *and*-gate in Figure 17.1 has a one and a zero on its

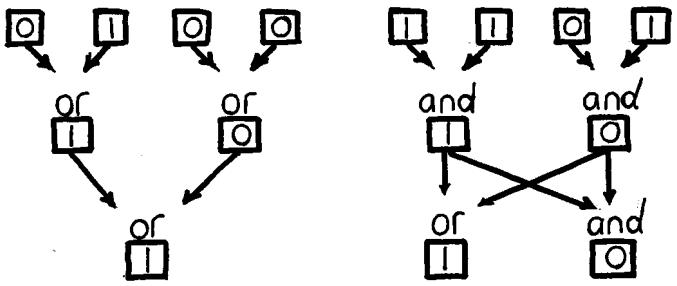


Figure 17.2: Combinations of *or*-gates and *and*-gates

inputs (at the top), so the output (the square at the bottom) is a zero. The output of an *or*-gate is one (*true*) if either (or both) of the inputs is one (*true*), and zero (*false*) only if both the inputs are zero. Thus in Figure 17.1 the output of the *or*-gate is a one for the inputs zero and one.

The output of one gate can be connected to the input of another (or several others) to produce a more complicated effect. For example, in the left-hand circuit of Figure 17.2 the outputs from two *or*-gates are connected to the inputs of a third *or*-gate, with the effect that if any of the four inputs is a one then the output will be a one. In the right-hand circuit of Figure 17.2 the outputs of each of the top two *and*-gates feeds into the lower two gates, so the whole circuit has two values in its output.

For the Peruvian coin flip we need even more complex circuits. The circuit in the reproducible sheet on page 183 has six inputs and six outputs. Figure 17.3 shows a worked example for one particular set of input values.

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

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

2. Show the children how to place counters on the inputs to show the digits that Alicia chooses. Then explain the rules of *and*-gates and *or*-gates, which are summarized at the bottom of the sheet (consider getting the children to color these in).

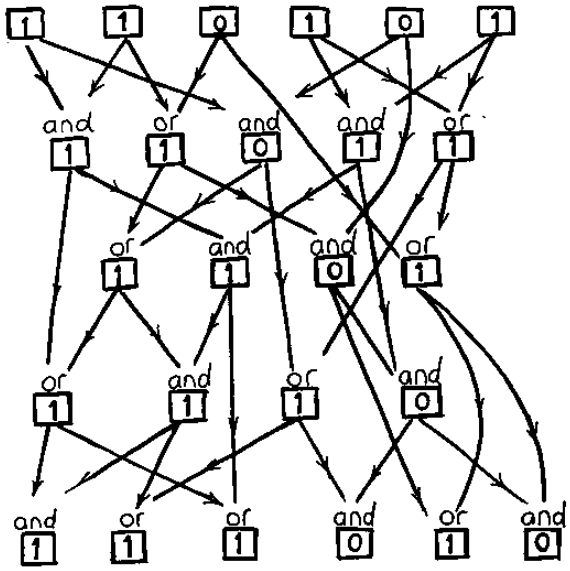


Figure 17.3: An example of Alicia's work

3. Show how to work through the circuit, placing counters at the nodes, to derive the corresponding output. This must be done accurately and takes some care; Table 17.1 (which should *not* be given to the children) shows the output for each possible input for your own reference in case of any doubt.

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

   At this point the coin toss has been completed.

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

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

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

Table 17.1: Input-Output function for the circuit in Figure 17.3

5. See if the children can find a way for Alicia or Benito to cheat.

   From the first line of Table 17.1 you can see that several different inputs generate the output 010010—for example, 000001, 000011, 000101, etc. Thus if Alicia declares the output 010010, she can choose input 000001 if Benito guesses that the parity is even, and 000011 if he guesses that it is odd.

   With this circuit, it is hard for Benito to cheat. But if the output happens to be 011000, then the input must have been 100010—there is no other possibility (you can see this by checking right through Table 17.1). Thus if this is the number that Alicia happens to come up with, Benito can guess even parity and be sure of being correct. A computer-based system would use many more bits, so there would be too many possibilities to try (each extra bit doubles the number of possibilities).

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

Variations and extensions

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

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

2. The children may have noticed that the all-zero input, 000000, is bound to produce the all-zero output, and likewise the all-one input 111111 is bound to produce the all-one output. (There may be other inputs that produce these outputs as well; indeed, there are for the example circuit—000010 produces all zeros, while 110111 produces all ones.) This is a consequence of the fact that the circuits are made up of *and* and *or* gates. By adding a *not*-gate (Figure 17.4), which takes just one input and produces the reverse as output (i.e. $0 \rightarrow 1$ and $1 \rightarrow 0$), the children can construct circuits that don't have this property.

