Alternative Search Methods

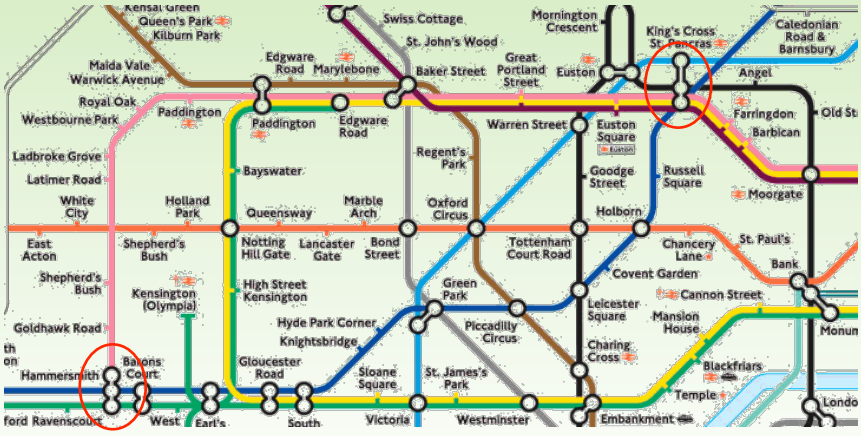
Introduction

- We have looked at greedy search as a method of finding solutions to a problem.
- Greedy search only works where local improvements to a solution lead us to the best overall solution.
  - This is often called 'hill climbing'
- We will now consider alternative methods that allow us to find solutions where we may find worse solutions before we find the best (or at least a better one).

Route Finding

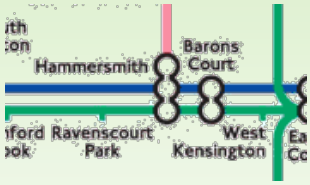
- Let's try and get around on the London Underground
- Our simple model tells us:
  - Which lines leave a given station
  - The next station on any given line in a given direction
- This is what a human would know without a map

From Hammersmith to King's Cross



From Hammersmith to King's Cross

- We start in Hammersmith with a choice of three lines:
  - Piccadilly
  - District
  - City



First Step

- We don't know where the lines go, our system just tells us which lines we can choose (like a human with no map)
  - So we choose the District line, pick a direction (East) and ask for the next station
  - We are told it is Barons Court
  - We check to see if we are at the goal (King's Cross). We are not. Now what?

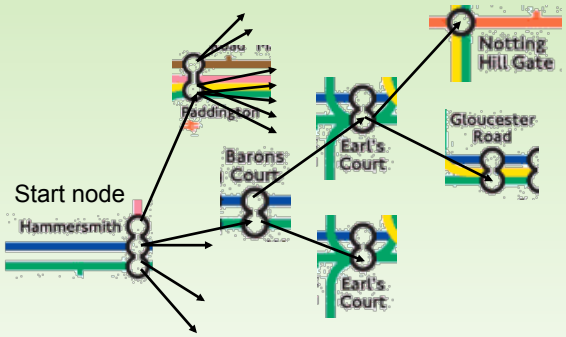
Second Step

- We could repeat the process above, picking lines at random where there is a choice, and travelling through stations where no change is possible
- In the end, we would probably find King's Cross, but it could take years.
- We would also waste a lot of time in stations we had been to in the past
- This is a random search and is useless

Search Trees

- Let's try to be a bit more structured about things
  - Let's also assume that we can give up at any time and return to Hammersmith (which we can, because we are running the model in our head, not the real tube)
- Search trees represent decisions at each step of a multi-step search problem

The London Underground Tree



Well, a bit of it anyway

The tree's nodes represent decisions, so to cover the whole space in a structured way, you need to traverse every node of the tree until you find the goal.

Tree Searching Strategies

- We will look at four uninformed, tree searching strategies
- That is, there is no feedback (or it is not used) relating to how close to the goal you are, other than Yes or No.
  - Breadth first search
  - Depth first search
  - Depth limited search
  - Iterative deepening search

Where is the Tree?

