CS4FN

A Computer Science for Fun/Teaching London Computing/CHI+MED Special

Computational Thinking: Puzzling Tours

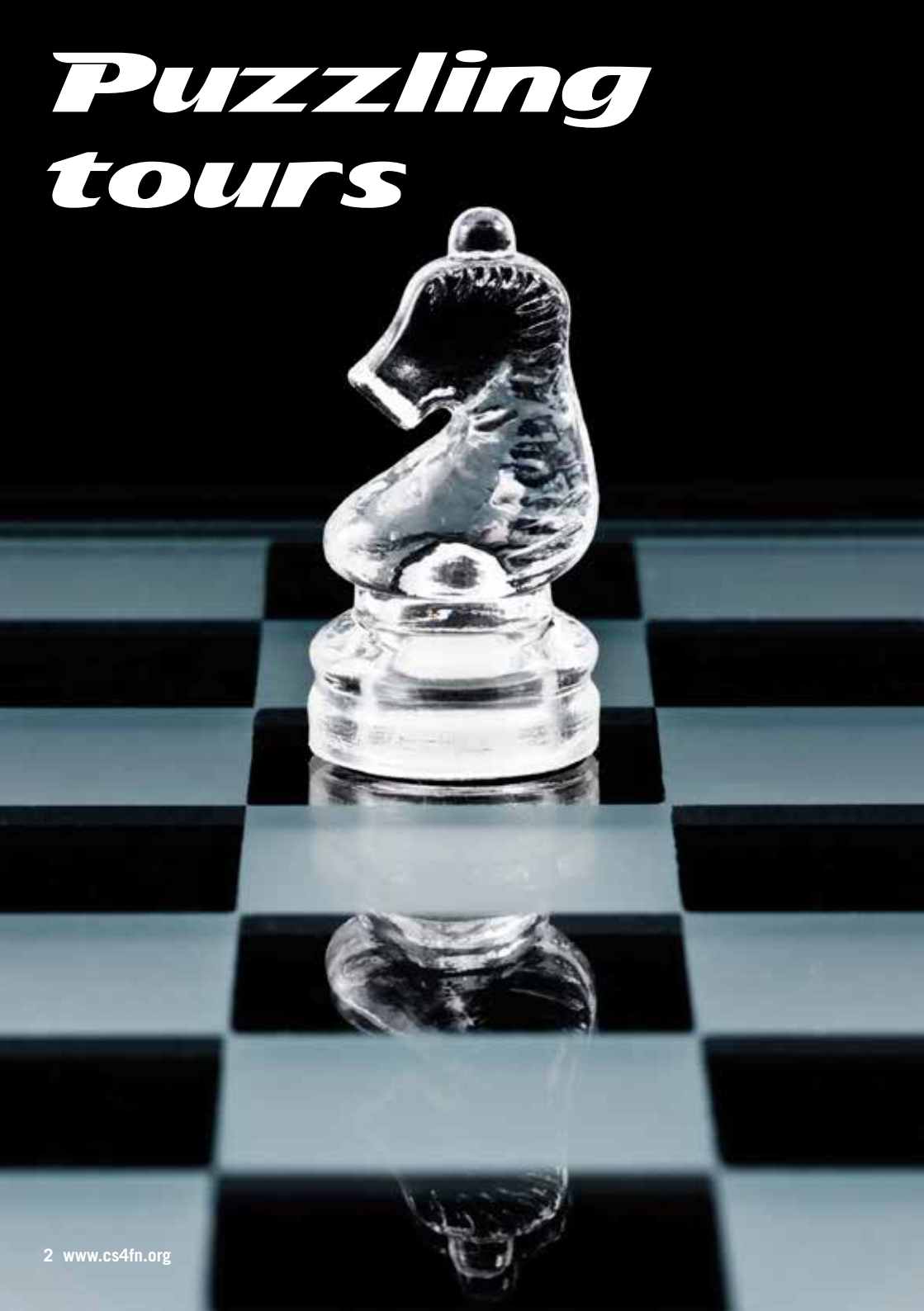
The Knight's Tour puzzle

The Tour Guide puzzle

Explore abstraction and representation

Queen Mary University of London

Puzzling tours



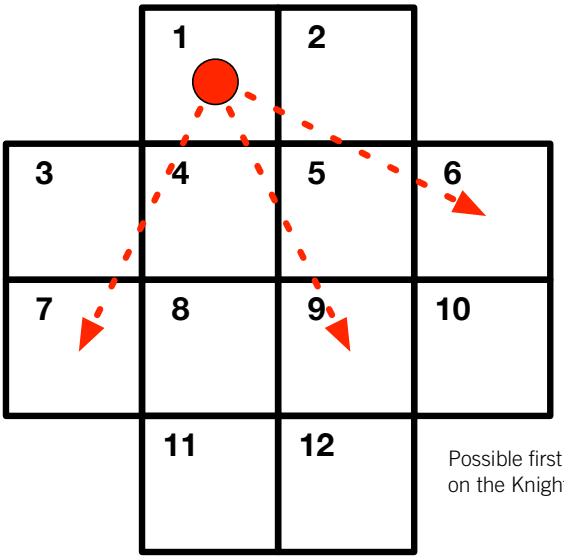
Three puzzles

Find a way for a knight to visit every square on a board exactly once, and then solve a problem for a city tour guide. In doing so, find out what computational thinking is all about. See how algorithms are at its heart, allowing computer scientists to solve a problem once, and then - as long as they have checked it carefully - avoid having to think about it ever again. See why computer scientists think hiding things makes their life easier, especially when they find a good way to represent information, and how an ability to match patterns lets the lazy computer scientist do no more work than absolutely necessary. Oh, and finish off by advising a tourist information centre using some logical thinking. Perhaps surprisingly, computer science is not just about computers: it is about computation, and that crops up in all sorts of situations. Above all, it is about thinking in a completely different way.

The Knight's Tour puzzle

In the Knight's Tour puzzle, a single chess knight is able to move on a small, cross-shaped board. A knight can move two spaces in one direction and then move one square at right angles, or vice versa as shown. It jumps to the new square without visiting any in between, and must always land on a square on the board.

You must find a sequence of moves that starts from square 1, visits every square exactly once by making a knight's moves, and finishes where it started.



Possible first moves on the Knight's Tour

Solve it!