3. Two other important kinds of gate are *and-not* and *or-not*, which are like *and* and *or* but followed by a *not.* Thus a *and-not* b is *not* (a *and* b). These do not allow any functionally

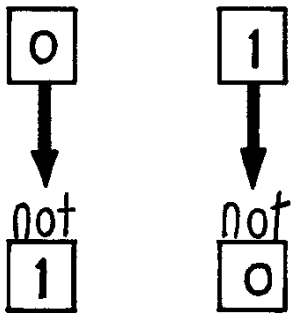


Figure 17.4: The *not*-gate

different circuits to be achieved, since their effect can always be obtained with the corresponding *and* or *or* gate, followed by *not*. However, they have the interesting property that all other gate types can be made out of *and-not* gates, and also out of *or-not* gates.

Having introduced *and-not* and *or-not*, challenge the children to discover whether any of the gates can be made from other gates connected together, and further, if they can be made from just one type of gate connected together. Figure 17.5 shows how the three basic gates, *and*, *or* and *not*, can be constructed from *and-not* gates, in the top row, and *or-not* gates, in the bottom row.

What's it all about?

Recent years have seen huge increases in the amount of commerce being conducted over computer networks, and it is essential to guarantee secure interchange of electronic funds, confidential transactions, and signed, legally binding, documents. The subject of *cryptography* is about communicating in secure and private ways. Two decades ago, computer science researchers discovered the counter-intuitive result that secrecy can be guaranteed by techniques that ensure that certain information is kept *public*. The result is the so-called "public key cryptosystem" of Activity 18, Kid Krypto, that is now regarded as the only completely secure way of exchanging information.

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

In a real situation, Alicia and Benito would not design a circuit themselves, but acquire a computer program that accomplishes the transformation internally. Probably neither would be interested in the innards of the software. But both would want to rest assured that the other is

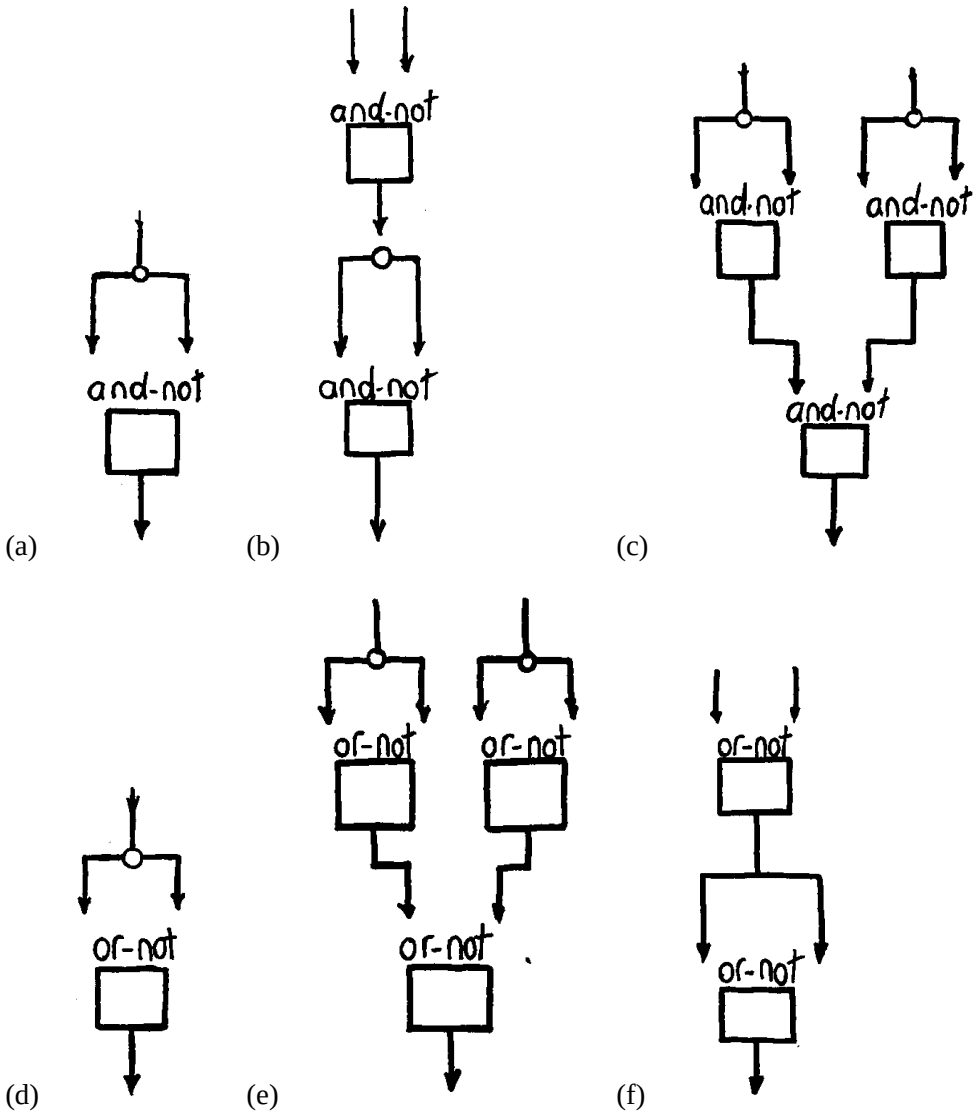


Figure 17.5: Making the three basic gates from *and-not* and *or-not* gates. (a) and (d) are *not*-gates, (b) and (e) are *and*-gates, while (c) and (f) are *or*-gates.

unable to influence the outcome of the decision, no matter how good their computer skills and how hard they tried.

