DAT405 Assignment 8 – Group 111

Oscar Karbin - (12 hrs)
Jihad Almahal - (12 hrs)

March 10, 2023

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'.

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

The maximum number of Breadth-First Search (BFS) iterations required to reach the solution can be calculated using the formula:

$d^0 + d^1 + d^2 + \cdots + d^r$

This formula calculates the maximum number of nodes that can be generated at each level of the search tree.

This is a geometric series with a common ratio of d. Using the formula for the sum of a geometric series, the maximum number of BFS iterations can be simplified as:

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

Therefore, the maximum number of BFS iterations required to reach the solution in terms of d and r is:

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

1.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' ?

The maximum amount of memory required for BFS can be calculated by considering the maximum number of nodes that can be present in the memory at any point in time.

In the worst case, BFS needs to store all the nodes in the search tree until it reaches the goal node. The maximum number of nodes that can be present in the memory at any given time is the maximum number of nodes generated at any level of the search tree.

The maximum number of nodes generated at level i of the search tree is d^i. Therefore, the maximum number of nodes that can be present in the memory at level i is:

$d + d^2 + d^3 + ... + d^i$

This is a geometric series with a common ratio of d. Using the formula for the sum of a geometric series, the maximum number of nodes that can be present in the memory at level i can be simplified as:

$(d^i - 1)/(d - 1)$

The maximum number of nodes that can be present in the memory at level r is:

$(d^r - 1)/(d - 1)$

Therefore, the maximum amount of memory required for BFS in terms of d and r is:

$r * (d^r - 1)/(d - 1) ==> O(rd^r)$

Note that this formula assumes that each node is stored only once in the memory. If a node can be generated multiple times during the search, then the actual memory requirements may be higher.

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?

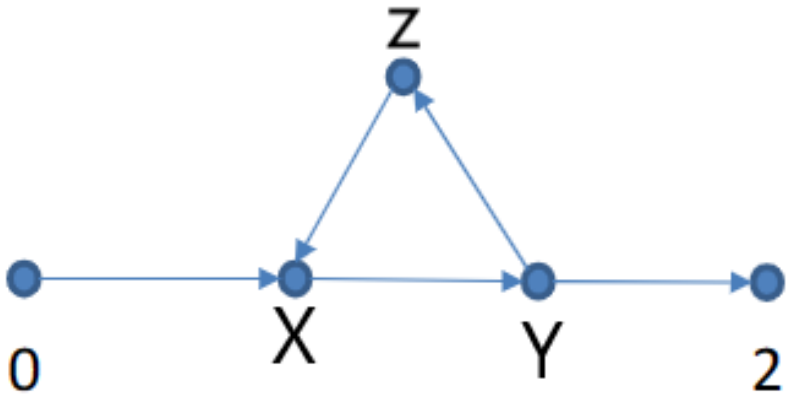


Figure 1: Graph with 5 nodes labeled 0, 2, x, y, z.

We want to find all labelings of the three nodes x, y, and z, such that DFS will never reach the goal node 2 from the start node 0.

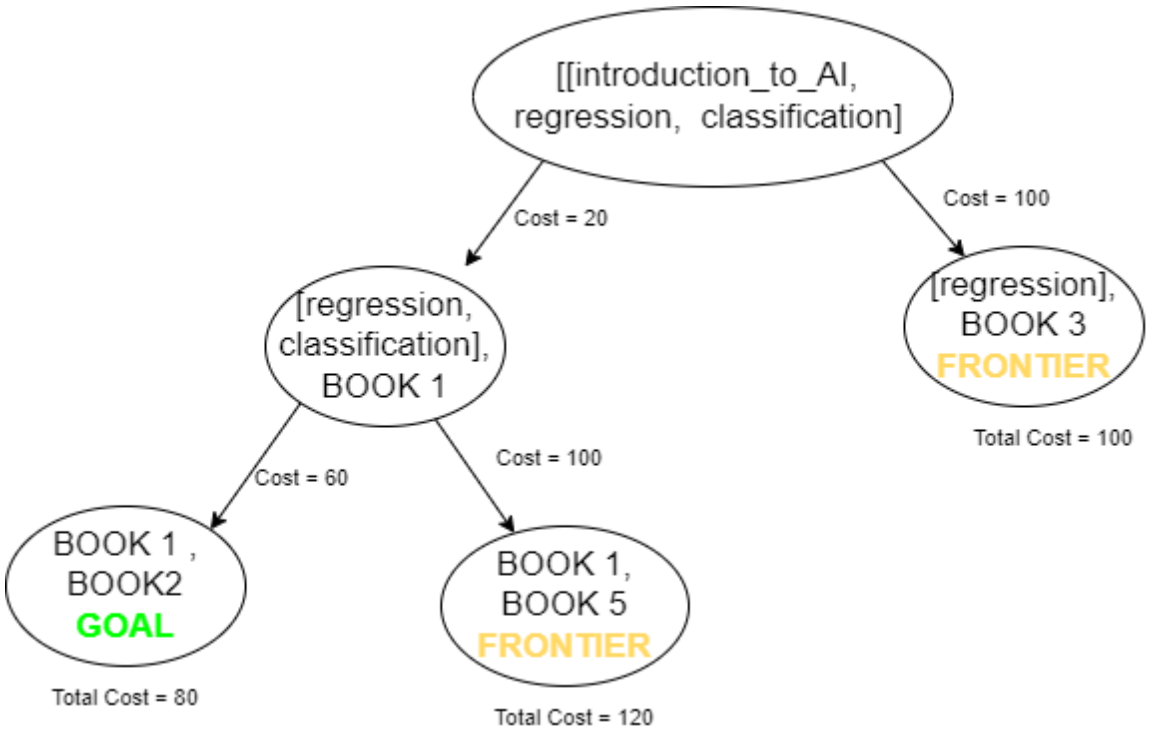
We start by traversing from 0 to x, which has only one search direction. From x, we can go to y. At y, we have two options: either go to 2 or go to an unlabeled node z (where z can be 1, 3, or 4).