Try to solve the Knight's Tour puzzle and time yourself to see how long it takes you. You must do more than just get the knight to do a correct tour, though: you must find an algorithmic solution. That means more than just moving the piece around the board. You must record the series of moves that works: that is, you must write an **algorithm** that solves the puzzle. Your algorithm could just be written as a list of the numbers of the squares to visit in the order they should be visited. Or you could write the algorithm as a series of commands like: 'move from square 1 to square 9'- it's up to you.

Once you have an algorithm that works, double check that it really is a solution by trying it out. Follow your instructions, marking the squares as the knight visits them. That way, you can be sure that it doesn't break the rules: that it visits every square exactly once.

If you solve the puzzle, then well done! If you can't, don't worry – we will see how to make it easier to do later.

Algorithmic thinking

This puzzle involves two skills that matter to computer scientists, two aspects of what they call computational thinking: **algorithmic thinking** and **evaluation**. Algorithmic thinking is based on the idea that we only have a solution to a problem when we have an algorithm that can do it. If we do have an algorithm, then we can write a program to do it. Computer programs are just algorithms written out in a programming language. **Evaluation** is about checking algorithms really do work – if you are going to get a computer to do things for you, you need to check in advance that the instructions you give them will do the right thing. Always. We will see later that problems like this involve several more computational thinking skills, but first, let's try a simpler puzzle.

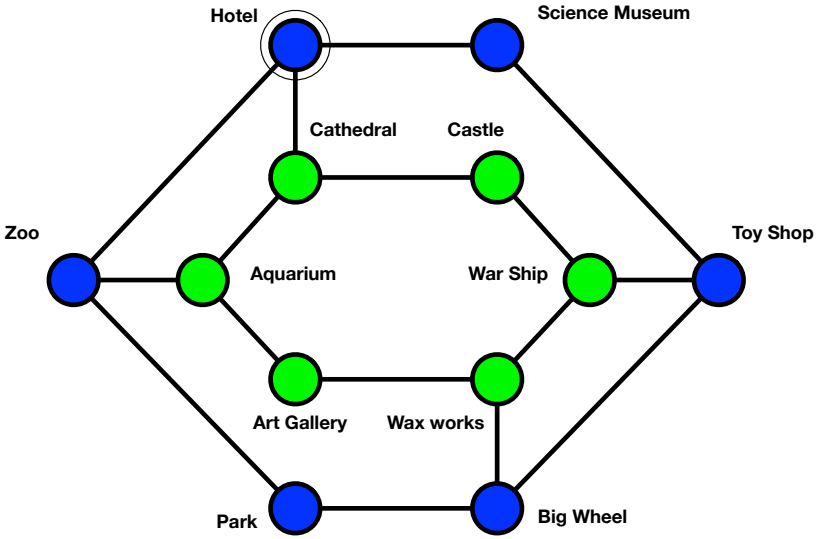
The Tour Guide puzzle

You are a hotel tour guide. Tourists staying in your hotel expect to be taken on a tour visiting all the city's attractions. You have been given an underground map that shows all the locations of the attractions and how you can get from one to another using the underground network.

