DAT405 Assignment 8 – Group 95

Martin Blom - (2 hrs)
Jakob Windt - (4 hrs)

March 10, 2023

Question 1

0.1 A

The maximum amount of BFS iterations in terms of ”d” and ”r” is $2\hat{r}$ + d - 1. The reason is because each layer has double the amount of children compared to the previous layer. This is what gives us the $2\hat{r}$ part of the equation. Since r is the length to the goal from the start, and not the amount of total layers, we need to add with d which is the amount of children in the final layer. Lastly, we need to subtract with 1 because the first layer starts with 1 child making it uneven.

0.2 B

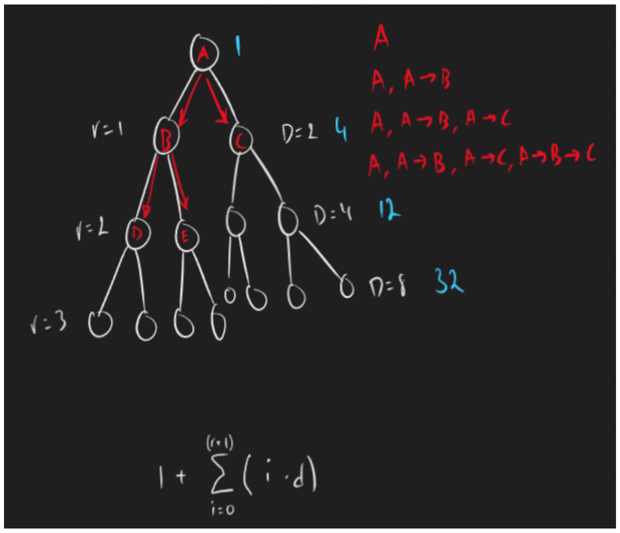


Figure 1: Max memory equation (seen at bottom of tree)

Assuming that ”each entire path” is the path from node ”A” to a goal node. The maximum amount of memory required is the summation of each entire paths memory usage. We calculate this by summarizing the multiplication of each layers number of nodes (d) with their respective number of nodes to the start (r+1), then add 1 to account for the start node.

