15. The Path of Righteousness (Path Algorithms)

Thou shalt find it after many days
The Bible, Ecclesiastes Chapter 11, Verse 3.

As we saw previously, graphs occur in a wide range of situations. What kind of computation might we then want to do on them? One of the most common problems that arises is that of finding a path between two points. Usually we wish to do something slightly harder than this – find the shortest path.

Finding a path
The most common situation where we must find a path through a graph is when driving from one place to another. Dirk Gently, the hero of The Long Dark Tea-Time of the Soul uses a "Zen" method of finding his way to a destination when driving "*which was to find any car that looked as though it knew where it was going and follow it*"(Adams, 1989, page 29). This, however, is not much good as an algorithm since "*the results were more often surprising than successful*".

Path finding is considered such a difficult task that it is used as a test of intelligence. One of the most lasting stereotypes of a scientist is of a person in a white coat timing a rat or mouse finding its way round a maze in search of food. The aim of such tests is usually to test the rodent's memory skills, intelligence or ability to learn (though if you believe *The Hitch Hiker's Guide to the Galaxy*, it was the mice who are conducting subtle tests on humans rather than the other way round (Adams, 1979, page 119). A maze is just an undirected graph, and the problem facing the rodent is to find a path through the graph from start to finish. The same problem is used in children's quiz books, where the puzzle is to draw a line through a maze. Mazes made from hedges are often found in the gardens of stately homes such as at Hampton Court and are major tourist attractions.

Path finding is obviously fun when set as a maze, but is there any systematic way of quickly finding a path, or is the only possibility to wander randomly? Since with a maze there is no way in advance to know which paths are the best to try, we thus really have to solve the more general problem of coming up with a systematic way of visiting every node in a graph. If we do this, then we just stop when we get to the destination node (i.e. the centre of the maze). One systematic way would be to retrace your steps back to the start and begin again completely every time you get to a dead end. This would be incredibly slow however. Worse still, unless you used the Hansel and Gretal approach of leaving a trail of stones, you would probably have just set yourself a new path finding problem of finding the exit again, each time you got to a dead end.

Obviously, the first thing to do is remember junctions you have already been at – possibly by marking them in some way. Once you can recognise junctions you have previously been to, you must recognise exits you have already explored – there is no point going down a path you have already been down. Instead you choose a different path. If you have already explored all the exits at a junction, then you go back the way you came, retracing your steps until you get to the previous node, where you again either take a new exit or retrace your steps once more. This algorithm corresponds to

the well known trick for exploring a maze – put your hand on one wall and just keep going – as you go into a dead end, you automatically retrace your steps as you come back along the hedge down the other side of the path. As you return to a junction, by following the edge of the hedge you go down the next exit.

Exploring Trees

If we know something of the structure of the graph we are dealing with, we can use it to tailor the algorithm. For example, consider the problem of determining if someone is a direct ancestor of you. My father has collected a large amount of family tree details of a wide range of people in the village I grew up in, and it turns out that most of the old families are interconnected. Suppose his aim when he started was to determine if some particularly interesting person from the past history of the village is one of his ancestors. How does he answer this question? The ancestors of everyone in the village are connected in a graph. He is the start node. He must determine some way of exploring his ancestry in a way that ensures he does not miss anyone. He could therefore just follow the general maze-exploring algorithm. However, as we noted before the part of the graph he is interested in exploring – the part containing his ancestors – is actually a binary tree with him at the root. We can therefore simplify the problem to that of searching a tree. There are two main strategies that could be used called **Depth-first search** and **Breadth-first search**. In essence these are just giving different orderings of the way we back up and choose the next exit to explore in the general graph exploring algorithm.

One way he could work would be to follow one lineage back, checking his father, then his grandfather, then his great grandfather, etc. Only when getting to the furthest point possible down that lineage does he back up to the previous level to follow the other line from this point. This is called depth-first search as the depths of the tree are searched first. This algorithm has the disadvantage that if the tree is infinite you could never back up. If my father followed this algorithm strictly he would have to follow the chain all the way back to Adam or Eve (or perhaps Neanderthal Man) before starting on a second branch – not a sensible way to conduct the search. Instead depth-first searches are often cut short. In our case, if my father knows the birth date of the person of interest, he could stop the search of a branch, once arriving at a person who died before that date.