In principle, any disputes would have to be resolved by appeal to a neutral judge. The judge would be given the circuit, Alicia's original binary number, the output that she originally sent Benito, and the guess that Benito sent in return. Once the interchange is over, all this is public information, so both participants will have to agree that this is what the outcome was based on. The judge will be able to put Alicia's original number through the circuit and check that the output is as claimed, and therefore decide whether the decision has been made fairly. Needless to say, the very fact that there is a clear procedure to check that the rules have been followed makes it unlikely that a dispute will arise. Compare with the situation where Alicia flips an actual coin and Benito calls heads or tails—no judge would take on that case!

A circuit as small as the one illustrated would not be much use in practice, for it is easy to come up with Table 17.1 and use it to cheat. Using thirty-two binary digits in the input would provide better protection. However, even this does not *guarantee* that it is hard to cheat—that depends on the particular circuit. Other methods could be used, such as the one-way function introduced in Activity 14, Tourist Town. Methods used in practice often depend on the factoring of large numbers, which is known to be a hard problem (although, as we will learn at the end of the next activity, it is not NP-complete). It is easy to check that one number is a factor of another, but finding the factors of a large number is very time consuming. This makes it more complex for Alicia and Benito (and the judge) to work through by hand, although, as noted above, in practice this will be done by off-the-shelf software.

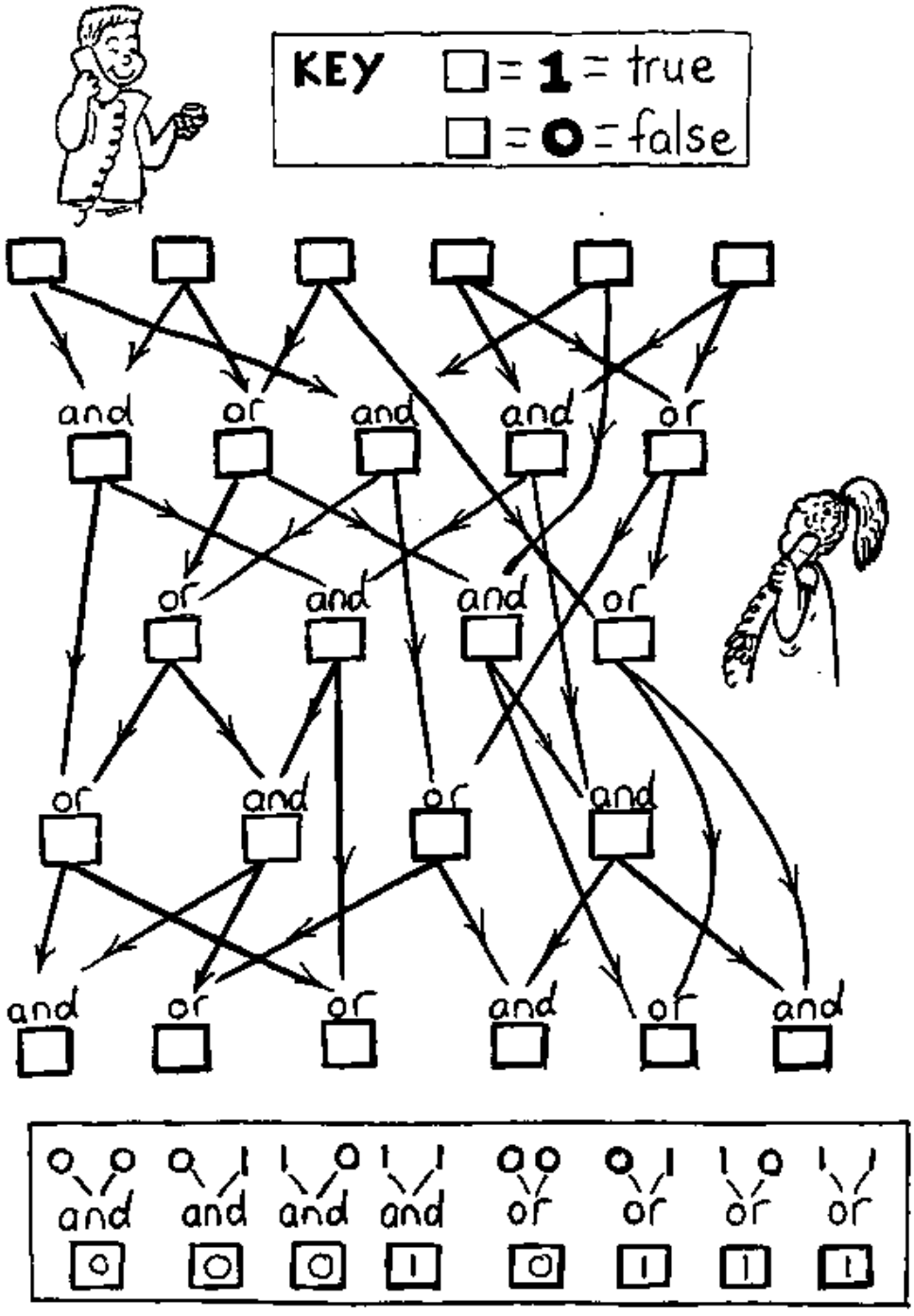
Digital signatures are based on a similar idea. By making public the output of the circuit for the particular secret input that she has chosen, Alicia is effectively able to prove that she is the one who generated the output—for, with a proper one-way function, no-one else can come up with an input that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