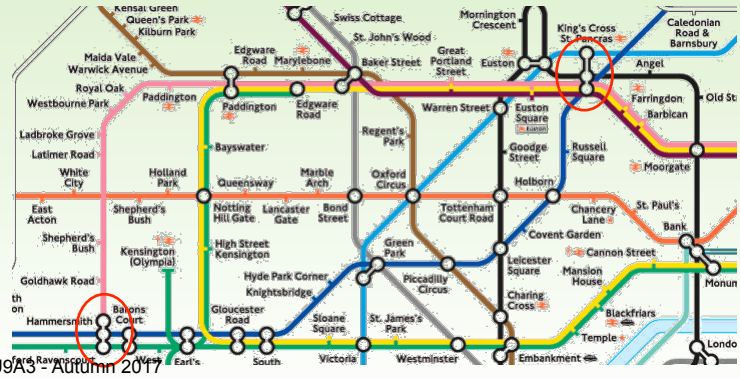
- Although it is possible to represent an environment as an explicit search tree, in reality the tree would be too big to store, as it would need to contain every possible combination of situations and actions in the environment
- So think of the tree as virtual – it is a tree of possibilities, but only those that you try are actually realised
- Asking for all the possible actions at a given node is known as expanding that node

Breadth First Search

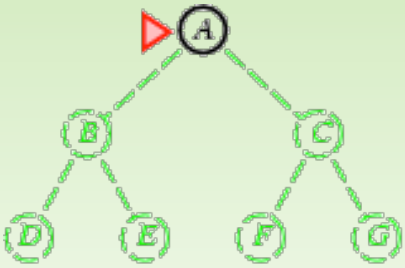
- Search each option in turn at every decision point
- If the goal is not found, return to the first node at the next level and repeat the process
- Involves 'jumping' across to nodes at the same level in the tree rather than always following branches

Breadth First From Hammersmith

- From Hammersmith, try every line in turn, but only go until we hit a new choice
- If they all fail, then its back to Paddington to try the next four from there



Breadth First Search



The order of the breadth first search of the tree above is

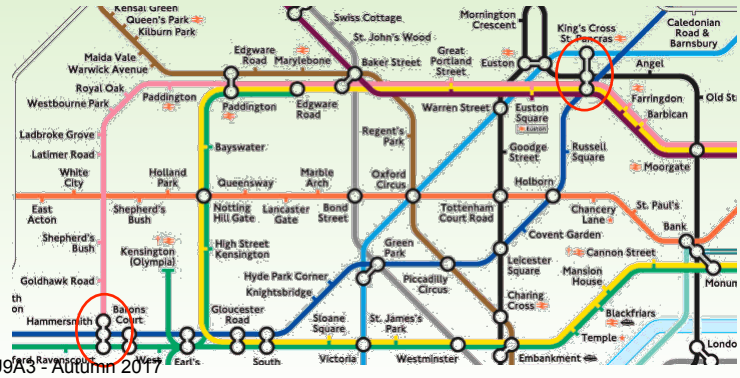
A B C D E F G

Depth First Search

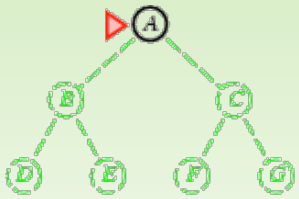
- Depth first search follows a single route to the end of the tree, taking the first unexplored branch on the left each time
- If the goal is not there, back up one step and try the next branch
- A more ‘natural’ traversal as it follows branches to their conclusion
- No new branch at a given level is tried until all options below the current branch have been explored

Depth First From Hammersmith

- From Hammersmith, City line first, first branch is at Paddington, then choose Bakerloo to Baker Street, then Jubilee to its end – fail
- Go back one step to Baker Street, try City – find King’s Cross!