You must work out a route that starts from the hotel and will take your tour group to every tourist site. The tourists are only in the city for the day, so don't want to waste time. They will be unhappy if they pass through the same place twice. Obviously, they also want to end up back at their hotel that evening.

As with the Knight's Tour, your task is to come up with an algorithmic solution. Check that your solution works. Time yourself to see if it is harder or easier than the Knight's Tour.

How long did it take you? And was it easier (ie you solved it faster) than the Knight's Tour (if you solved that at all)?



What's required?

Why is it important that you check that you definitely do have a correct solution? Well, you wouldn't want to actually do a tour and find at the end of the day that you missed something important. You don't want to have to deal with angry tourists!

One way to check an algorithm is to do what computer scientists call **dry running** (or **tracing**) your algorithm. That simply means that you follow the steps of the algorithm on paper before you do it for real. That is probably how you checked your solution for the Knight's Tour if you came up with one. For the Tour Guide puzzle, you can draw the route on the map as you follow each instruction, ticking each location as you visit it.

Of course, as a real tour guide, you wouldn't just rely on checking the route on paper. You would then go out and test it for real too, but it can save a lot of time to check it on paper first. Programmers do the same thing. They check their program works on paper (they 'dry run' it) but then also check it for real – that is called **testing**. Just as you did for the puzzle, programmers test their programs to make sure they always work.

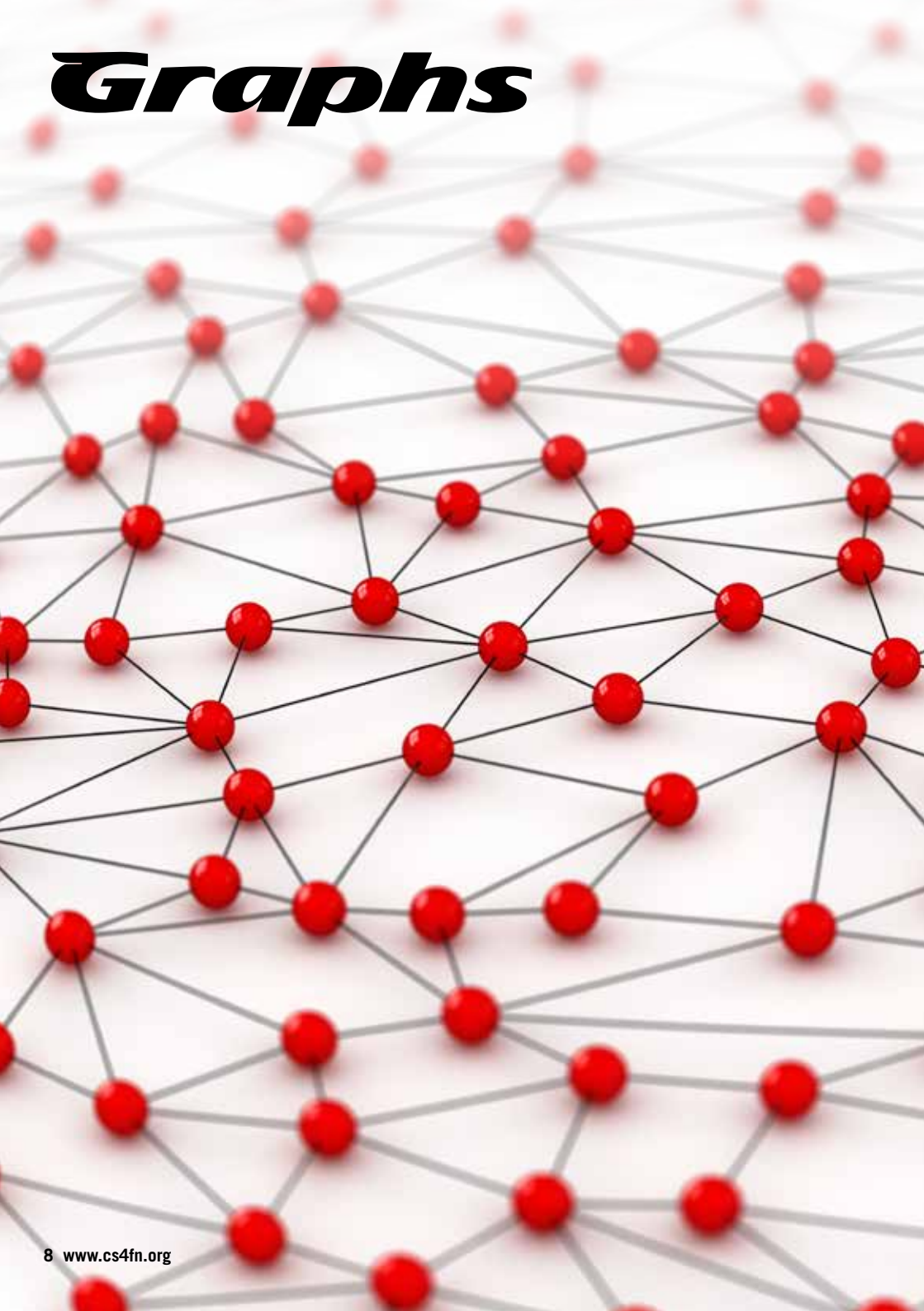
We can actually be a bit more precise about our evaluation. We can work out exactly what properties matter for us to have a correct solution. If we write a list of those necessary properties, we can then tick them off as we check our solution meets them. Computer Scientists call properties like this **requirements**.

