DAT405 Assignment 8 – Group 52

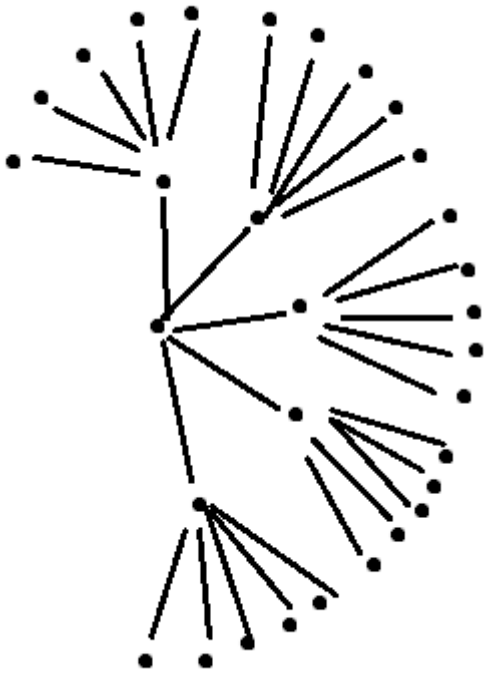
Hampus Jansson – (6 hrs)

Erik Johannesen – (6 hrs)

June 5, 2023

1.

a) d is the maximum of children a node can have, and the length between the goal and the initial state is r. Since the question is about finding the maximum number of iterations for BFS, the graph's nodes should all have d children. Here is an example where d=5 and r=2.

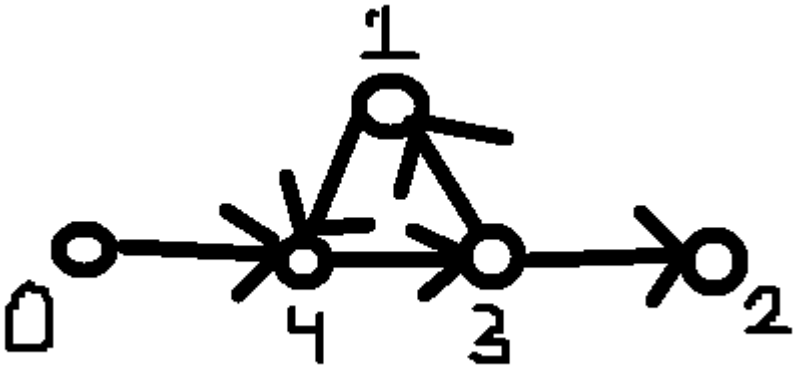


The breath first algorithm will search the breadth of the graph first, so it will look at all the inner nodes before reaching the outer layer. In this example this is 5. The algorithm will then start searching the outer nodes, and in a worst-case scenario the goal node will be the last one it searches. This means it will also search all the outer nodes, in this case 25. So the amount of iterations in this case would be 5+25=5^1+5^2. This pattern continues no matter what values r and d have. So the mathematical formula is;
max iterations = d^1+d^2+...+d^(r-1)+d^r
If the tree had more nodes after the level r these would never be looked at before reaching the goal, hence these are not included in the equation.

b) For this question we consider the same graph. If the goal node is the last one to be searched of the outer nodes, then each path will be searched in this case. The amount of paths in the worst case for every level is d^r and the length of the paths for

each level is r. For this example the memory taken would be: 1*5^(1)+2*(5)^2. This pattern continues no matter what values r and d have. Each node takes up k memory, so the mathematical formula is:
max memory = 1*d^(1)+2*d^(2)+...+(r-1)*d^(r-1)+r*d^r

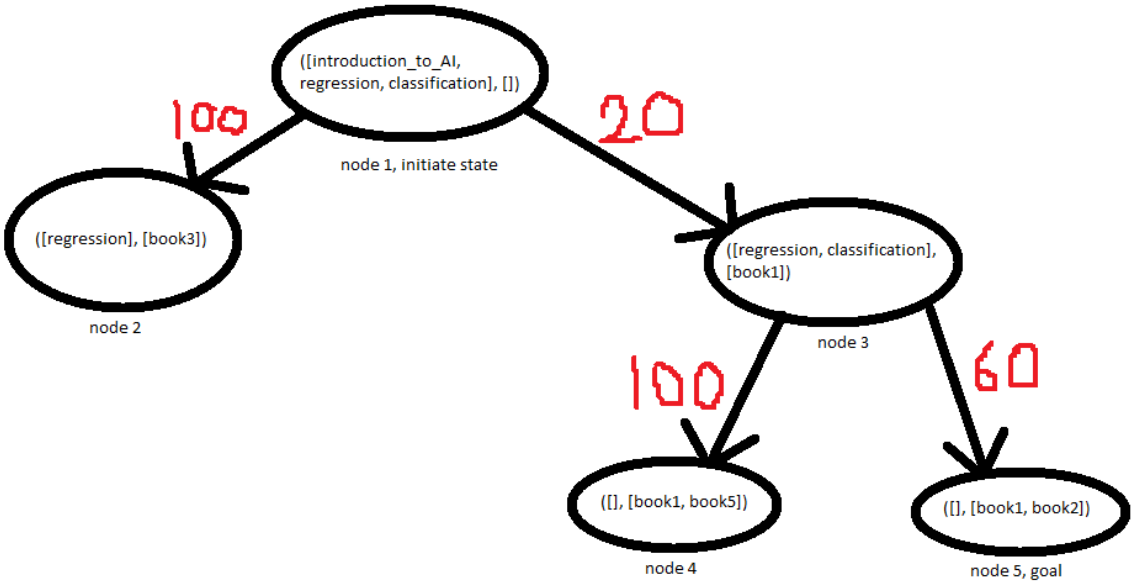
2. If the top node is 1 the depth-first algorithm will never find a solution, no matter what values the middle two has. Here's an example:



