DAT405 Assignment 8 – Group 38

Julia Szczepaniak - (10 hrs)
Binsha Nazar Othupalliparambu Nazar - (10 hrs)

May 24, 2023

Problem 1

The branching factor 'd' of a directed graph is the maximum number of children (outer degree) of a node in the graph. Suppose that the shortest path between the initial state and a goal is of length 'r'.

a) What is the maximum number of Breadth First Search (BFS) iterations required to reach the solution in terms of 'd' and 'r'?

b) Suppose that storing each node requires one unit of memory and the search algorithm stores each entire path as a list of nodes. Hence, storing a path with k nodes requires k units of memory.What is the maximum amount of memory required for BFS in terms of 'd' and 'r' ?

Answer : a) BFS means breadth first search, so the algorithm will mark the children nodes at each iteration and choose nodes from left to right in the same "row" of the graph.

In the worst case, the shortest path between the starting node and the goal will have length r. This means that BFS will have to explore all the nodes that are at most r steps away from the starting node. The number of nodes that are at most r steps away from the starting node is given by the sum of the first r powers of d.

Thus, the number of growing nodes is following a geometrical progression with the branching factor d as common ratio, and the first number of nodes as the scale factor a which is 1 in our case: $U_{n+1} = d \times U_n$ where U_0 = a. We can also write this as $U_n = ad^n$. Thus, the sum of this progression is written like this :

$$\sum_{k=1}^{n-1} ad^k = a\frac{1-d^n}{1-d} = \frac{1-d^n}{1-d}$$

Finally, the variable n stands for the length of the graph. In our case it's r. Finally we have :

$$\frac{1-d^r}{1-d}$$

If we assume that r is the number of edges between two nodes, then this formula needs to be written as follow :

$$\frac{1-d^{r-1}}{1-d}$$

b) The maximum amount of memory required for BFS is the same as the maximum number of BFS iterations required to reach the solution, which is O(dr). This is because BFS needs to store a list of

all the nodes that it has explored. The number of nodes that BFS has explored is equal to the number of iterations that it has performed. Therefore, the maximum amount of memory required for BFS is O(dr)

We can use a similar reasoning. Indeed, if we take the worst scenario, we take the maximum amount of child nodes in each node. Thus, storing paths of nodes means storing the geometric progression at step r with k unit for each nodes.

Then, we can reuse our geometric progression and write: $k_{max} = ad^r$ But r means again the length between the starting nodes and the last. So we shift the geometric progression to :$k_{max} = ad^{r-1} = d^{r1}$ where a is the number of starting nodes

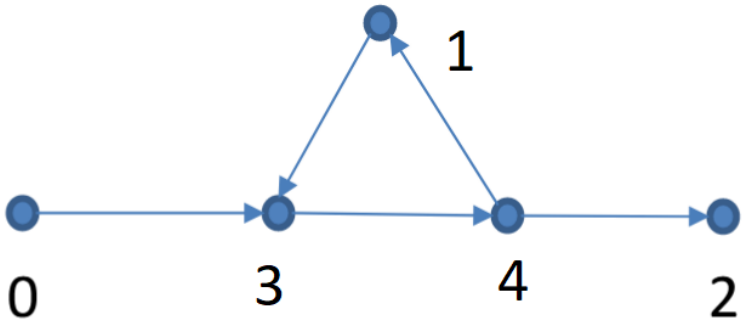


Figure 1: 1st solution

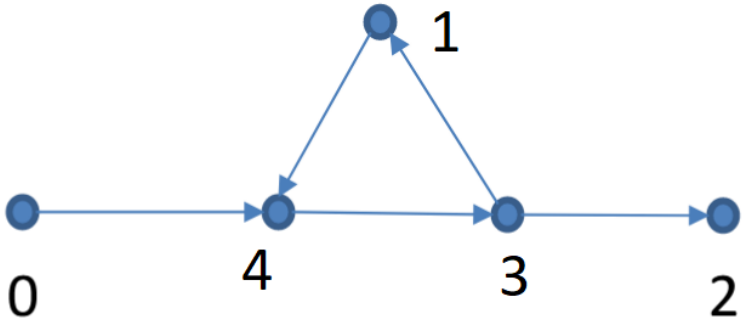


Figure 2: 2nd solution

Problem 2

Take the following graph where 0 and 2 are respectively the initial and the goal states. The other nodes are to be labelled by 1,3 and 4.