Chess players follow a similar strategy, here the tree consists of the possible positions that could result from a given move. Each move from a position gives another branch. A player explores each move by attempting to explore the tree, following sequences of moves. Better players are capable of descending further into the trees of weaker players – i.e. looking more moves ahead.

My father could alternatively work back through his family tree a generation at a time, checking that none of the current layer of the tree is the person of interest before moving to the next layer. Before moving to the great-grandparents, he would check all the grandparents. If he did this he would be using breadth-first search. This is a safe algorithm in that you are guaranteed to get the answer eventually. However, it can be slower. Whatever level the person is to be found at, using this algorithm, you will have fully checked all nodes at earlier levels of the tree first. With a depth-first search, you could hit lucky, and find the node you were searching for in the first branch you looked at. Depth-first search thus has a better best case, but worse worst case.

Finding the Shortest Path

If path finding is a good test of intelligence, then it is neither the Dolphin's nor the mice that are the most intelligent creatures on Earth as *The Hitchhiker's Guide to the Galaxy* might suggest (Adams, 1979). The most remarkable path finding creatures are Ant colonies. Once a source of food has been found they very quickly start to follow the shortest path round obstacles between the nest and the food source. Even more remarkably, their algorithm can cope with new obstacles being placed in their path. They soon find the shortest way round it, including it into their route. Their algorithm is thus adaptive.

How do they do it? In fact it is actually based on the Zen approach used by Dirk Gently, which turns out not to be so silly if you are an ant. They basically use the Hansel and Gretal method of dropping a trail – though in their case they leave a scent trail, rather than white pebbles. For finding the shortest path, they rely on the fact that subsequent ants will follow the trail left and reinforce it with more scent.

Richard Feynman, who won a Nobel prize for Physics, but who had a passion for understanding things whatever the subject, described how he worked out for himself how ants find shortest paths one year when his bath was invaded by a colony (Feynman, 1992). He put down piles of sugar for them, then once an ant had found a pile, he used a coloured pencil to careful marks its trail. As other ants found the sugar he followed their trail too but with different colours. Eventually, he had a multicoloured bath but also the answer. All the subsequent ants follow the trail of the first ant to find the sugar. It wiggles its way back to the nest and its route is not remotely the shortest path. As others travel along it however, in their rush, they often overshoot some of the wiggles, going in more of a straight line, though eventually making it back to the trail. Thus gradually the trail is straightened and also shortened.

Consider the situation where a short path has been found, but then a new obstacle blocks it. The trail disappears, so new ants arriving decide at random to go one way or the other. Roughly half can be expected to go each way. However, those choosing the shortest route will get back to the original trail more quickly. In any given period of time, more ants will get back to the trail via the shorter of the routes, thus that route will have a stronger scent. Subsequent ants coming the other way will therefore be more likely to choose it. When ants are creating the original path a similar thing happens. Once later ants have followed a slightly different path, at the branches the shorter path will gradually get more scent.

The ant's approach has been suggested as a method of finding shortest paths by computer by generating a colony of artificial ants. British Telecom has used this method to solve networking problems. However, in most human situations it is not so useful (though something very similar seems to happen every morning in Finsbury Park Station in the rush hour, as trails of people moving in opposite directions each try to optimise their route in or out of the station). It does however hint of an algorithm that we do use.

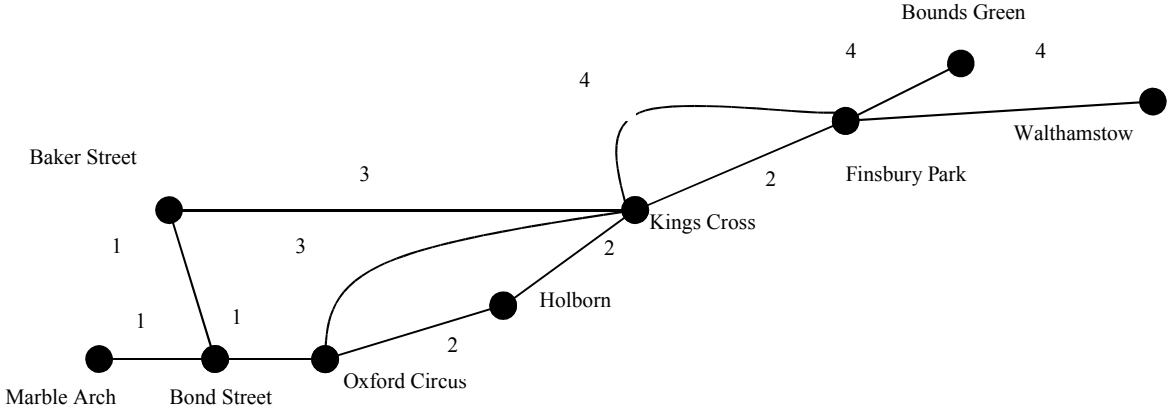
The most straightforward algorithm is to follow each path through the graph in turn from start to finish and make a note of the length of each. We keep a note of the