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

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

Further reading

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



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

©Bell, Witten, and Fellows, 1998

Activity 14

Tourist town—*Dominating sets*

Summary

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

Curriculum Links

- ✓ Mathematics Level 2: Position and orientation

Skills

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

Ages

- ✓ 7 and up

Materials

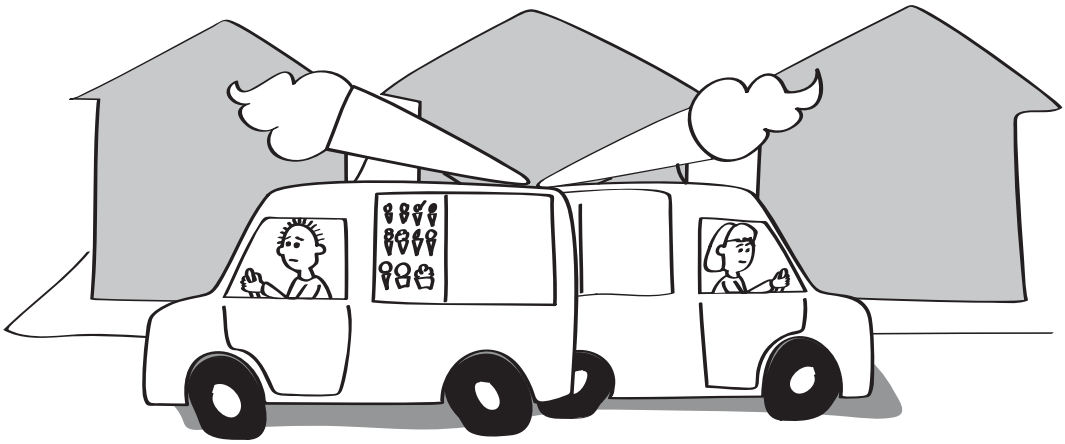
Each group of children will need:

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

You will need

- ✓ an overhead projector transparency of the blackline master *Ice Cream Vans Solution*, or a blackboard to draw it on.

