

DAT405 Assignment 8 – Group 95

Martin Blom - (2 hrs)

Jakob Windt - (4 hrs)

March 25, 2023

Question 1

0.1 A

To find the maximum number of BFS iterations, we need to assume the worst scenario, which in this case equates to 'd' children per node. Furthermore, 'r' symbolizes the shortest path between a start and end node in the graph, which means there could be a maximum of 'r' BFS layers in the graph. As a result of the BFS algorithm, the solution will grow exponentially with each layer, since BFS searches layer by layers. With this, we arrive with our equation.

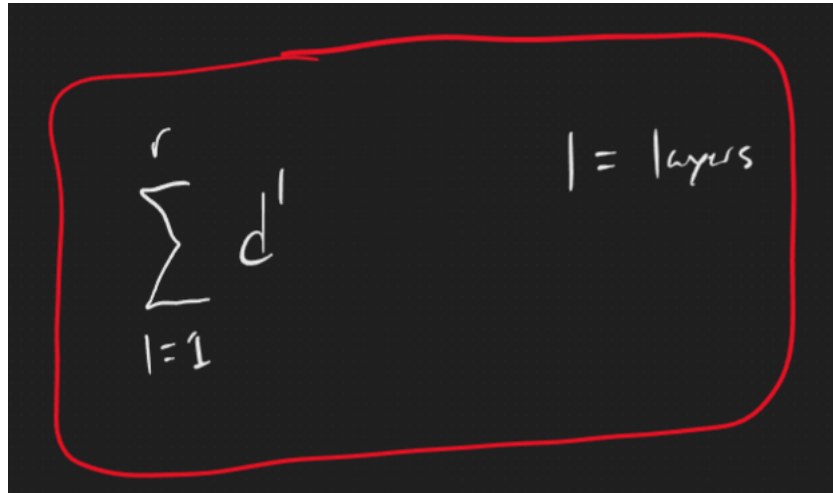
A photograph of a blackboard with a red border. On the blackboard, the equation $\sum_{l=1}^r d^l$ is written in white chalk. To the right of the equation, the text "| = layers" is written in white chalk.

Figure 1: Max iterations required for solution

This equation, summarizes the worst-case amount of children d^l from the first layer to 'r' layers since BFS only has to search until the end node is found at the 'r' layer.

0.2 B

Yet again, to find the solution we need to observe the worst scenario which is depicted in the previous question. To solve this, we need to store each nodes "path" for each node. This means that for each layer, if we multiply d_l with $(l+1)$ we arrive at the total memory for that layer. We need to $+ 1$ to account for the first node in the zeroth layer.

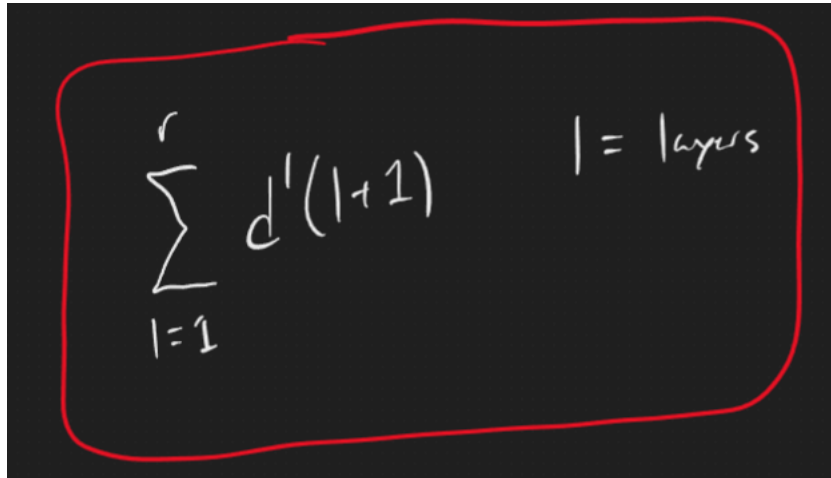
A handwritten formula on a black background, enclosed in a red rounded rectangle. The formula is $\sum_{l=1}^r d_l(l+1)$ followed by $l = \text{layers}$.
$$\sum_{l=1}^r d_l(l+1) \quad l = \text{layers}$$

Figure 2: Maximum amount of memory required for solution

Assignment 8



After finding the 2 layouts that cause an infinite loop, the problem is clearly shown. The problem is that when the label 1 is on the top node in the triangle path, the DFS algorithm will never reach the goal. Our solution for this is to add a feature to DFS which increments the label when it has been visited. With this fix, the path will never end up as an infinite loop, instead traverse the triangle loop x amount of times depending on the goal nodes label value.

