Student ID: 21427139  Module code: 6G7V0011\_2122\_9F

Artificial intelligence princeples

Contents

[Problem formulation 1](#_Toc86326211)

[Environment and algorithm design 4](#_Toc86326212)

[Results 5](#_Toc86326213)

[Evaluation and comparison 6](#_Toc86326214)

[Discussion 7](#_Toc86326215)

# Problem formulation

a). Introduction to the problem

The problem presented in our assignment brief states that we need to represent a

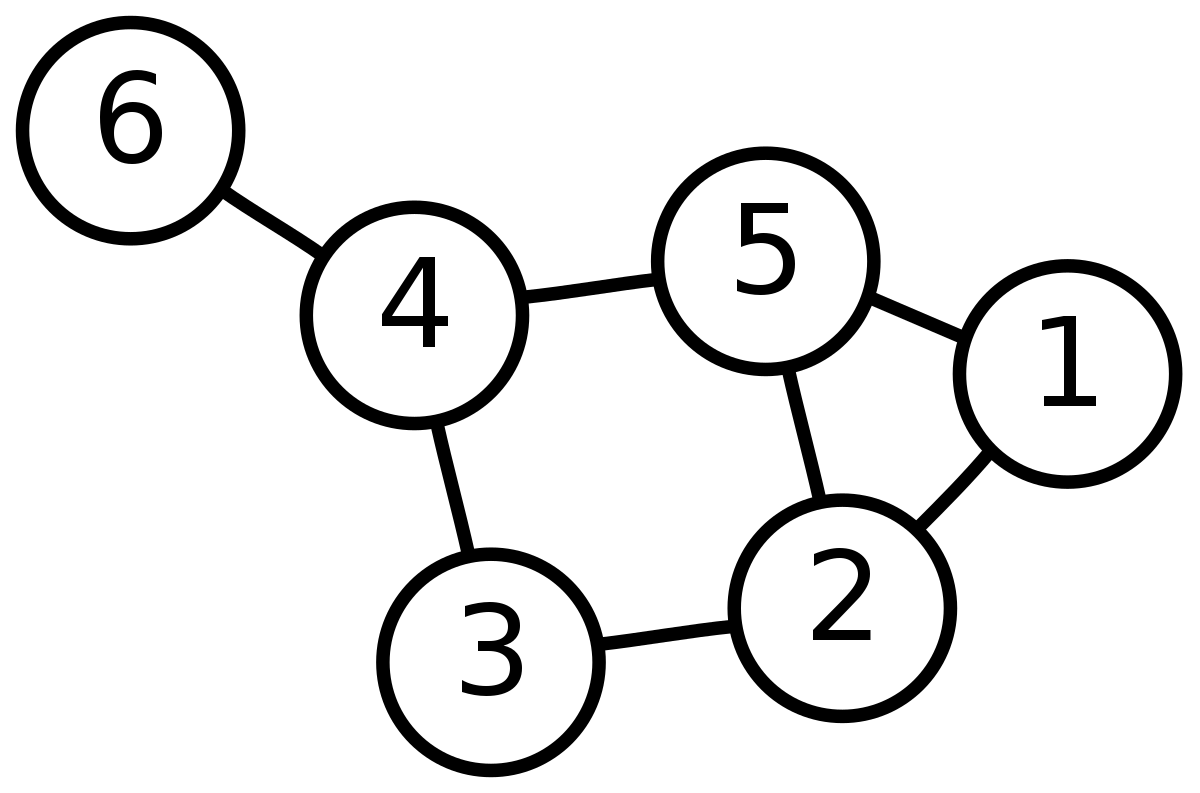
mobile robot of sorts, such as room cleaning robot alike to the Roomba and represent its movements from position ‘a’ to position ‘b’ and return the given route.   
The definition given to us for a mobile robot is as follows:

‘Is a robot that is capable of moving in the surrounding’ the real-life application of this would be useful as we are seeing many more automated machines in our daily lives such as the above-mentioned Roomba. Other examples could include Amazon delivery drones and even potentially Unmanned ariel vehicles used in military applications such as the Reaper drone.

We are also tasked to do this with multiple different search algorithms. A search algorithm is a simple method of determining the path from one specified point to another. There are many ways that we can code this into our problem but per the brief we have been asked to use four methods: The Breadth-first search, the depth-first search, the uniform-cost search and the A\* search methods.   
  
Each method works in its own unique way which will be expanded upon in the below section.