Dominating Sets



Introduction

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

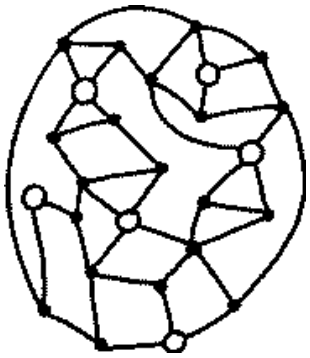
Discussion

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

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

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

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



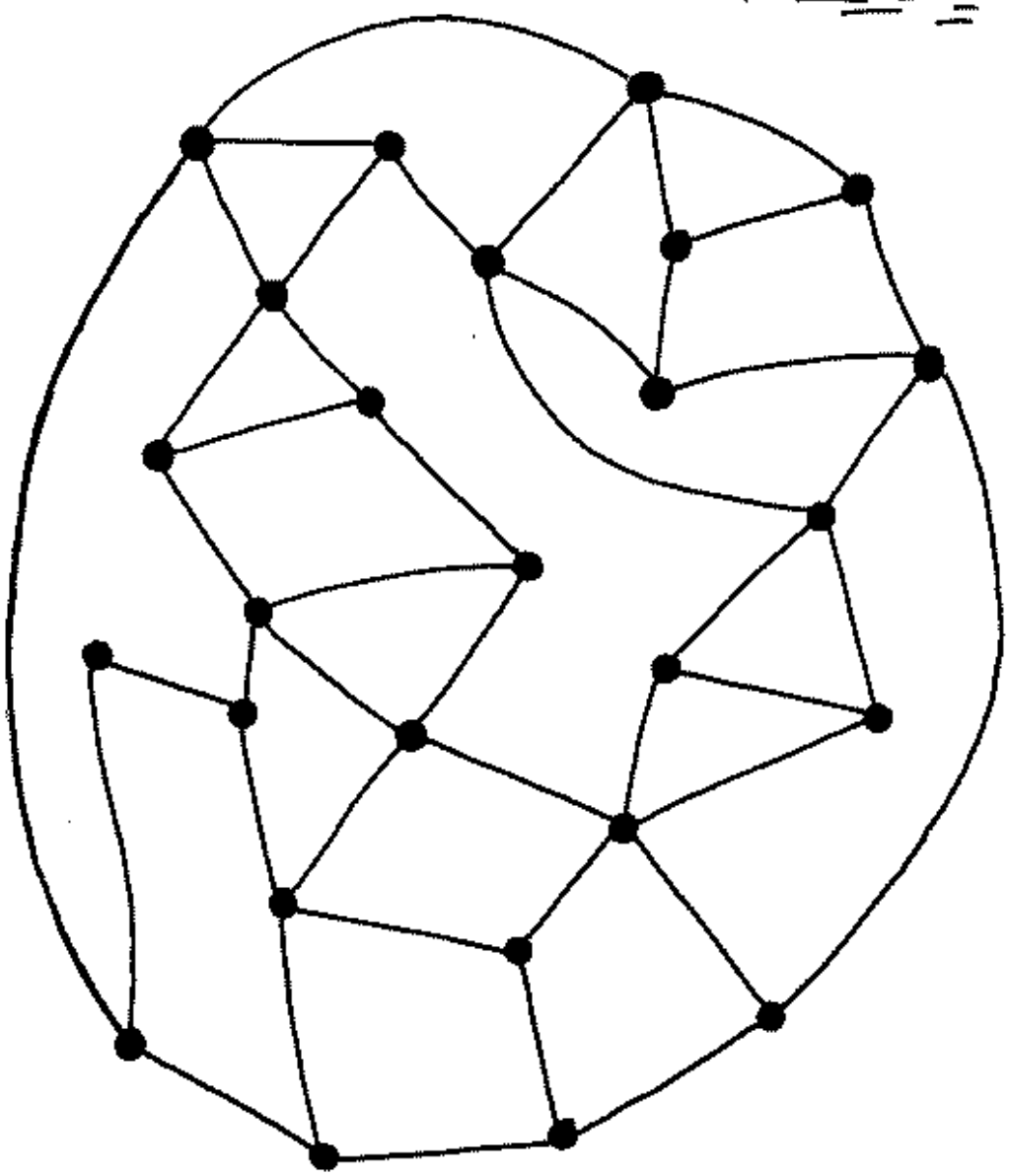
5. The map of Tourist Town was made by starting with the six map pieces at the bottom of the *Ice Cream Vans* worksheet, each of which obviously requires only one ice-cream van, and connecting them together with lots of

streets to disguise the solution. The main thing is not to put any links between the solution intersections (the open dots), but only between the extra ones (the solid dots). Show the class this using an overhead projector, or by drawing it on the board.

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

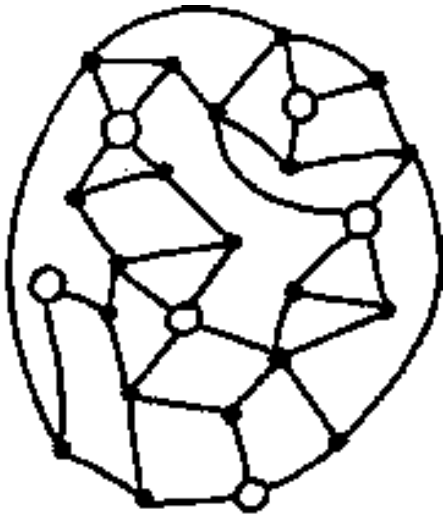
Worksheet Activity: Ice Cream Vans

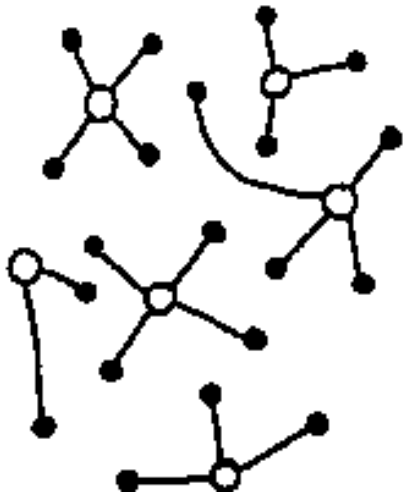
Work out how to place ice-cream vans on the street intersections so that every other intersection is connected to one that has a van on it.



