Assignment 8 - Searched based AI

Albin Ekström: 9h
Pauline Nässlander: 9h

March 2022

Q1: A directed graph

a) In worst case, all nodes have n children. Hence, searching through each level will take n^l, where l is the current level. Starting at $l = 0$ the goal will be at $l = r$. Number of iterations will be:

$$\sum_{l=0}^{r} n^l = \frac{n^{r+1} - 1}{n - 1}$$

b) Every node takes one unit of memory n^l. In addition, for every node a path is stored which takes $l \cdot n^l$ units of memory for the level l. Starting at $l = 0$ the goal will be at $l = r$. Number of iterations will be:

$$\sum_{l=0}^{r} n^l + l \cdot n^l = \sum_{l=0}^{r} (1 + l)n^l = \frac{(r+1)n^{r+2} - (r+2)n^{r+1} + 1}{(n-1)^2}$$

Q2: Depth first search

Table 1 describes the current states and the possible paths to next states. There's only one state which has two possible paths: **b**, see figure 1. Hence, it doesn't matter which number is placed on **a** and **b** since the only tie will be between **c** and 2. DFS will choose the path with the smallest label. The goal is to create a loop, hence if **c** has the label 1 this will be chosen resulting in a never ending loop; from **a**, to **b** to **c**. To avoid this the label 1 would have to be placed on either **a** or **b**.

Current State	Next State
0	a
a	b
b	c, 2
c	a

Table 1: Path from current to next state in Figure 1.

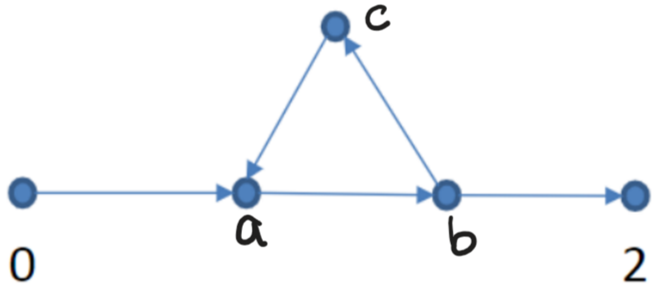


Figure 1: Directed Graph

Q3: Graph searching

a) The tree will expand from the lowest cost path to the next node. If any of the paths finds the goal this path is chosen, even if the other path has a lower cost. Figure 2 shows how the algorithm will find the goal. The total cost will be 60, the goal will be $<$ [], [seg0, seg4] $>$ and the frontier when the goal is found will be $<$ [skiing, [seg2] $>$ & $<$ [robots, [seg0, seg1] $>$.

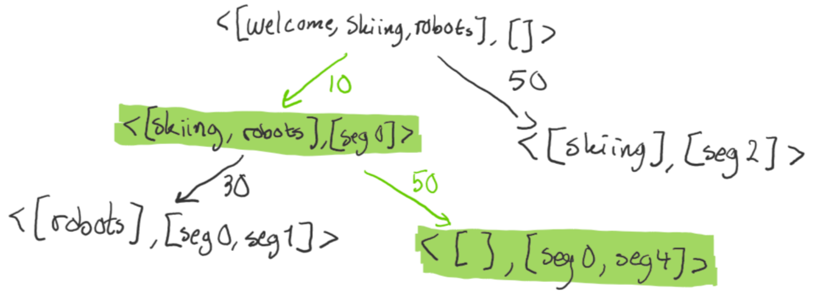

<[welcome, skiing, robots], []>
10
50
<[skiing, robots], [seg 0]>
<[skiing], [seg 2]>
30
50
<[robots], [seg 0, seg 1]>
<[ ], [seg 0, seg 4]>

Figure 2: Search space as a tree

b) Assuming that S equals the set of all segments and S(p) equals the segments used in path to the node. In addition let T equal the set of all topics and T(p) equal the topics left to cover at node in end of path p. Then S\S(p) is the set of segments that are not yet used. Define a function min{S} as the length of the shortest segment, then a heuristic admissible function can be defined as:

$$h(p) = \begin{cases} \text{min } \{\text{S } \backslash \text{S(p)}\}, \text{ if T(p) } \neq \textit{null} \\ 0, \text{ else} \end{cases}$$

This non trivial heuristic function is admissible since the minimum segment can never be longer than the actual segments left to the goal state, securing the condition $h(p) \leq cost(p')$ for all paths p, where p' is the path from the end of p to the goal.