shortest found so far. Once all paths have been checked, the shortest found so far is *the* shortest path. Note that unlike when just trying to determine if a path exists we do apparently have to check every possible path. This is more or less what the ants do when they come to a new obstacle. They are using parallel processing however – making use of the fact that there are many hundreds of them all working on the problem at the same time.

Can we do better than this? Well we can start by using the trick from maze searching. Rather than going back to the start each time, we just back track to the previous junction. We can make a note of the shortest distance found so far to every node at the node. If we get to a node, and the sum of the distance to the previous node and of the last edge we travelled along is shorter than the current distance then we have just found a shorter path to the node we are currently at.

Combining the above idea of keeping track of the shortest route to each node so far with the idea of always choosing the nearest node to the start next, we obtain an algorithm known as Dijkstra's Algorithm after its inventor.

Suppose after a hard day at work I decide to go to see a film at the Odeon Marble Arch followed by Pizza at Ask. I must solve the problem of finding the shortest route from Bounds Green to Marble Arch using the London Underground. If we count distance as number of stations travelled, then we are dealing with a labelled graph of the following form:



In planning my route, I start at Bounds Green and put it in a list. I look at the possible legs I could initially follow and pick the shortest. There is only one route (at least on the simplified version we are considering), so I add it to a list. I now know the shortest route from Bounds Green to Finsbury Park and that it is 4 long. I now look at all possible routes from either of the stations in my list. There are three, both from Finsbury Park, two to Kings Cross and one to Walthamstow, one of length 2, the others of length 4. Since both are via Finsbury Park, they are of distance 6 and 8 from Bounds Green. I ignore the longest ones and irrespective of the destination pick the nearest one via the shortest route. I thus add Kings Cross to my list. I now know the shortest distance to it from Bounds Green – via Finsbury Park on the Victoria Line with length 6. My list of known shortest distances from Bounds Green looks as follows:

Bounds Green 0
Finsbury Park 4

Kings Cross 6

I now look for the station with the shortest route out of any of the three stations in my list from Bounds Green to it, and then on to the new station. There are still the two of distance 8 that I originally ignored, but now there are three more, two of distance 3 and one of distance 2 from Kings Cross, so of distances 8 and 9 from Bounds Green. We add the two stations of distance 8 from Bounds Green.

Bounds Green	0
Finsbury Park	4
Kings Cross	6
Walthamstow	8
Holborn	8

Walthamstow adds nothing new, but at Holborn we have a new shortest link of length 2 to Oxford Circus. However, we can get to Oxford Circus directly from Kings Cross on a link we previously ignored. Now it makes Oxford Circus the next nearest station to Bounds Green at distance 6+3, i.e. 9 stops. Baker Street is also connected to Kings Cross and of distance 9 from Bounds Green, so we add both.

Bounds Green	0
Finsbury Park	4
Kings Cross	6
Walthamstow	8
Holborn	8
Oxford Circus	9
Baker Street	9

Bond Street can be added next and then Marble Arch at distances 10 and 11 from Bounds Green. Our final list of distances to stations is thus:

Bounds Green	0
Finsbury Park	4
Kings Cross	6
Walthamstow	8
Holborn	8
Oxford Circus	9
Baker Street	9
Bond Street	10
Marble Arch	11

We have thus processed the stations in order of shortest distance from our source station. We have then used the fact that we already know the distances to the nearest ones to work out the next nearest. If we already have calculated a distance to a station, we ignore any other links to it. In the above we just recorded the distances. We could similarly note the actual routes to each station, using the routes already in the list to give us the shortest route to the next one added.

When we are doing route planning on a map, we do not look at all possible routes. We quickly discard many alternatives because they are going in the wrong direction, or are too long. Similarly the ants quickly discard routes that are going in completely the wrong direction. At each junction the shortest path over the next leg is taken.