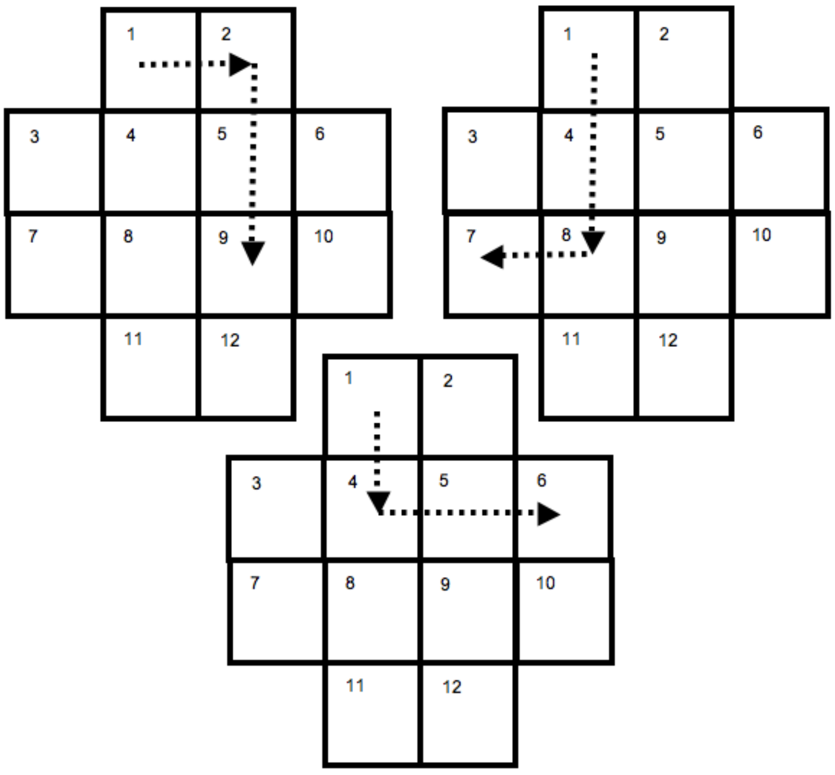
# A Brief Tour of Computational Thinking: The Knight's Tour and Other Puzzles

**Paul Curzon**
**Queen Mary University of London**

*Find a way for a Knight to visit every square on a board exactly once. 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's do no more work than absolutely necessary. Oh, and help a tourist guide at the same time.*

## The Knight's Tour

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. You must find a sequence of moves that starts from Square 1, visits every square exactly once and finishes where it started.



Version 1.0, 17 January 2014

Try and solve the puzzle, timing 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 steps of the sequence 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. Alternatively, 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 doesn’t visit any square twice and does visit them all.

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

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 just the idea that we only have a solution to a problem when we have an algorithm that can do it. If we 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 the instructions you give them will do the right thing. We will see later that problems like this involve several more computational thinking skills, but first lets try a simpler puzzle.

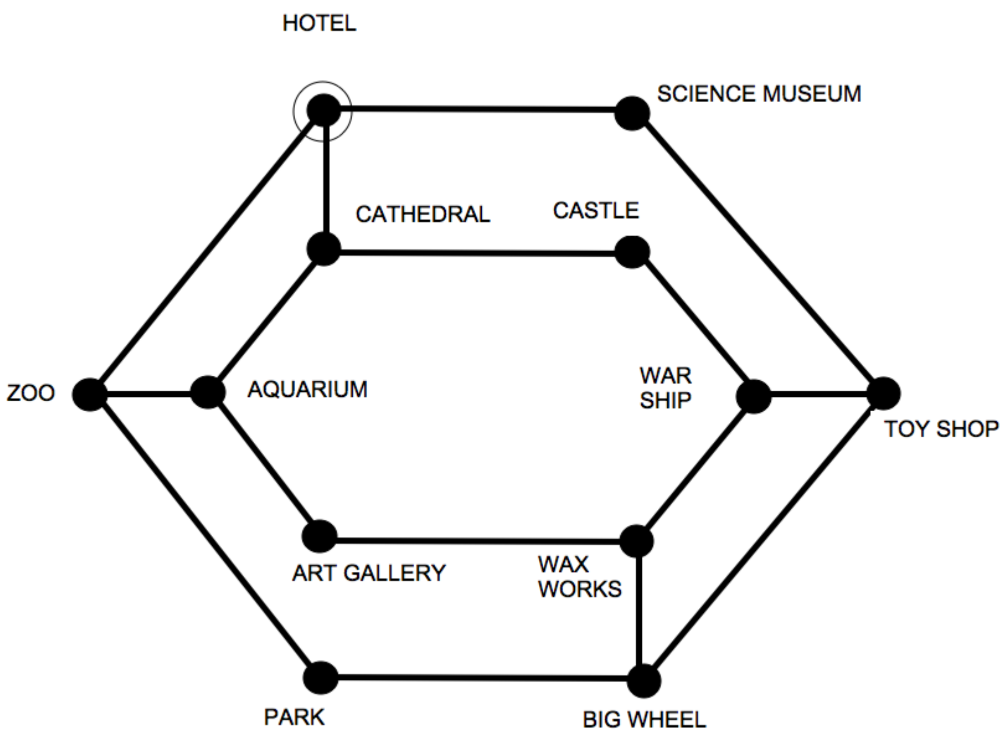
## The Tour guide

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 (see below) 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 angry if they pass through the same place twice. Obviously they also want to end up back at their hotel that evening.