Q4: Graph searching

a) Defining the coordinates according to figure 3 the start position has coordinates, $s = (3, 2)$ and the goal position, $g = (2, 5)$. The estimated cost $f(n)$ from the start node to the goal node, depended on the current node, n is calculated with $f(n) = g(n) + h(n)$. Where $g(n)$ is the cost from start to n and $h(n)$ is the heuristic function. The manhattan distance is used as the heuristic function which can be defined as $g(n) = |x_n - x_g| + |y_n - y_g|$. The absolute difference between n and g position.

If a direction is not valid i.e. it will lead to a black box on the grid, it will not be added as a next possible step. Also notice that in the fourth step the path does not continue from state (2,3) and go east since the state (3,3) is already visited. In addition when the algorithm chooses between the paths with weight 6 there is a tie and because of the lexicographic order tie breaker the algorithm starts with taking east from the start node.

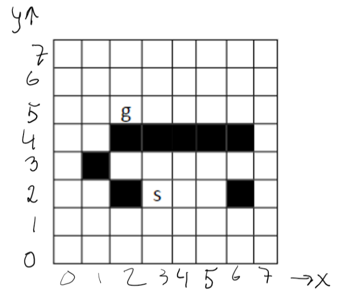
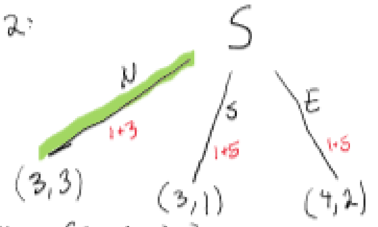


Figure 3: Coordinates on the grid

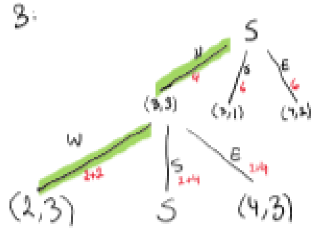
iteration 1: path stored : {S}

iteration 2:



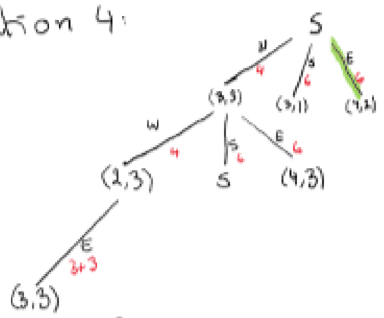
path: {S, (3,3)}

iteration 3:



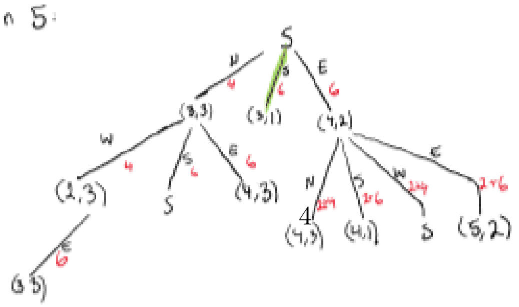
Path: {S, (3,3), (2,3)}

iteration 4:



path: {S, (4,2)}

iteration 5:



Path: {S, (3,1)}

b) Firstly, we let the program solve the problem using A* algorithm this produces the shortest path in 71 operations. This algorithm uses an estimated cost from the start node to the goal node using the equation $f(n) = g(n) + h(n)$ to calculate this estimated cost. Since the heuristic function is the Manhattan distance to the goal node the algorithm will start by moving straight towards the goal node not knowing that there will be obstacles there. After exploring the path blocked by black boxes the algorithm will search further away from the goal always choosing the cheapest path not yet explored. This is a very fast and effective algorithm.

Secondly, the BFS algorithm used 364 operations to solve the problem. This algorithm does not have any perception of where the goal is and does not take any weights into consideration. The algorithm is still sure to always find the shortest path if there is one but since it does not have any weights or heuristic function to nudge it towards the goal it will search in circles from the start node until it happens to find the goal which takes more time and operations.

Lastly, the Best-first search algorithm solved the problem using 48 operations.The Best-first search algorithm will expand on the most promising node according to some specified rule (the heuristic function). This is very similar to the A* algorithm but in difference to A* the Best first search does not take the cost from the starting point to that node into consideration. That is if $g(n) = 0$ in the A* estimated cost function we get best first search cost function. This algorithm is faster at solving the problem than A* since it won't search downward before going left past the obstacles like the A* does as these nodes are closer to the starting node.