Question 2

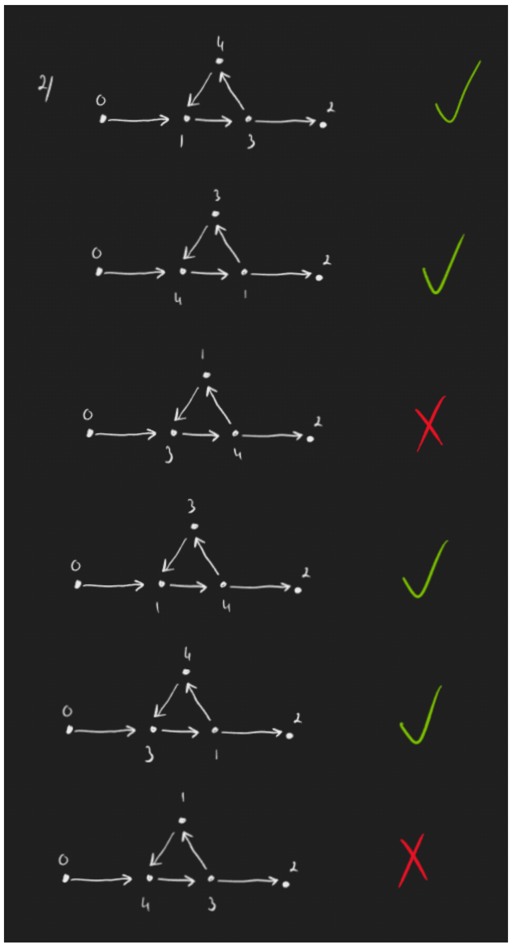


Figure 2: Tree and frontier-list step by step

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

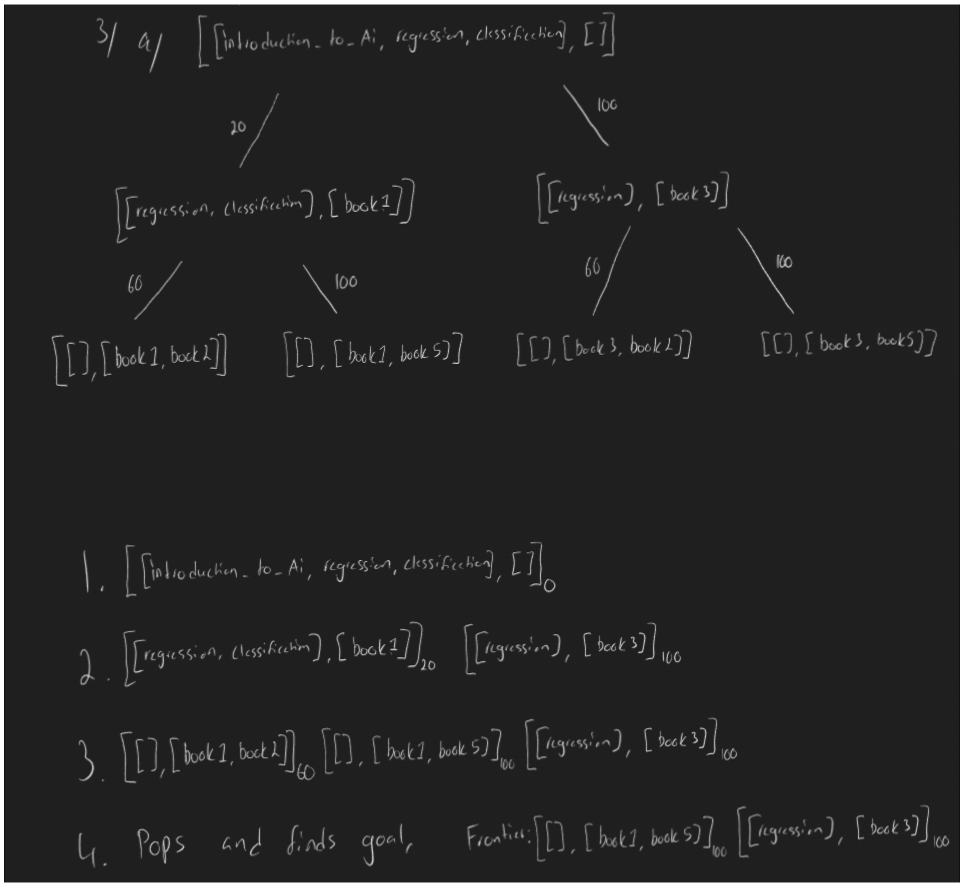

3/ a/ [[introduction_to_Ai, regression, classification], []]
20
100
[[regression, classification], [book 1]]
[[regression], [book 3]]
60
100
60
100
[[], [book 1, book 2]]
[[], [book 1, book 5]]
[[], [book 3, book 2]]
[[], [book 3, book 5]]
1. [[introduction_to_Ai, regression, classification], []]0
2. [[regression, classification], [book 1]]20 [[regression], [book 3]]100
3. [[], [book 1, book 2]]60 [[], [book 1, book 5]]100 [[regression], [book 3]]100
4. Pops and finds goal, Frontier: [[], [book 1, book 5]]100 [[regression], [book 3]]100

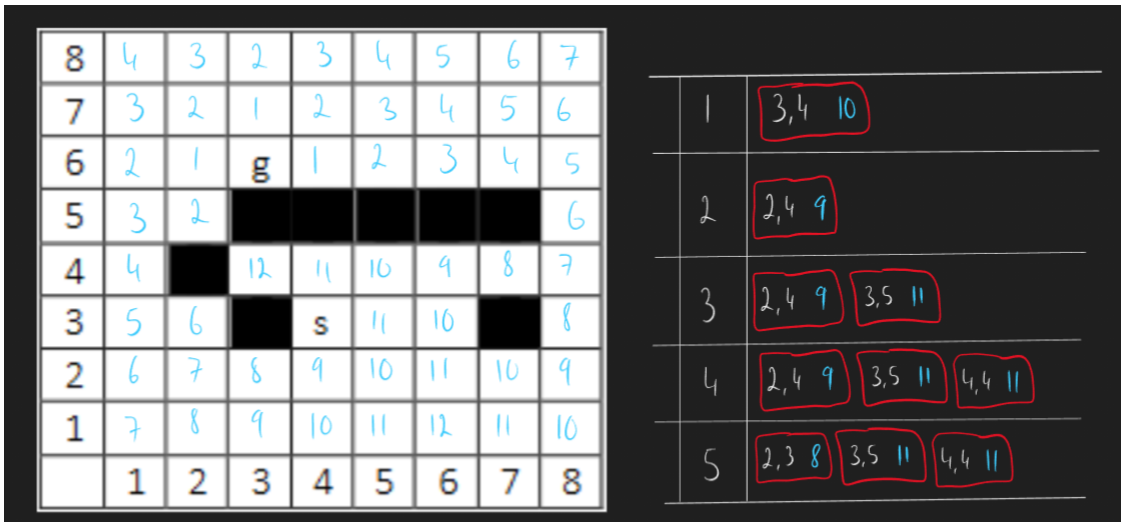
Figure 3: Tree and frontier-list step by step

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



8	4	3	2	3	4	5	6	7
7	3	2	1	2	3	4	5	6
6	2	1	g	1	2	3	4	5
5	3	2	■	■	■	■	■	6
4	4	■	12	11	10	9	8	7
3	5	6	■	s	11	10	■	8
2	6	7	8	9	10	11	10	9
1	7	8	9	10	11	12	11	10
	1	2	3	4	5	6	7	8

1	3,4 10		
2	2,4 9		
3	2,4 9	3,5 11	
4	2,4 9	3,5 11	4,4 11
5	2,3 8	3,5 11	4,4 11

Figure 4: 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

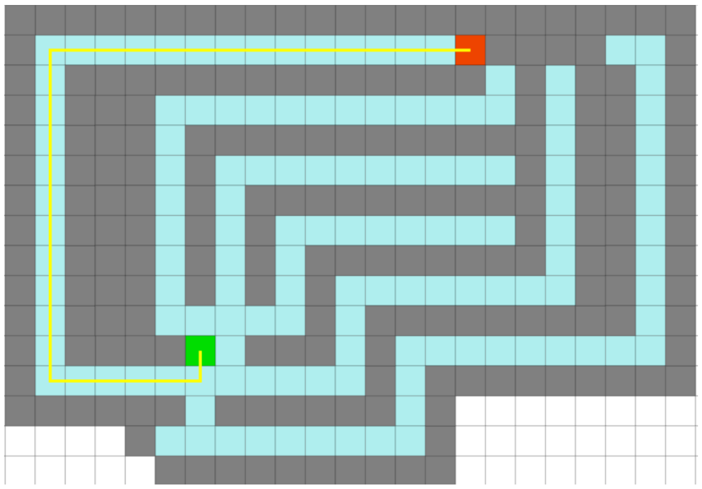


Figure 5: Grid where BFS outperforms Best-first-search

In the grid above best-first-search requires 270 iterations to find the goal while BFS only needs 269. This is because best-first-searched 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 priorities as worst in the beginning ends up being the only path, BFS finds the goal one iteration faster.