Just as with the Knight’s Tour come up with an algorithmic solution and time yourself to see if it really is simpler than the Knight’s Tour.

How long did it take you? And was it easier (i.e., you solved it faster) than the Knight’s Tour?



## 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 just means 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 saves a lot of time to check it on paper first. Programmers do the same thing. They check their program works on paper (**dry run** it) but then also check it for real – that is called **testing**. Just as you did for the puzzle, programmers test their program against a list of requirements of what it must do.

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 Scientist's 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.

## Great Graphs

You probably found the Knight's Tour puzzle harder to solve, but actually it does not 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 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 of a graph **represent** something about the data that we are interested in. The edges just 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!

In our case 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, as 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.

Graphs are often used to represent information about the connections between things. You will find them on the signs at bus stops, 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 detailed map as then the information that mattered would be hard to see amongst all the detail.

Computer scientists actually have a special name for this kind of tour of a graph where you visit every node in a graph exactly once 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 traveling to every corner of a 3-dimensional shape called a dodecahedron by traveling 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 ends 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 by abstracting 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 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 spot instead, just as the tourist attractions were spots on the underground map. They are just nodes of a graph.

Secondly, which squares are actually next to one another actually doesn't 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 spots (edges and nodes). Start with square 1 – draw a spot and label it 1. Now from square 1 you can move to square 9 so draw another spot 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 spot 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 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 2 moves possible from each of the inner three squares so their spots will each have 2 lines out of them in the finished map. There are 3 moves possible from all the other squares so their spots will have 3 lines out of them.

## Go for Depth first

This way to explore all the possible moves needed to draw the graph is a variation of what is called **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, then 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

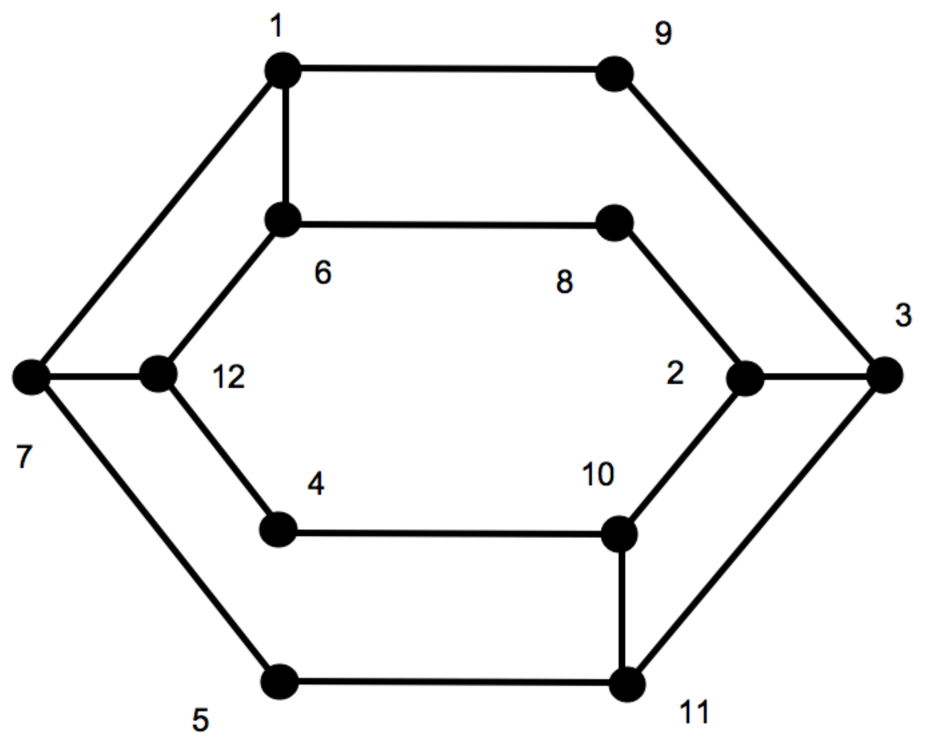
If the drawing you end up with is a bit messy you may want to redraw it neatly with no lines crossing. It can be done very neatly as two linked hexagons one inside the other (see graph below).

Once you have drawn the graph, try and solve the 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.

## 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 have a solution to the other immediately 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.