Worksheet Activity: Ice Cream Vans Solution

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





Variations and extensions

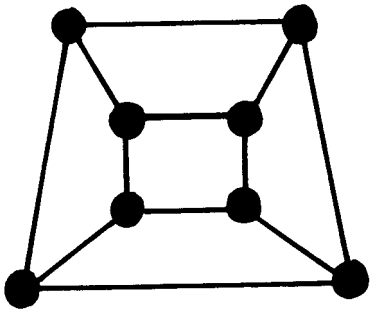
There are all sorts of situations in which one might be faced with this kind of problem in town planning: locating mailboxes, wells, fire-stations, and so on. And in real life, the map won't be based on a trick that makes it easy to solve. If you really have to solve a problem like this, how would you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

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

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

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

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



What's it all about?

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

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

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

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

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

Further reading

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

Activity 18

Kid krypto—*Public-key encryption*

Age group	Junior high and up.
Abilities assumed	This is the most technically challenging activity in the book. While rewarding, it requires careful work and sustained concentration to complete successfully. Children should already have studied the example of one-way functions in Activity 14, Tourist Town, and it is helpful if they have completed the other activities in this section (Activity 16, Sharing Secrets, and Activity 17, the Peruvian coin flip). The activity also uses ideas covered in Activity 1, Count the dots, and Activity 5, Twenty guesses.
Time	About 30 minutes.
Size of group	Requires at least two people, can be done with a whole class.

Focus

Puzzle solving.

Secret codes.

Summary

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

©Bell, Witten, and Fellows, 1998

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

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

Sounds impossible? Read on.

Technical terms

Public-key cryptosystems, encryption, decryption, NP-complete problems.

Materials

The children are divided into groups of about four, and within these groups they form two subgroups. Each subgroup is given a copy of the two maps on page 193. Thus for each group of children you will need:

two copies of the blackline master on page 193.

You will also need:

an overhead projector transparency of page 194, and

a way to annotate the diagram.

What to do

Amy is planning to send Bill a secret message. Normally we might think of secret messages as a sentence or paragraph, but in the following exercise Amy will send just one character — in fact, she will send one number that represents a character. Although this might seem like a simplistic message, bear in mind that she could send a whole string of such "messages" to make up a sentence, and in practice the work would be done by a computer. And sometimes even small messages are important — one of the most celebrated messages in history, carried by Paul Revere, had only two possible values.

We will see how to embed Amy's number in an encrypted message using Bill's public lock so that if anyone intercepts it, they will not be able to decode it. Only Bill can do that, because only he has the key to the lock.

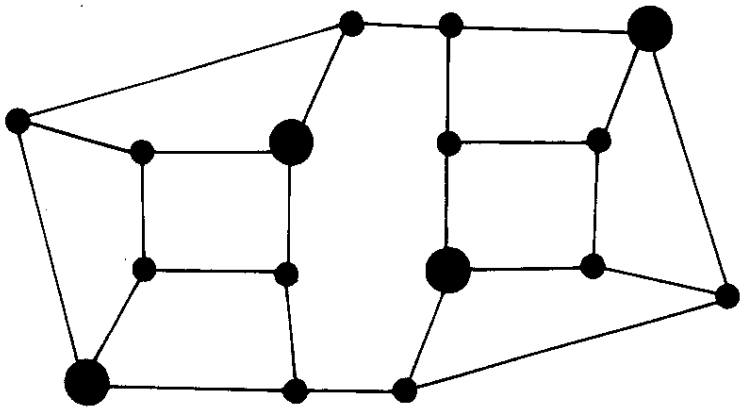


Figure 18.1: Bill's private map

We will lock up messages using *maps*. Not Treasure Island maps, where X marks the spot, but street maps like the one of Tourist Town on page 149, where the lines are streets and the dots are street corners. Each map has a public version—the lock—and a private version—the key.

Shown on page 194 is Bill's public map. It's not secret: Bill puts it on the table (or a web page) for everyone to see, or (equivalently) gives it to anyone who might want to send him a message. Amy has a copy; so has everyone else. Figure 18.1 shows Bill's private map. It's the same as his public map, except that some of the street corners are marked as special by enlarging them. He keeps this version of the map secret.

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

1. Put Bill's public map (page 194) on the overhead projector. Decide which number Amy is going to send. Now place random numbers on each intersection on the map, so that the random numbers add up to the number that Amy wishes to send. Figure 18.2 gives an example of such numbers as the upper (non-parenthesised) number beside each intersection. Here, Amy has chosen to send the number 66, so all the unbracketed numbers add up to 66. If necessary, you can use negative numbers to get the total down to the desired value.