Question 3

0.3 A

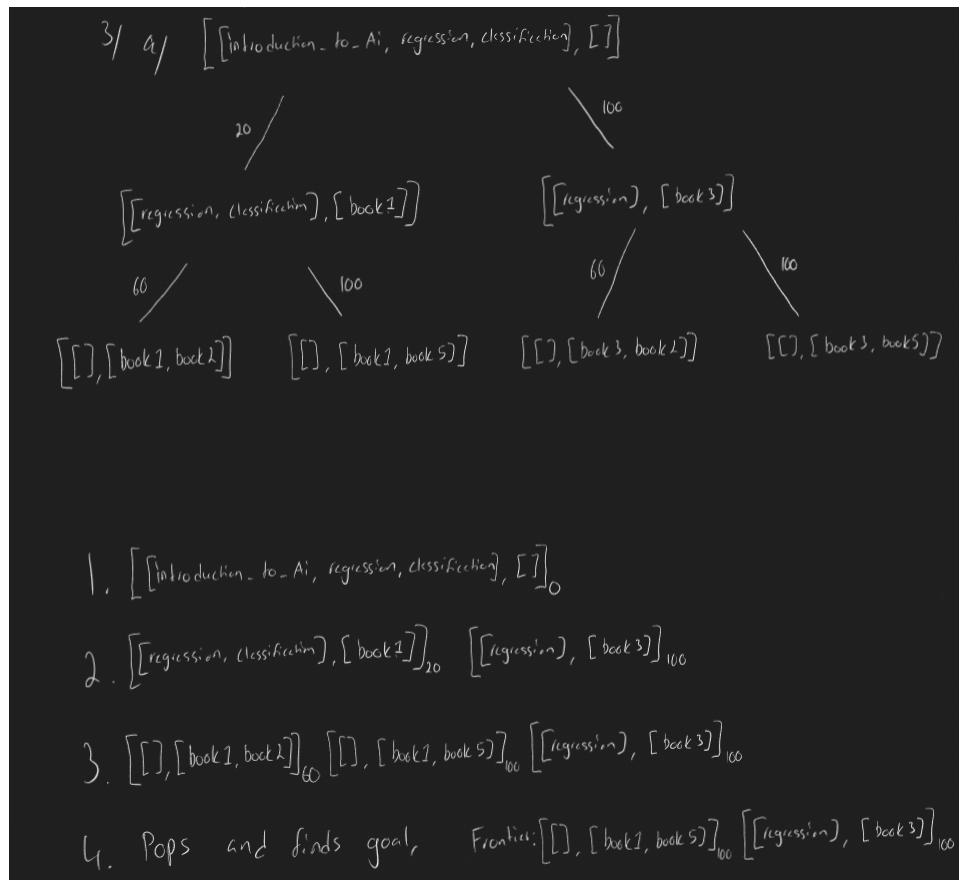


Figure 4: Tree and frontier-list step by step

0.4 B

We decided our heuristic would be (Page numbers) / (Categorizes removed by book), this represents the average number of pages used to cover each topic. Using this for part A's trees second layer, the left child would be $20/1 = 20$, while the right child would be $100/2 = 50$. In this case, it would prioritize the lower weight which is 20.