Suppose that we use the Depth First Search (DFS) method and in the case of a tie, we chose the smaller label. Find all labelling of these three nodes, where DFS will never reach to the goal! Discuss how DFS should be modified to avoid this situation?

Answer : In this problem we won't reach the goal if there is a 1 in a tie. Such nodes arrangement is shown in figures 1 and 2. In such case an algorithm would go in circle through (4, 3, 1) or (3, 4, 1) without reaching 2. To avoid such situation an algorithm should choose a new path (if possible) when it revisits a node.

Problem 3

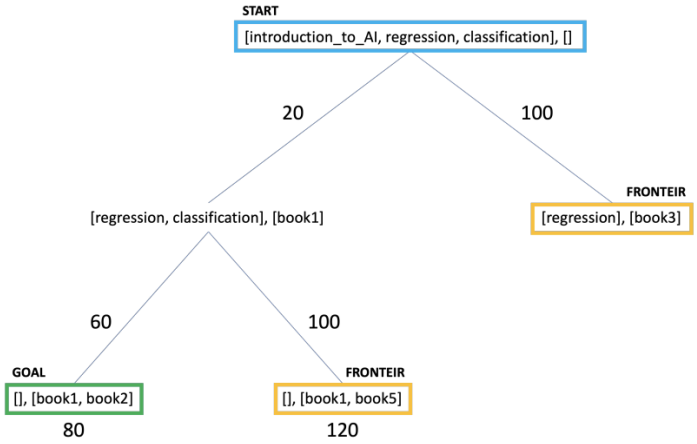
A publisher allows teachers to "build" customised textbooks for their courses by concatenating the text from different books in their catalogue. The catalogue contains the following books, together with the number of pages that the book contains, and the topics covered in that book:

a) Suppose a teacher requests a customised textbook that covers the topics (introduction

to AI,regression, classification) and that the algorithm always selects the leftmost topic when generating child nodes of the current node. Draw (by hand) the search space as a tree expanded for a lowest- cost-first search, until the first solution is found. This should show all nodes expanded. Indicate which node is a goal node, and which node(s) are at the frontier when the goal is found.

b) Give a non-trivial heuristic function h that is admissible. [h(n)=0 for all n is the trivial heuristic function.]

Answer : a)



The leftmost topic is "introduction to AI", it is treated first in the child nodes with books 1 and 3. Book 1 is the one that costs the least (20 pages), it is the one that we will prefer. The other node with book 3 is therefore a "frontier" node.The last two topics "regression" and "classification" are both covered by books 2 and 5. Book 2 costs less (60 pages) than book 5 (100 pages), so we choose this one: this is our goal node. The other node with book 5 is therefore a "frontier" node.

b) The heuristic value of a node in a graph tries to define the importance of the value of this node. The number of topics left to process in a node can tell us if we are progressing in the right direction or not: it will therefore be our heuristic value that will allow us to minimize the final cost.This heuristic is also non-trivial because it is not always equal to 0.

Problem 4

Consider the problem of finding a path in the grid shown below from the position s to the position g. A piece can move on the grid horizontally or vertically, one square at a time. No step may be made into a forbidden shaded area. Each square is denoted by the xy coordinate. For example, s is 43 and g is 36. Consider the Manhattan distance as the heuristic. State and motivate any assumptions that you make.

a) Write the paths stored and selected in the first five iterations of the A* algorithm, assuming that in the case of tie the algorithm prefers the path stored first.

b) Solve this problem using the software in http://qiao.github.io/PathFinding.js/visual/ Use Manhattan distance, no diagonal step and compare A*, BFS and Best-first search.