For the Tour Guide puzzle, we need to check our answer against the following requirements:

- The tour starts at the hotel.
- It visits every location.
- It does not pass through a location already visited.
- It ends at the hotel.

Go back and write a list of requirements for the Knight's Tour too. Perhaps you can see similarities? We will come back to that.

Graphs



Why is it easy?

You probably found the Knight's Tour puzzle harder to solve, but actually it doesn't need to be any harder at all. It can be solved really easily if you use some more computational thinking tricks.

Why is the Tour Guide problem easy? The Underground map shows the information that matters clearly, ignoring the detail that doesn't matter. It is a good **abstraction** of the problem, in that it makes the solution easy to see. Without the map, it would have been harder, even if we knew which stations linked to which. The Tube map is a special way of representing the information that we know about the problem we have to solve. It is a special kind of diagram called a **graph**. A graph to a computer scientist is a series of dots (we call those the **nodes** of the graph) and lines that join them (we call those the **edges** of the graph). The nodes and edges **represent** something about the data that we are interested in. The edges show which nodes are linked in the way that we are interested in for the problem. The tourist attractions are presumably linked by roads too, but in different ways. The road graph would be different. That is the graph we would need if we were running a coach tour!

Ignore it!

We are interested in the tourist attractions (our nodes) and which ones are linked to each other by the underground connections (our edges). We aren't interested in anything else about the places so we ignore everything else. We hide the exact locations, how far they each are apart, road links and a lot more that doesn't matter to our problem of finding an Underground route that visits them all. The graph is an **abstraction** of the real city. We have hidden all the extra detail that we don't need when we create the graph. The graph only shows the information that matters. That makes it much clearer to see the information we need to use to solve the problem.

Simplifying things

Graphs are often used to represent information about the connections between things. You will find them on the signs at bus stops, and showing the routes on train and Underground maps. They are a very good **representation** in situations where you want to find routes from place to place as we did here. The simplified graph makes it easier to find a route than if we had a fully accurate and detailed map, as then the information that mattered would be hard to see amongst all the detail.

Cycles

Computer scientists actually have a special name for this kind of tour of a graph where you visit every node exactly once before returning back to the start. They call it a **Hamiltonian cycle**, named after an Irish physicist, William Rowan Hamilton. He invented a puzzle that involved travelling to every corner of a 3-dimensional shape called a dodecahedron by travelling along its edges: a Hamiltonian cycle.

Back to the beginning

You may have noticed by now that the Knight's Tour and Tour Guide problems are very similar. If you wrote out the requirements for the Knight's Tour, you may have seen they were essentially the same for both puzzles:

- The tour starts at a given point
- It must visit every point
- It must not pass through a point already visited
- It must end at the point it started at.

Both puzzles are asking you to find a Hamiltonian cycle! What we have just done is a computational thinking trick. We have **generalised** both problems to be the same kind of problem. We did it by pattern matching – seeing the essential similarities. We abstracted away detail like whether it is hotels and tourist attractions, and whether you move using a knight's moves or by following Tube lines.