Question 4

0.5 A

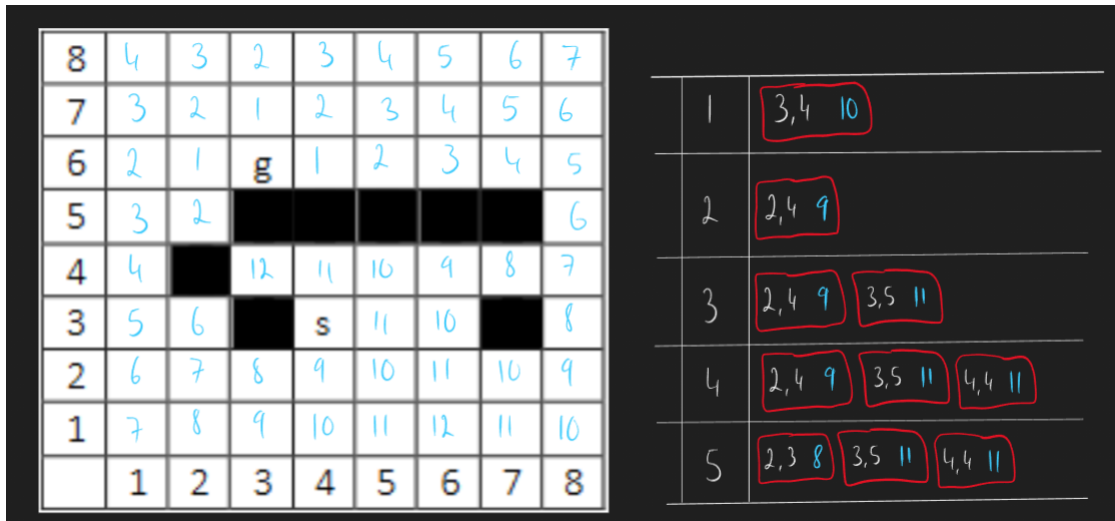


Figure 5: 5 Iterations of A*

0.6 B

Ranking the 3 algorithms from best to worst by efficiency it would be Best-first-search, A*, and then BFS.

Best-first-search is the best performing algorithm in this case. It traverses the grid looking for the nodes with the lowest cost storing the children in priority queues by weight. This leads it to instantly start to move towards the goal, which in this case is on the first best path it finds. This is why it performs better compared to the A* algorithm since in this case, the first path it started traversing was the best path.

A* explores by ranking the children of the start node by weight (Manhattan distance). Then it pops the first child from the list (which is the child with the lowest weight) and explores and inserts its children into the list. Then the process repeats until the goal is found.

Comparing the A* algorithm with the Best-first-search algorithm, A* is more consistent in finding the goal since it takes multiple equal weighted paths into consideration, which will allow it to outperform the best-first-search algorithm in more complicated grids, where the first path isn't the best.

BFS is in this case the worst performing algorithms since it doesn't care about heuristics like Manhattan distance. This means that it explored each nodes children in its usual pattern which resembles flooding all the children of the start node and outward. This means it has to explore an equal amount of nodes to the goal node in all directions.

0.7 C

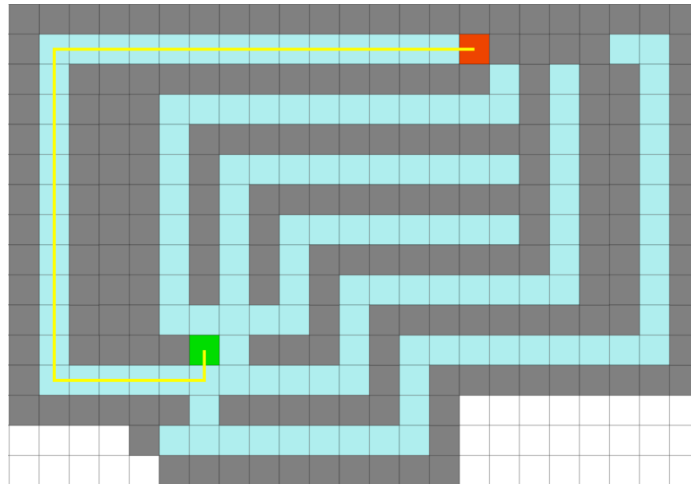


Figure 6: Grid where BFS outperforms Best-first-search

In the grid above best-first-search requires 270 iterations to find the goal while BFS only needs 269. This is because best-first-search gets lured into the paths that seem shorter but leads to dead ends, while BFS traverses all paths equally.

The reason the best-first-search algorithm is one iteration slower is that it traverses twice to the same node in the beginning, but since it finds nodes with lower weights they get prioritised first. But since the path it prioritises as worst in the beginning ends up being the only path, BFS finds the goal one iteration faster.