Describe your observations. Explain how each of these methods reaches the solution. Discuss the efficiency of each of the methods for this situation/scenario.

c) Using a board like the board used in question 4a) or in http://qiao.github.io/PathFinding.js/visual/ , describe and draw a situation/scenario where Breadth-first search would find a shorter path to the goal compared to Greedy best-first search. Consider that a piece can move on the grid horizontally or vertically, but not diagonally. Explain why Breadth-first search finds a shorter path in this case

Answer :

a)

Let the heuristic be Manhattan distance from point p to final square g.

$$h(p, g) = |x_p - x_g| + |y_p - y_g|$$

Let the cost function be distance from starting point s to evaluated point p.

In figure 3 number in left corner of each square in the grid states for distance from starting point s and number in right corner states for value of heuristic $h(p, g)$, where p is set of coordinates of evaluated square. In each iteration square with lowest f value is chosen and then f values of it's neighbours are calculated. In each drawing the square with lowest f value from previous iteration is marked with orange colour.

iteration	stored paths	selected path (lowest f value)
1	(43,44),(43,53),(43,42)	(43,44)
2	(43,44,34), (43,44,54), (43,53), (43,42)	(43,44,34)
3	(43,44,54),(43,53),(43,42)	(43,44,54)
4	(43,44,54,64),(43,53),(43,42)	(43,53)
5	(43,44,54,64),(43,53,63),(43,53,52),(43,42)	(43,42)

b)

All the algorithms performed similarly well. They found the optimal path in similar time. However BFS needed the biggest number of operations. Comparison

	A*	Breadth-First Search	Best-First Search
path length	10	10	10
time	0 ms	1 ms	1 ms
operations	71	364	48

However time vary in each try.

A* algorithm

The algorithm uses evaluation function to determine which node to be expanded first while searching. The evaluation function is a sum of heuristic function and path cost. It keeps all the nodes in the memory while searching. It requires more memory than Best-First Search.

Breadth-First Search algorithm

It starts at the graph's root and visits all nodes in the current distance from root before moving on to the nodes in bigger distance. It always finds the shortest path. It has the similar time complexity as A* but needs more memory. BFS is more suitable for searching vertices close to the given source.