So, if the reason the Tour Guide puzzle was easy was that we had a map – we represented the problem as a graph – then why don't we represent the Knight's Tour problem as a graph too?

We need to do a further **abstraction** of the problem. There are two things to realise. First of all, it doesn't actually matter how the board is laid out – we don't care that the squares of the board are squares for example – they could be any shape and size. Let's draw each square as a small circle instead, just as the tourist attractions were circles on the underground map. They are just nodes of a graph.

Secondly, seeing which squares are actually physically next to one another doesn't actually matter for the puzzle either. The only thing that matters is which ones you can jump between using a knight's move. So let's draw lines between any two dots when you can use a single knight's move to jump between them. That is just like the way the Underground map shows which attractions can be jumped between using the Underground. They are the edges of a graph.

Creating the graph

To create the graph for the Knight's Tour puzzle, move from square to square drawing lines and circles (edges and nodes). Start with square 1 – draw a circle and label it 1. Now, from square 1, you can move to square 9, so draw another circle and label it 9, drawing a line between them. From square 9, you can only move back to 1 or on to square 3, so put a new circle marked 3 and draw a line to it from spot 9.

Keep doing this until you get back to a spot you have already drawn. Then go back to square 1 and follow another trail. For example: from square 1, you can also move to square 7, so if you have a spot 7 already, then draw a line to it. If not, draw a new spot marked 7 and again draw a line to it, then continue the trail. Once you have followed all trails from spot 1, move to spot 2 and follow all trails from it in the same way (adding spot 2 if it's not already there). Then move on to drawing trails from spot 3. Keep doing this until you have covered all trails from all spots.

When you have finished you have created a map of the Knight's Tour problem.

HINT: there are only two moves possible from each of the inner three squares so their spots will each have two lines out of them in the finished map. There are three moves possible from all the other squares so their spots will have three lines out of them.

Go for depth

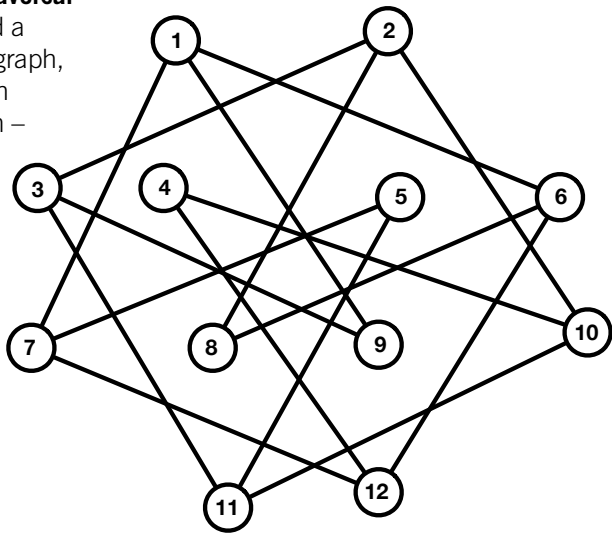
Go for depth

This method of exploring all the possible moves needed to draw the graph is a variation of what is called a **depth first search** of the graph: we explore paths to their end, following the trail 1 – 9 – 3 – 11 to the end, before backing up and trying different paths. An alternative (called **breadth first search**) would involve drawing all the edges from a node before moving on to a new node. So, for breadth first search, we would draw all the edges from node 1. Then we might draw all the edges from node 9, followed by all the edges from node 6, and so on. These are two different algorithms for exploring graphs exhaustively: two different **graph traversal algorithms**. Once you have realised a problem can be represented as a graph, you can use these algorithms as an organised way to explore the graph – and so, the problem.

Nice and neat

If the drawing you end up with is a bit messy with lots of lines crossing each other like the one below, you may want to redraw it neatly with no lines crossing. It can be done very neatly as two linked hexagons one inside the other. A version is given overleaf.

Once you have drawn the graph, try and solve the Knight's Tour puzzle again. Start at node 1 and follow the lines, noting the nodes you pass through. It should be fairly easy to come up with a solution.