Depth First Search



The order of the breadth first search of the tree above is

A B D E C F G

Depth Limited Search

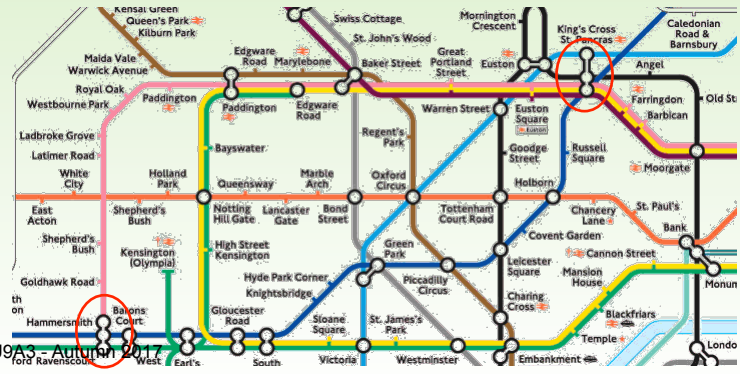
- Remember that the tree is virtual, not an actual representation of all possible decisions
- So a route can take us in a loop back to a state (or station in this example) we were in before, but we wouldn't know – the same state can appear many times in a tree
- Depth first search could loop round forever as there would never be a leaf node to end on
- Also, if a particular route is taking longer than we think the true path should take, it makes sense to abandon it unfinished and back-track
- Depth Limited search does this

Depth Limited Search

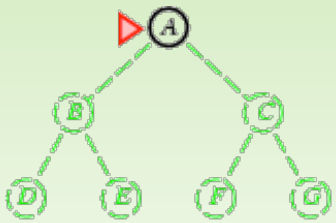
- Simply carry out a depth first search but terminate any search path after a given number of steps
- Removes risk of looping forever
- Allows searches that are taking too long to be terminated
- 'Too long' is defined as *d*, the stopping depth

Limited Depth First

- From Hammersmith to Paddington to Baker Street, to Oxford Circus, to Bond Street to Baker Street
- Now, this is not the same Baker street as before – logically it is further down the tree so we repeat the loop forever, unless stopped by the depth first criteria.



Limited Depth First Search



The order of the breadth first search of the tree above is

A B C when *d* is set to 1 (follow 1 branch)

Iterative Deepening Search

- What if we set our limited depth search to 10, but the solution is at depth 12?
- Iterative deepening search allows a tree to be explored to a sensible depth in its entirety
- If that search fails, a new, greater depth is tried
- With no knowledge about the problem, you should use *d*=1, *d*=2, *d*=3 and so on
- With some knowledge about where the solution might be, *d* could be set with more sense

That Looks Familiar

- Isn't that just a less efficient version of breadth first? The order of new nodes visited is the same
- Well, yes and no. IDS turns out to be far MORE efficient than BFS.
- The next slides compare the different techniques and you'll see why…

Comparing Techniques

- Let's assume that at every decision point, you have *b* choices and there are *d* levels
- That is *b* choices at the root node
- b^2 choices at level 1
- b^3 choices at level 2
- b^{d+1} choices at level *d*
- To cover the whole tree takes $O(b^{d+1})$ time
- Is that bad? Well, a depth of 12 with 10 decision choices would take over 30 years!

Time For Each Technique

- Breadth first takes O(b^{d+1}) where d is the level of the shallowest solution
- Depth first takes O(b^{m}) where m is the maximum depth of the tree
- Limited depth first takes O(b^{l}) where l is the limit of the search depth. l is less than m, but the search might fail
- Iterative deepening takes O(b^{d}) time – it is actually quicker than breadth first.

It Gets Worse

- The tree search algorithms require you to remember where you have been
- That requires a stack where visited nodes are placed and removed in order to backtrack to complete traversals
- Not all search techniques use the same amount of memory

Memory Usage

- Breadth first uses O(b^{d+1}) memory, which, as we have seen, is an awful lot
- Depth first uses O(bm) memory, which is much better
- Depth limited search uses O(bl) memory, where l is the depth limit – also very good
- Iterative deepening search uses O(bd) memory – so it is MUCH more efficient than breadth first