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

   Instead, choose any intersection, look at it and its three neighbors—four intersections in

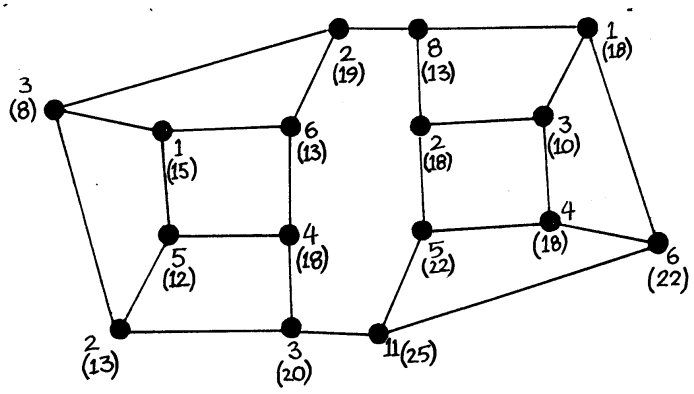


Figure 18.2: Amy's calculations, using Bill's public map

all—and total the numbers on them. Write this number at the intersection in parentheses or using a different color of pen. For example, the rightmost intersection in Figure 18.2 is connected to three others, labeled 1, 4, 11, and is itself labeled 6. Thus it has a total of 22. Now repeat this for all the other intersections in the map. This should give you the number in parentheses in Figure 18.2.

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

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

4. Only someone with Bill's private key can decode the message to find the message that Amy originally wanted to send. On the coded message mark the special enlarged nodes in Bill's private map (Figure 18.1).

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

5. How does it work? Well, the map is a special one. Suppose Bill were to choose one of the marked intersections and draw around the intersections one street distant from it, and repeat the procedure for each marked intersection. This would partition the map into non-overlapping pieces, as illustrated in Figure 18.3. Show these pieces to the children by drawing the boundaries on the map. The group of intersections in each partition is exactly the ones summed to give the transmitted numbers for the marked intersections, so the sum of the four transmitted numbers on those intersections will be the sum of all the original numbers in the original map; that it, it will be the original message!

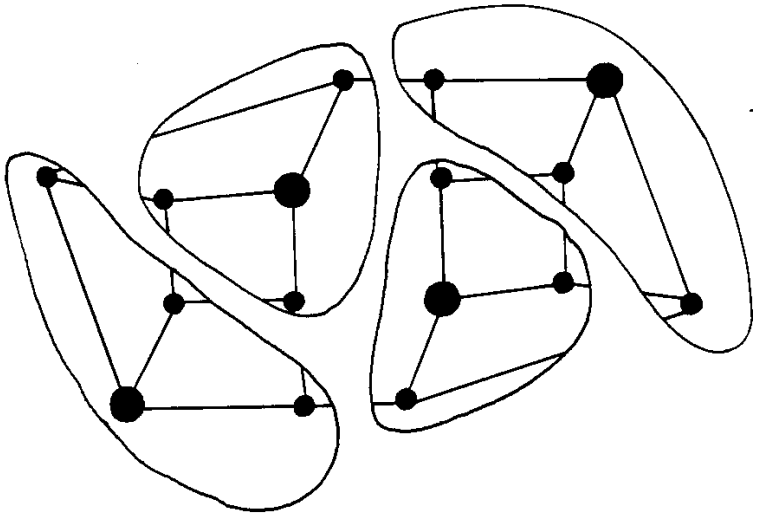


Figure 18.3: The regions that comprise Bill's private map

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

Perhaps your class would like to know that they have joined the very select group of people who have actually worked through a public-key encryption example by hand—practising computer scientists would consider this to be an almost impossible task and virtually no-one has ever done it!

Now, what about eavesdropping? Bill's map is like the ones in the Tourist Town activity (Activity 14), where the marked intersections are a minimal way of placing ice-cream vans to serve all street corners without anyone having to walk more than one block. We saw in Tourist Town that it's easy for Bill to make up such a map by starting with the pieces shown in Figure 18.3, and it's very hard for anyone else to find the minimal way to place ice-cream vans except by the brute-force method. The brute-force method is to try every possible configuration with one van, then every configuration with two vans, and so on until you hit upon a solution. No-one knows whether there is a better method for a general map—and you can bet that lots of people have tried to find one!

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

6. Having been through one example with the whole class, divide the children into groups of, say, four. Give each pair of each group the public map on the blackline master on

page 193. Each pair should choose a "message" (any integer), encode it with the public key, and give the resulting map to the other group. The other group can try to decode it, but they are unlikely to be successful until they are given (or work out!) the private map. Then give out the private map and see if they can now decode it correctly.

7. Now each pair can design their own map, keeping the private version secret and giving the public version to the other pair—or indeed "publishing" it on the classroom board. The principle for designing maps is just the same as was discussed in the Tourist Town activity, and extra streets can be added to disguise the solution. Just be careful not to add extra streets into any of the "special" points. That would create an intersection from which *two* ice-cream vans could be reached in one hop, which is all right for the tourist town situation but would cause havoc when encrypting. That is because the special points no longer decompose the map into *non-overlapping* pieces, as illustrated in Figure 18.3, and this is essential for the trick to work.

What's it all about?