When reaching node 3, the algorithm will prioritize node 1 over node 2. After this it will prioritize node 4 over node 2 because this node is considered to be at a "deeper level". Same applies for the algorithm´s next step when choosing node 3. At this point node 2 and node 1 is added again to the list but the algorithm will once again choose node 1 and the cycle will continue. The answer is therefore; the top node is always 1, the two middle nodes can each be 3 or 4, it does not affect the outcome. There are therefore 2 possible combinations.

The depth-first could be modified in a way where the algorithm recognizes nodes it has already "visited", and then removes them from the list. This way the cyclic behavior is avoided.

3. a)



The goal node is node 5. The frontier: [node 2, node 4].

b) When calculating a heuristic function h(p) for a path p, the only relevant part is its end node. A possible way to calculate heuristic value for a path could be to consider how many topics the node has left. The fewer topics it has less the higher priority it should get (and value). An example of a heuristic function that achieves this is h(p) = 1/(len(n.topics)). Here n = end node of the path. The lower n.topics is the higher does h(p) become. This is a pretty good choice of a heuristic function because the less topics that are left to be covered, the closer the customized book is to the desired result.

4. a)
A nodes cost is displayed by node(x, y) - x = order in the path, y = Manhattan distance to g

Iteration 1, current node S:
Three nodes are added to the list:
44(1+3), 53(1+5) and 42(1+5)
List: [44(1+3), 53(1+5), 42(1+5)]
Node 44 are removed from the list and added to the corresponding path:
path: [44]

Iteration 2, current node 44:
Three nodes are added to the list:
34(2+2), 54(2+4) and s(2+4)
List: [34(2+2), 53(1+5), 42(1+5), 54(2+4), s(2+4)]
Node 34 are removed from the list and added to the corresponding path:
path: [44, 34]

Iteration 3, current node 34:
One node is added to the list:
44(3+3)

List: [53(1+5), 42(1+5), 54(2+4), s(2+4), 44(3+3)]
Node 53 are removed from the list and added to the corresponding path:
path: [53]

Iteration 4, current node 53:
Four nodes are added to the list:
54(2+2), 63(2+6), 52(2+6) and s(2+4)
List: [54(2+2), 42(1+5), 54(2+4), s(2+4), 44(3+3), s(2+4), 63(2+6), 52(2+6)]
Node 54 are removed from the list and added to the corresponding path:
path: [53, 54]

Iteration 5, current node 54:
Three nodes are added to the list:
44(3+3), 64(3+5) and 53(3+5)
List: [42(1+5), 54(2+4), s(2+4), 44(3+3), s(2+4), 44(3+3), 63(2+6), 52(2+6), 64(3+5), 53(3+5)]
Node 42 are removed from the list and added to the corresponding path:
path: [42]

b)

A* required 85 operations to find the solution, BFS required 399 and Best-first search required 47. They all found the shortest path which had the length 10.

A* starts making a broad frontier when it gets a larger f(p) in every direction because it has to move away from the goal. Then when it reaches 14, it can add a new node to the frontier with an f(p) equal to that of 14 when it continues through that point but expanding the frontier in any other direction would get a larger f(p).

BFS starts with finding every path with a length of 1, expands the frontier to every path with the length 2 and so on until it reaches paths with a length of 10, at which point it finds the goal.

Best first starts by going up to the node 34, from there it starts adding nodes that follow the path of the wall in both directions until it reaches 22. From there, it can add nodes with a shorter h(p) all the way to the goal.

c)

These images show an example where the greedy best-first algorithm fails to find the shortest path.

When best-first reaches the node to the left of the start, the shortest h(p) is found along the suboptimal path that goes left of the wall. The shortest path that goes right of the wall gets a longer h(p) in parts of the path which tricks the algorithm away from there.

BFS finds a path with the length 8, while best-first finds a path with the length 10.