The following table 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 another. For each step in your answer to one, all you have to do is look it up and swap it for the corresponding label.

| **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 |

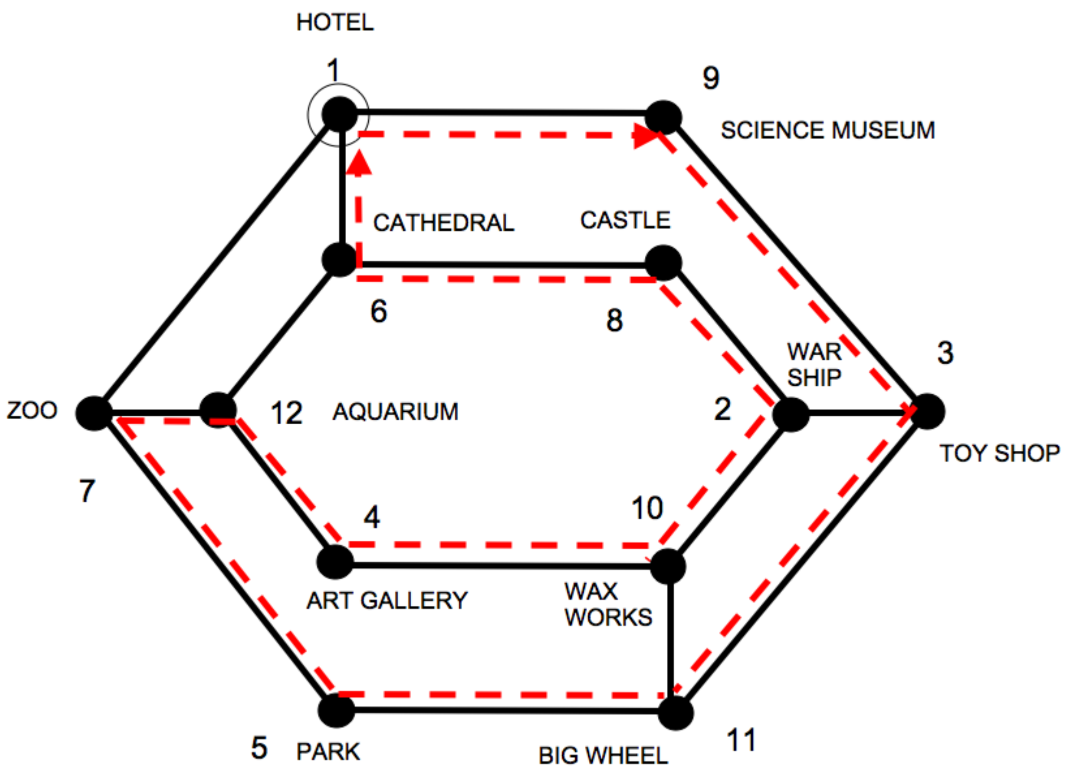
So if we came up with the following solution to the Tour Guide:

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

Using the table we immediately get a solution to the Knight's Tour.

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

You can see the mapping in the diagram below.



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 solutions (once generalized). Once you have solved one, you have solved both!

## Computational Thinking So Far

The route you came up with is a **sequence of instructions** that can be followed to visit every tourist attraction or square of the board and get 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 – so several different algorithms are solutions to the same problems. All the solutions to one are also solutions to the other.

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 (give tours over and over again with no extra problem solving work) or 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.

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. These are known as **requirements**.

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 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 clearer the things that matter in the puzzle. 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.

We also saw two examples of **generalization**. We could generalize both problem statements and see that they are really the same kind of problem of finding a series of moves that visit every point exactly once and returning to the start. Both 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 to use on graphs can be used on any of these different problems.

For our two problems, because the graphs are the same and the requirements the same, we could generalize 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.

Spotting when two problems are the same (or very similar) like this is an important part of computational thinking called **pattern matching**. It saves work, as it allows us to avoid reusing work. When you see a problem or puzzle that involves moving from place to place, whatever the places are, think about representing it as a graph. Another way of saying that is if you can **match** a problem to the **pattern** of moving from point to point then use a graph to represent it. In this case we could go a step further. Once we see that the two graphs are the same (i.e., we have pattern matched the solutions) we 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.

Queen Mary
University of London

## Use of this booklet

This booklet was created by Paul Curzon of Queen Mary University of London, cs4fn (Computer Science for Fun www.cs4fn.org) and Teaching London Computing (teachinglondoncomputing.org). The Knight's Tour puzzle was adapted from an idea by Maciej Syslo.& Anna Beata Kwiatkowska, Nicolaus Copernicus University.

See the Teaching London Computing activity sheets in the Resources for Teachers Section of our website (http://teachinglondoncomputing.org/resources/inspiring-unplugged-classroom-activities/) for linked activities based on this booklet: The Knight's Tour and The Tour Guide Activities.

Department for Education
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE

Version 1.0, 17 January 2014