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

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

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

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

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

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

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

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

What are the factors of the 100-digit number 9,412,343,607,359,262,946,971,172,136,294, 514,357,528,981,378,983,082,541,347,532,211,942,640,121,301,590,698,634,089,611,468, 911,681? Don't spend too long! They are 86,759,222,313,428,390,812,218,077,095,850,708,048, 977 and 108,488,104,853,637,470,612,961,399,842,972,948,409,834,611,525,790,577,216,753. There are no other factors: these two numbers are prime. Finding them is quite a job: in fact, it's a several-month project for a supercomputer.

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

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

How secure is the system based on prime numbers? Well, factoring large numbers is a problem that has claimed the attention of the world's greatest mathematicians for several cen-

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

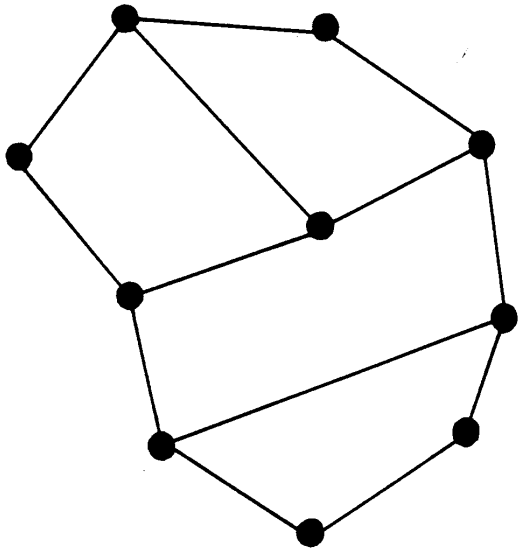
Another worry is that if there are just a few possible messages, an interloper could encrypt each of them in turn using the public key, and compare the actual message with all the possibilities. Amy's method avoids this because there are many ways of encrypting the same message, depending on what numbers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

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

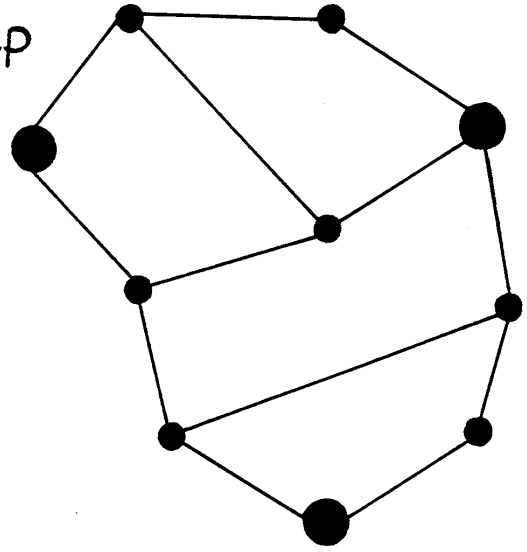
Further reading

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

public Map

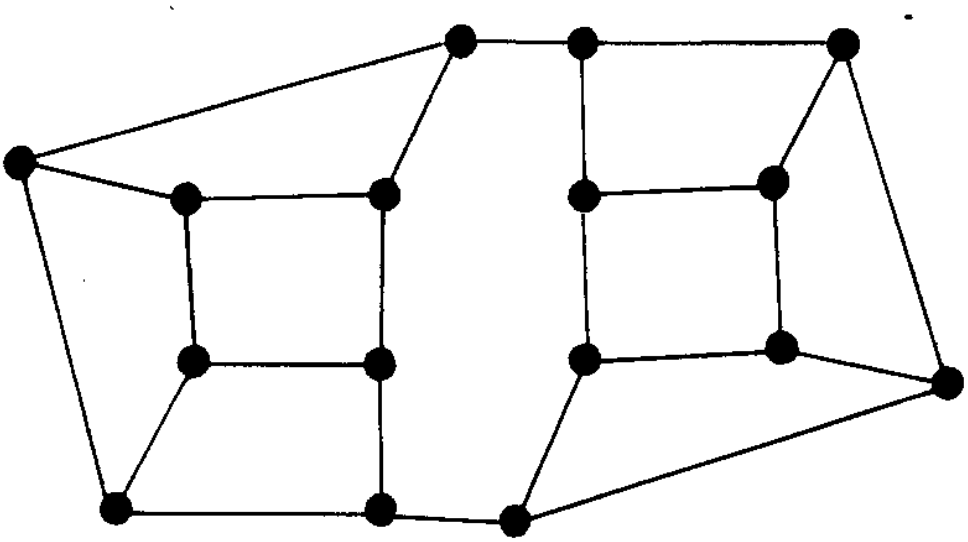


private Map



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

©Bell, Witten, and Fellows, 1998



Instructions: *Put this blackline master on an overhead projector transparency and use it to demonstrate the encoding of a message.*

©Bell, Witten, and Fellows, 1998