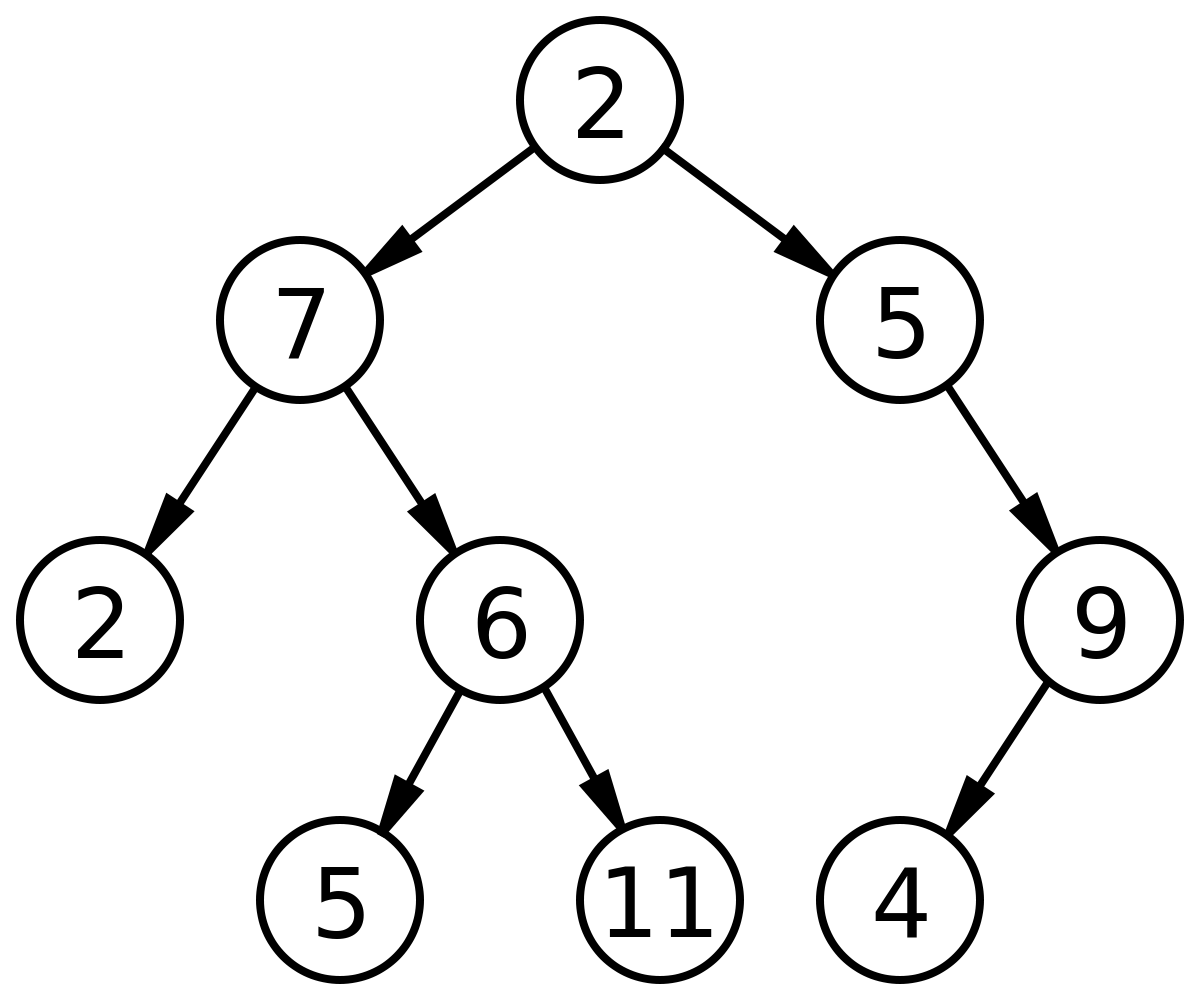
b). Introduction to possible methods

it would be crucial to mention that the given methods are to be coded in a way that represents them working on a array as if it was a graph data structure and a tree data structure. The difference between a graph and a tree is quite simple.



*(Figure 1: a graph representation)*

As seen in *Figure 1* a graph is represented by a series of nodes, sometimes referred to as vertices and edges which link each node. The most defining feature of a graph is that there can be more than one edge between each node.



*(Figure 2: a binary tree representation)*

On the other side of things, we have the tree structure which is similar to a graph in the sense that it too has nodes and edges to represent each item and the path between them. However, the differences are that unlike a graph a tree can only explicitly have one path to each node. This is where we expect to hit issues in the implementation of the code.