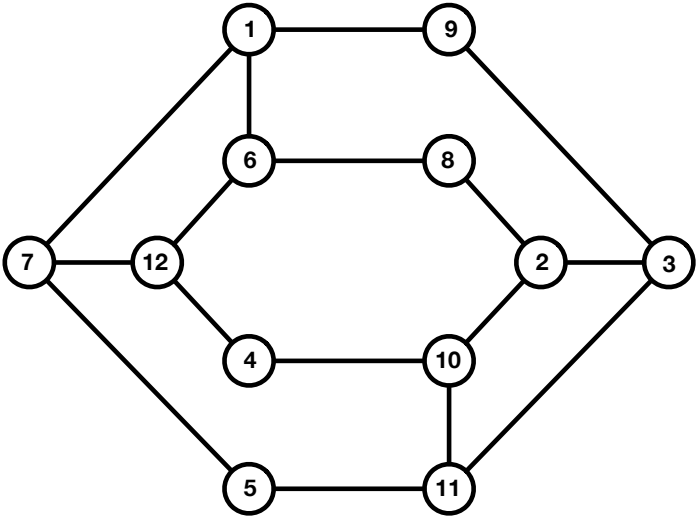


Solving problems

Same problem, same solution

Now look at your graph carefully. With a bit of care about how you draw it, you should be able to actually draw it so it looks exactly the same as the Tube map. The only difference will be in the labels attached to the nodes. They will be numbers instead of names of places.

What this shows is that we can actually **generalise** these two problems to be exactly the same problem, not just the same kind of problem. If you have a solution to one (an algorithm that solves it), then you immediately have a solution to the other too! All you have to do is re-label the graph. A generalised version of the algorithm will solve both. You don't actually have to solve it anew.



Mapping between maps

The table below tells you how to re-label a graph describing one of our two problems into a graph describing the other. It also tells you how to convert a solution for one problem into a solution for the other. For each step in your answer to one, all you have to do is look it up and swap it for the corresponding label.

So if we came up with the following solution to the Knight's Tour:

1-9-3-11-5-7-12-4-10-2-8-6-1

then – using the table – we immediately get a solution to the Tour Guide.

Hotel – Science Museum – Toy Shop – Big Wheel – Park – Zoo – Aquarium – Art Gallery – Wax Works – War Ship – Castle – Cathedral – Hotel

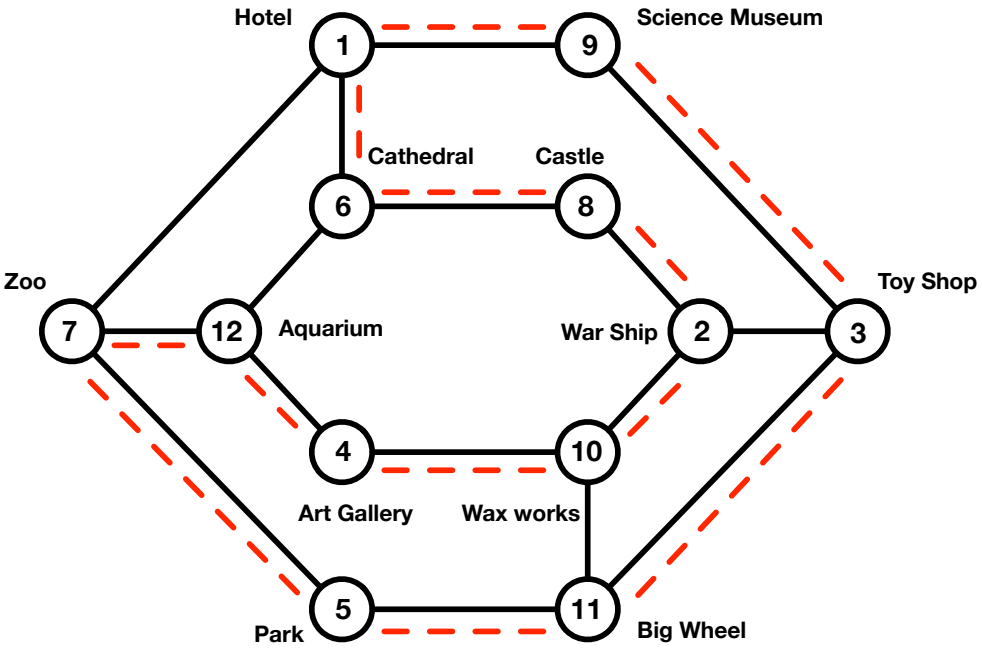
Knight's Tour Square	Tour Guide Attraction
1	Hotel
2	War ship
3	Toy Shop
4	Art Gallery
5	Park
6	Cathedral
7	Zoo
8	Castle
9	Science Museum
10	Wax Works
11	Big Wheel
12	Aquarium

A map for both problems

You can see the mapping in this version of the map that shows one solution.

Of course, as there are many solutions possible, the actual solutions you came up with might be different – but if so, both solutions will solve both puzzles.