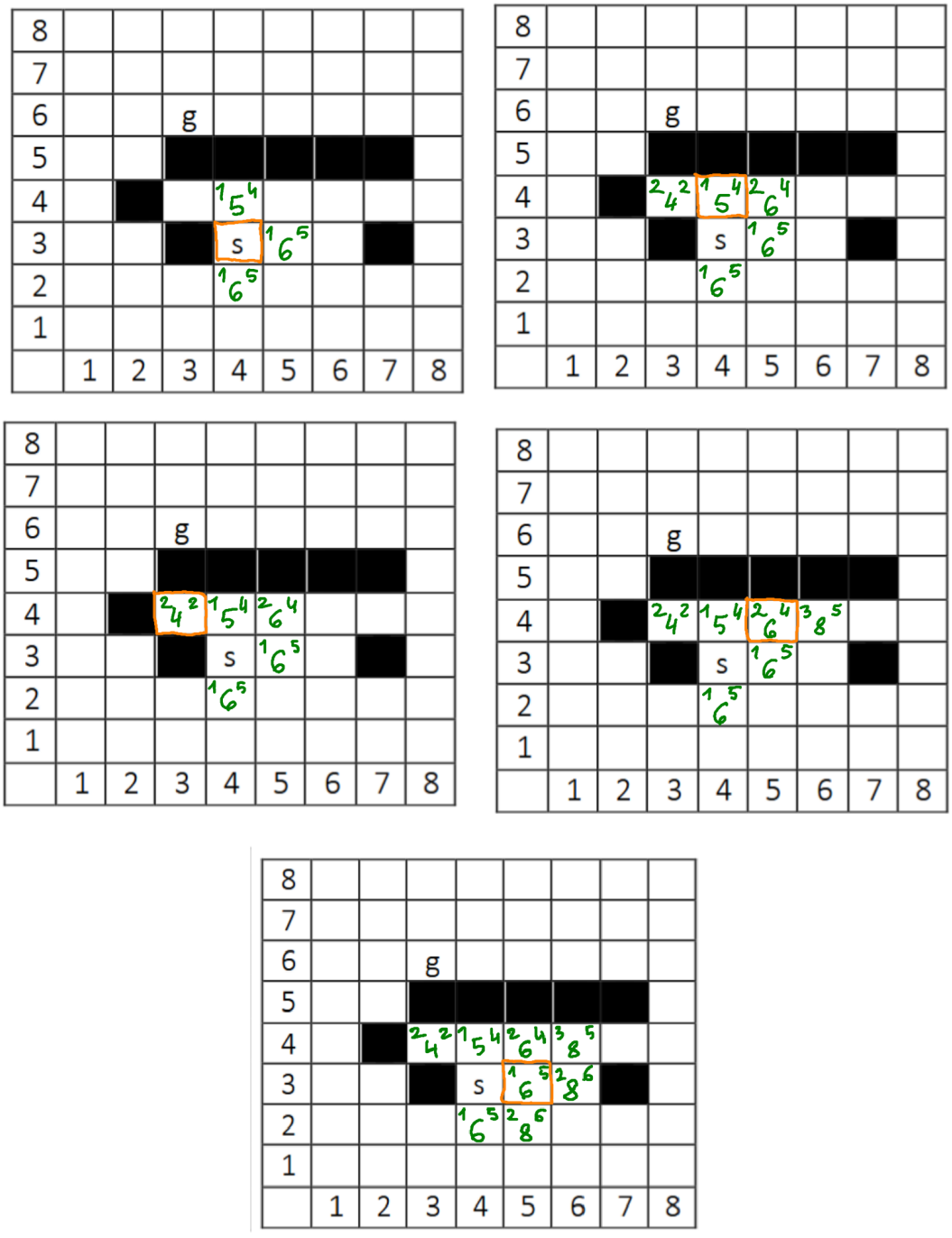


Figure 3: 5 iterations of A* algorithm

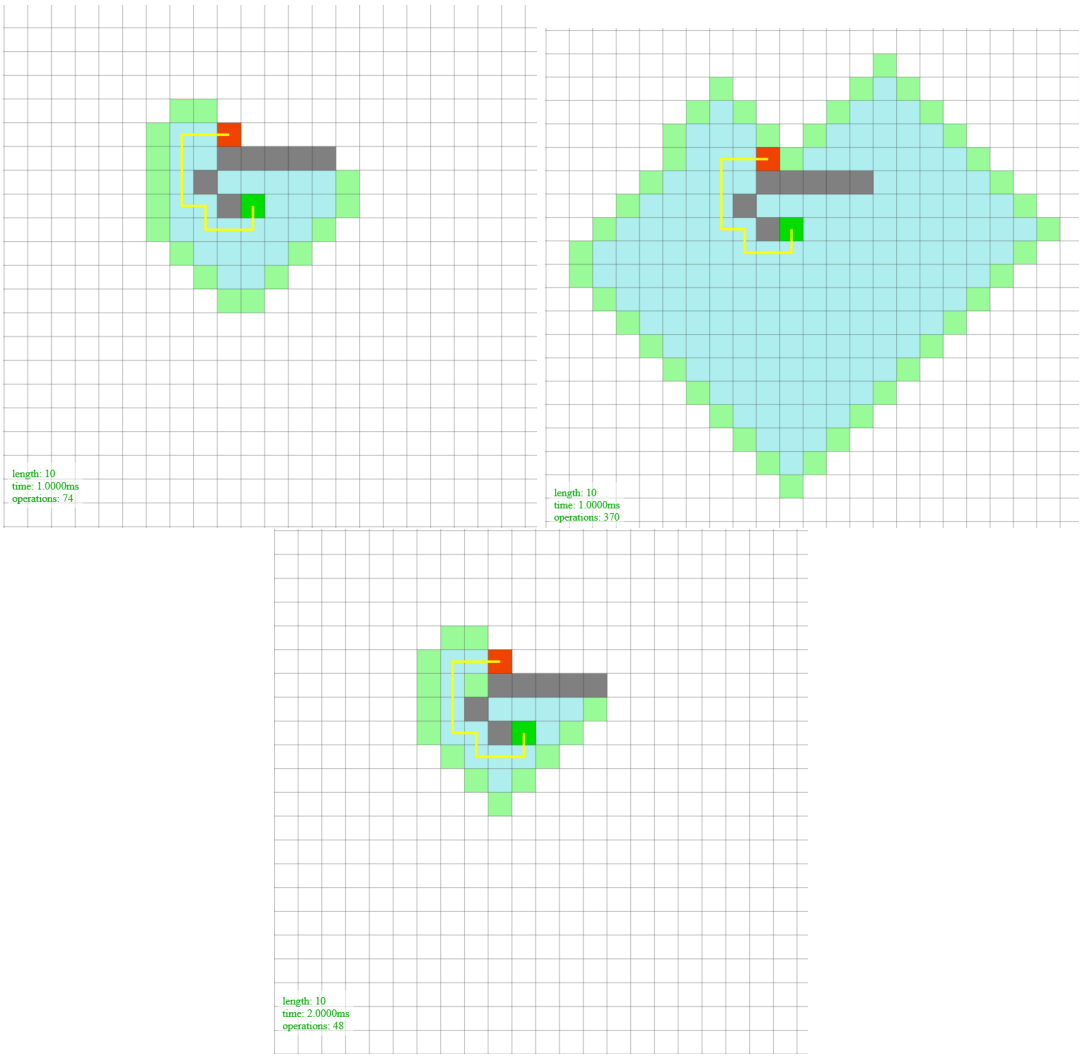


Figure 4: Comparison of A*, BFS and Best First Search algorithms

Best-First Search algorithm

The algorithm uses evaluation function to determine next step, which is heuristic function. It keeps all the fringe nodes in the memory while searching. It requires less memory than A*.

Summary

Every algorithm found found the same optimal path. However Best-First Search did it in smallest number of operations

c)