Another key difference on the code side of the coin is that a graph search will register visited nodes to avoid loops and possible infinite looping. Whereas a tree does not register visited nodes and therefore almost always devolve into a infinite loop which can crash the program.

to start with we will break down the basics of the breadth-first search(BFS).

A BFS works by moving through a graph or tree in layers, if we were to take *figure 1* and make our starting point node 1 to get to node 3 the program would start by registering the starting node(1) and adding it to a visited array. It would then look at the next two nodes linked for us, this would be 5 and 2. These would be added to a queue array and the program would then move on. It would move to the next point in our queue, in this case that would be 5. It would then register its neighbour nodes: 2, 4. After checking the next item in the queue, 2. It would then move onto 4 as it will not register the same node twice. From 4 it will see 3 and 6 finally reaching our goal and returning the path that it took which would be as follows:

1 -> 5 -> 2 -> 4 -> 6

A Depth first search(DFS) works in a different way to BFS. Instead of going down the child nodes in layers the search will go for the left most child and progress to the next left most child from there until it reaches the end of that branch. Whilst doing this it registers all parent nodes and deletes the final child node from the array it is tracking. After reaching the final child of one branch it will move back to the parent and check the next left most node and keep doing this till it find the goal state or has completed the array. It is easiest to show this style of search on *figure 2*:  
say we start at point 2 and our goal is node 5 the program will go to node 7, then 2 and then back to 7, moving on next to 6 and finally onto the goal of 5 and will output the following path:

2 -> 7 -> 2 -> 7 -> 5 -> 6

Uniform cost search(UCS) is a different kind of search that the ones above. Instead of following a set way of searching this search relies on the cost of traversing to each node to ultimately make the cheapest path, in a way this is almost identical to Dijkstra’s shortest path algorithm. To implement this, you would need to have a way to assigning weight to each edge in the graph and the tree. Ultimately recording the final cost of your route from start to goal and relaying this back to the user.

The A\* search method is in a way like UCS in that it tries to achieve the cheapest route to a path by a measuring statistic, in this case it is the heuristic of a path as well as the normal cost. The heuristic cost is the direct route from one node to the final node. If you were looking for a global map, say of the United Kingdom this would be the term known ‘as the crow flies) which would be the straight-line distance between two points. You cannot do this as you cannot fly but using this data along with the cost between two nodes on the ground means you can complete a journey in a relatively cheap cost or quick time. This search method is widely regarded as one of the best for real-world problems.

# Environment and algorithm design

a). Robot and Environment

designing the environment and robot on paper was simple. I decided that my robot for the sake of simplicity will be the size of one grid in an array, 1,1. This is because the problem is more focused on the ability to implement the searches rather than anything else, this means that I will have less problems in implementation.   
  
the environment was an issue. I created multiple instances before the final code. These ranged from a normal two-dimensional array to represent a grid with x and y coordinates, a graph to represent the north of England which used the code given in the ‘search\_BFS.ipynb’ file as well as modules from the ‘aima-python’ folder. I opted to not follow with this as all it was a graph and will allow me to do both graph and tree searches. It meant that I would not be able to faithfully represent a room problem as stated by the assignment brief. The other option was to use a binary matrix, my main issue for this was trying to map this over to a array with coordinates.   
  
Using the python package of Numpy I created a 5x5 binary matrix. To do this I started by initialising the array with the “zeros” function which populates the array with the value of zero. Then using the random module, I assigned ones into the array to represent obstacles that cannot be used as a movement square.

b). Algorithms   
the algorithm design was easy on paper but provided to be a little more difficult than expected in implementation.   
For the first two, BFFS and DFS I opted to start with the graph problem because this version of the search methods registers the visited nodes meaning it won’t devolve into an endless loop causing the interrupter program to crash. The key variables used to complete these are the fore mentioned binary array. A Boolean array called visited or visit which tracks where the robot has been and a queue array to register which nodes to visit next. Both algorithms make use of recursion for the searching as in my eye’s recursion is a key concept of artificial intelligence and it also reflects ability of the program.   
  
The UCS method uses the previously mentioned arrays along with an array of random integers called cost. This is so it can evaluate the cost of the next movement allowing the method to find the cheapest route to the objective.   
  
I did not manage to complete the A\* method as I lost time in the early weeks figuring out how to start the methods and by the time, I finished the UCS which is the basis of the A\* search there was only one day left. However, for the purpose to express my knowledge, my plan was to introduce a heuristic variable that would measure and track the heuristic distance from the robot’s current position and the goal state. Using an if loop to assess which of the next nodes reduces the heuristic as well as having a cheaper cost from the cost array.