Perhaps surprisingly, the two apparently different problems are actually exactly the same problem with exactly the same solution (once generalised). Once you have solved one, you have solved both!



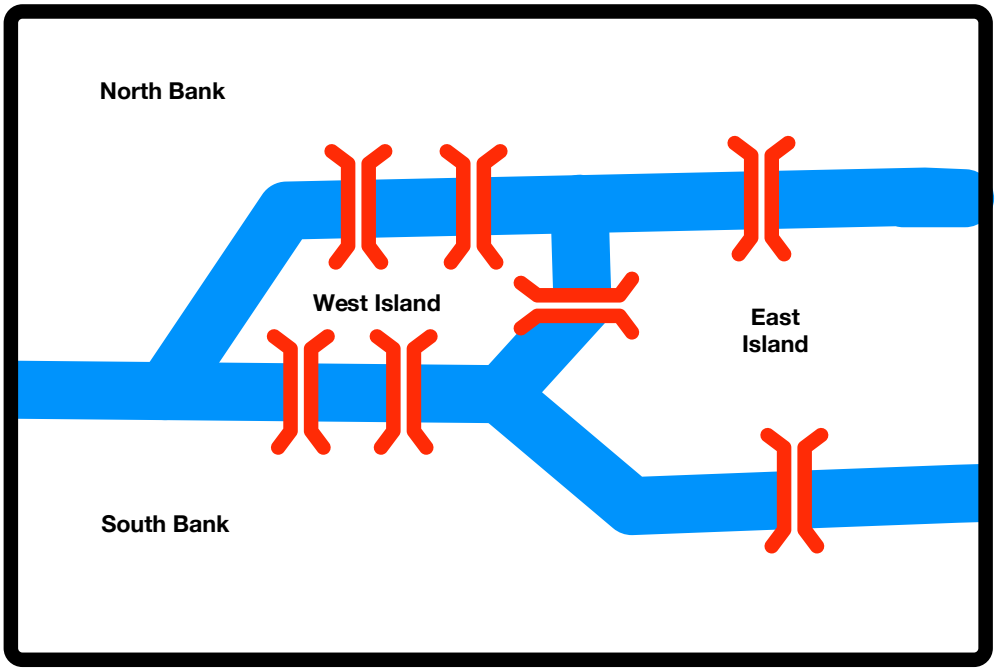
The Bridges of Königsberg

Here is a final puzzle to think about. On the opposite page is a map of the city of Königsberg, showing the river that runs through the middle, its two islands and the seven bridges that cross the river.

The tourist information centre would like to publish a route that visits each part of the city (both banks and both islands) and that crosses each bridge once (and no more). It should start and end in the same place. You have been asked to advise them. Either provide a route or if you can't, explain why not.

A variation of this puzzle was solved by the mathematician Leonard Euler in the 18th century. His solution introduced the idea of graphs in the first place. This ultimately led to them becoming a key computational thinking tool of both mathematicians and computer scientists. Victorian computer scientists Charles Babbage and Ada Lovelace are known to have had a go at solving the puzzle in the 19th century.

Draw a graph of the problem and see if you can solve it. The answer is given at the end of the booklet.


North Bank
West Island
East
Island
South Bank

Computational Thinking

Algorithms

Each route you came up with is a **sequence of instructions** that can be followed to visit every tourist attraction or square of the board before going back to the start. It is a simple **algorithm** for doing a tour of the city or of the board. There are several different routes you could take – several different algorithms can be solutions to the same problem.

Why is it important to write down an algorithm when we solve a problem? Well, once we write the algorithm down, we can follow it as many times as we want (eg we could give tours over and over again with no more problem solving work). We can even give it to someone else to follow: your junior assistant perhaps, assuming you are the Tour Company Manager, or just an individual tourist. Then you won't need that person to have to work a route out for themselves. Algorithms can also be turned into programs – and then a computer can do the work instead.

Once we have written down an algorithm, it is important that we **evaluate** it. We must check that it works. In particular, that means that we must check that the algorithm meets a set of properties that describe the problem. These are known as **requirements**.

Representation

We can make a problem easier to solve by choosing a good **representation** of it. The representation that we used is a kind of diagram that a computer scientist calls a **graph**. A graph is a series of spots (we call those the **nodes** of the graph) and lines that join them (we call those the **edges** of the graph). We turned the Knight's Tour puzzle into a graph by having a node for each square of the board. We then added an edge for each possible knight move. It is this **change of representation** of the data that makes the puzzle easier.