In Figure 5 there is a situation where Breadth-First Search found a path of 10 steps and Best-First Search found path of 20 steps. It happened because Best-First Search chooses squares which are close to the goal and doesn't explore less promising area. Breadth-First search goes in every direction and

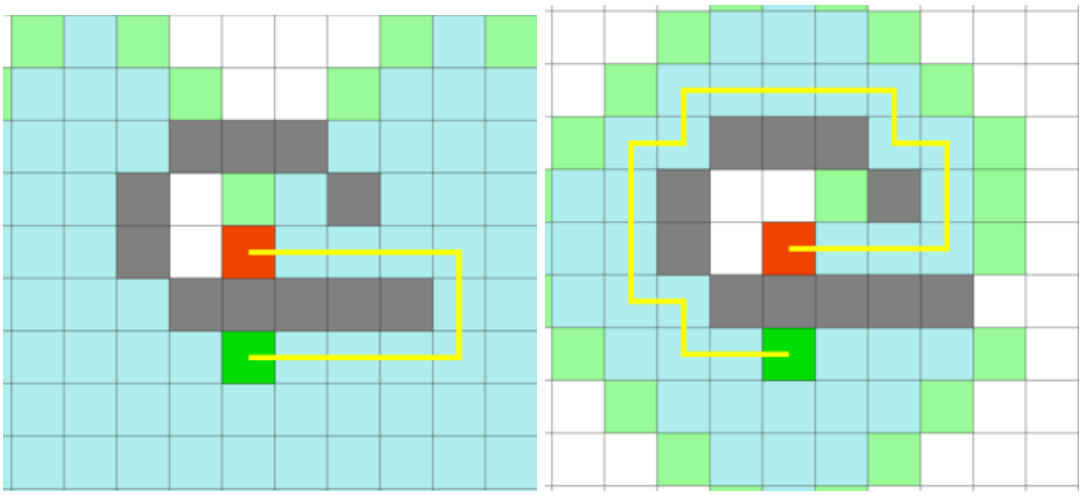


Figure 5: Example of situation of Breadth-First Search (left) choosing shorter path than Best-First Search (right)

it happens that path which seemed to be move away turned to direction of the goal quickly.