# Results

Using the time and guppy modules I can record how long the program takes before running the tree style code to execute and how much data it consumes while in use. I have not but it after because this code does not stop. The below, table1 is the results produced from the modules.

Total time taken = 12.006507635116577

Partition of a set of 375248 objects. Total size = 48154386 bytes.

Index Count % Size % Cumulative % Referrers by Kind (class / dict of class)

0 102059 27 8015624 17 8015624 17 types.CodeType

1 48396 13 6401670 13 14417294 30 function

2 53901 14 4002527 8 18419821 38 tuple

3 8265 2 3525872 7 21945693 46 function, tuple

4 25939 7 3275657 7 25221350 52 dict of type

5 28979 8 3146056 7 28367406 59 dict (no owner)

6 11246 3 2394251 5 30761657 64 dict of module

7 10856 3 2270608 5 33032265 69 type

8 19885 5 1701204 4 34733469 72 list

9 787 0 1198680 2 35932149 75 function, module

*Table1*

Graphical user interface, text

Description automatically generatedGraphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

Above are some visualisations of the robot moving, the dark squares represent obstacles, and the white blocks represent the slots that the robot has been too. There could be better ways to do this. I used the basic matplotlib function of implot. The reason there are no obvious ticks is because they did not match correctly with the squares. Research suggests this a bug with the module, so I decided to leave them out as it is obvious enough without them.

# Evaluation and comparison

a).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Time(s) | Path | Cost | Storage |
| G-BFS | 4~ | The path changes per program running as the obstacles are not static | - | 48154386 bytes |
| G-DFS | 4~ | The path changes per program running as the obstacles are not static | - | 48154386 bytes |
| G-UCS | 4~ | The path changes per program running as the obstacles are not static | The path changes per program running as the cost are not static | 48154386 bytes |
| G-A\* | - | - | - | - |
| T-BFS | ∞ | ∞ | - | 48154386 bytes |
| T-DFS | ∞ | ∞ | - | 48154386 bytes |
| T-UCS | ∞ | ∞ | ∞ | 48154386 bytes |
| T-A\* | - | - | - | - |

*\*These figures are taken from the report printed table1*

Through research, lectures and this assignment we are asked to answer which method we think works best for this current situation. A note is that this analysis won’t include A\* as it was not completed.   
  
Although the algorithms have their benefits, UCS seems to be the one that works the best for this situation. Even when the obstacles aren’t static and the path costs change it always results in the cheapest and quickest execution, which is evident when you run the code multiple times. Therefore, I believe it would be the best method to follow if this code was to be scaled up into a real application.   
I believe A\* would be able to throw up some good counter arguments to this view but my bad time management lead me to miss out on this area of the assignment.

If I had more time, another area I would work on would be the visualisation. It could look better and be labelled better. As well as this it would be a could idea to print it out once and load the new image each time, making it looking like an animation.   
There is one error of note, which I believe is a python bug. Sometimes on running the DFS method will not correctly run and crash, if you restart and run all in jupyter this solves itself and the method will run correctly, I cannot seem to explain why this happens.

# Discussion

Finally. The methods learnt are all useful in their own respects, I would avoid using a tree version of any of them as it seems to almost always decline to an infinite loop, this I believe is because the code doesn’t track where it has been. The fact it doesn’t track also lead it to become more costly as it will repeat positions and cause the whole method to take more time to complete its task making it inefficient.

# References:

Figure 1:

Google Images(2021), “Graph Data Structure”, <https://www.google.com/search?q=graph+structure&client=firefox-b-d&source=lnms&tbm=isch&sa=X&ved=2ahUKEwj23vCeldTzAhWTT8AKHbj4D10Q_AUoAXoECAEQAw&biw=1440&bih=790&dpr=2#imgrc=-fcFTwoO-r4NCM>

Figure 2:

Google Images (2021), “Tree Data Structure”, https://www.google.com/search?q=tree+structure+no+backgroujnd&tbm=isch&ved=2ahUKEwjj\_NvXldTzAhWawYUKHSy5AscQ2-cCegQIABAA&oq=tree+structure+no+backgroujnd&gs\_lcp=CgNpbWcQAzoFCAAQgAQ6BggAEAgQHjoECAAQGFCpnwlYmsEJYKnDCWgAcAB4AIABSIgBtweSAQIxNpgBAKABAaoBC2d3cy13aXotaW1nwAEB&sclient=img&ei=foRtYePFLZqDlwSs8oq4DA&bih=790&biw=1440&client=firefox-b-d#imgrc=mi8mHl0UHqH-hM