To create the graph, we had to do an **abstraction**. Abstraction is just the hiding of information. To think of it another way, you must change the **representation** of the puzzle – change the way that it is presented, change what the board looks like and how the moves are done – to make the things that matter in the puzzle clearer. The positions of the board and how you can move between them are the only things that matter, so that is the abstraction we use – we hide all other information like the shape, size and position of the 'squares' of the board.

Computational Thinking

Generalisation

We also saw two examples of **generalisation**. First of all, we could generalise both problem statements and see that they are really the same kind of problem that involves finding a series of moves that visit every point exactly once before returning to the start. Secondly, both problems could be represented as a graph (and actually the same one). A graph is a general representation – lots of apparently different problems can be represented by graphs. Then any algorithm we work out that applies to graphs can be used on any of these different problems.

For our two problems, because the graphs are the same and the requirements are the same, we could generalise the solutions too. The solutions to both problems turn into exactly the same series of steps of the graph up to the words/numbers we have used to label the nodes. That, of course, isn't always the case for different problems. The graphs or requirements - and so solutions - of two problems could be very different.

Pattern matching

Spotting when two problems are the same (or very similar) is an important part of computational thinking called **pattern matching**. It saves work as it allows us to avoid redoing essentially the same thing every time we are given a new problem.

When you see a problem or puzzle about the way different places are linked, think about graphs. Another way of saying this is to see if you can **match** a problem to the **pattern** of moving from point to point, then use a graph to represent it. The points don't have to be physical places. They could be web pages (with hyperlinks between them), alarm clock states (with button presses that move between them), cities (with flights between them) or much more.

If we find that two graphs match, we may realise that we already have the answer. All we have to do is transform the graphs to be the same thing by swapping the labels over, and we can transform the answer of one into the answer of the other. We then just read off an answer to both problems from the general solution.

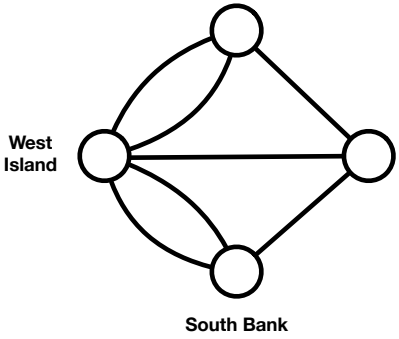
Logical Thinking

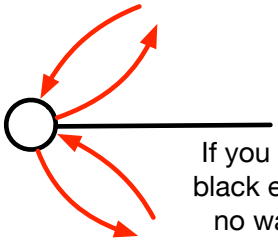
Another core part of computational thinking is being able to think logically. A good representation helps with this by removing the clutter so that you can focus on what matters. That is exactly what Leonard Euler found with the Bridges of Königsberg puzzle when he came up with the idea of drawing a graph. He then combined it with some very clear thinking.

What he realised looking at the graph was that it was impossible to come up with a route. Why? Any suitable route must visit every node. It must also involve every edge but only once (as they are the bridges and the route must cross every bridge once). Let's suppose there is such a route and we draw a red line over the graph to show it. All the edges must be on the route so should be coloured red. Now think about a node on that route. It must have a red line in to it for every red line out from it. Otherwise, the route will get stuck when it arrives on that extra edge. It will have no way out without going back over a bridge already crossed. The same reasoning applies to every node. That means all nodes must have an even number of edges connected to them if there is such a route.

All the nodes on the Königsberg graph have an odd number of edges, so there is no such tour possible.

Find out more at:
www.cs4fn.org/ada/puzzlingdoodles.php





CS4FN

Teaching London Computing:
www.teachinglondoncomputing.org

Computer Science for Fun:
www.cs4fn.org

This booklet was written by Paul Curzon, Queen Mary University of London, June 2015, with support from the CHI+MED and Teaching London Computing Teams. It was funded by Queen Mary University of London, CHI+MED (EPSRC EP/G059063/1), the Mayor of London and Department for Education. cs4fn is a partner on the BBC's Make it Digital campaign. The Knight's Tour puzzle was adapted from an idea by Maciej Syslo.& Anna Beata Kwiatkowska, Nicolaus Copernicus University. Linked classroom activity sheets are available from Teaching London Computing.

295_15

Queen Mary University of London

Department for Education

BBC MAKE IT DIGITAL

SUPPORTED BY MAYOR OF LONDON

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE

EPSRC
Engineering and Physical Sciences Research Council

chi+med