We can identify one case where DFS will get stuck in an infinite loop. If we label z as 1, then DFS will always choose the edge (y, z) over (y, 2) since it prefers smaller labels. This will lead to an infinite loop.

To prevent DFS from getting stuck in an infinite loop, we can make two modifications to the algorithm. The first is to keep track of previously visited nodes and detect repeating patterns in the path. If a repeating pattern is found, we can break the loop by choosing a different path, if possible. The second modification is to use a variant of DFS called IDDFS. This method performs DFS with increasing depths, and limits the search to a certain threshold. By doing so, IDDFS avoids getting stuck in infinite loops and ensures that the search is completed in finite time.

3

3.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.



In the above tree, each node is represented as (Topics, Books). The start node is ([introduction_to_AI, regression, classification], []), indicating that we need to cover these three topics using no books initially. The goal node is ([], [book2, book1]), indicating that all topics are covered and the customised textbook is made up of books 2 and 1.

The tree is expanded using a lowest-cost-first search strategy, where the cost of an arc is equal to the number of pages in the selected book. At each node, the algorithm selects the leftmost topic from the list of Topics and generates child nodes by selecting a book that covers this topic and removing all of the covered topics from the Topics list.

The node ([], [book1, book2]) is the optimal solution since it covers all topics with the minimum number of pages.

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

One possible non-trivial heuristic function h that is admissible for this problem is as follows:

For a given node n = (Topics, Books), let the set of uncovered topics be represented as U(n) (i.e., the topics requested by the teacher that are not yet covered by the books in Books). Then, the heuristic function h(n) can be defined as the sum of the minimum number of pages needed to cover each topic in U(n), divided by the maximum number of pages in any single book that covers any topic in U(n).

Formally, $h(n) = \sum minPages(t)/maxPages(U(n))$, where t belongs to U(n), minPages(t) is the minimum number of pages needed to cover topic t (i.e., the number of pages in the smallest book that covers t), and maxPages(U(n)) is the maximum number of pages in any single book that covers any topic in U(n).

This heuristic function is admissible because it never overestimates the cost of reaching a goal node. Specifically, for any given node n, h(n) provides an estimate of the minimum cost to reach a goal node from n, assuming that the remaining topics in U(n) are covered by the smallest possible books that cover them. Since this assumes the minimum cost, it can only be an underestimate of the true cost. Additionally, h(n) is non-trivial because it takes into account the specific topics that still need to be covered, as well as the relative sizes of the books that cover those topics.

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.

4.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.

Iteration 1:

Path stored: (4,3) Path selected: (4,3)

Iteration 2:

Paths stored: (4,4)[1+3=4], (5,3)[1+5=6], (4,2)[1+5=6] Path selected: (4,4)

Iteration 3:

Paths stored: (3,4)[2+2=4], (5,3)[1+5=6], (4,2)[1+5=6], (5,4)[2+4=6] Path selected: (3,4)

Iteration 4:

Paths stored: (4,2)[1+5=6], (5,3)[1+5=6], (5,4)[2+4=6] Path selected: (5,3) (because it was stored first)

Iteration 5:

Paths stored: (4,2)[1+5=6], (5,4)[2+4=6] Path selected: (4,2) (because it was stored first)

4.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 efficienc y of each of the methods for this situation/scenario.

The three algorithms that can be used to find the shortest path to a goal that is located 10 length units away. The A* algorithm requires 71 operations to achieve this, while the Breadth-First Search algorithm requires 364 operations. On the other hand, the Best-First Search algorithm requires only 48 operations to reach the goal.

The A*, BFS, and Best-First Search algorithms are all commonly used to find the shortest path to a goal in various applications such as navigation, robotics, and gaming.

A*: It combines the actual cost of moving from the start node to a given node and the estimated cost from that node to the goal node using the heuristic function. It chooses the node with the minimum total cost for further exploration.

BFS: It explores all possible paths from the start node to the goal node and finds the shortest path. It uses a queue data structure to store the nodes to be explored.

Best-first search: It uses only the heuristic function to guide its search and chooses the node with the minimum heuristic value for further exploration.

Based on our result, it appears that the Best-First Search algorithm is the most efficient among the three, taking only 48 operations to reach the goal. However, it is important to note that the efficiency of these algorithms can vary depending on the specific problem and its constraints.

4.c Using a board like the board used in question 4a) 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.

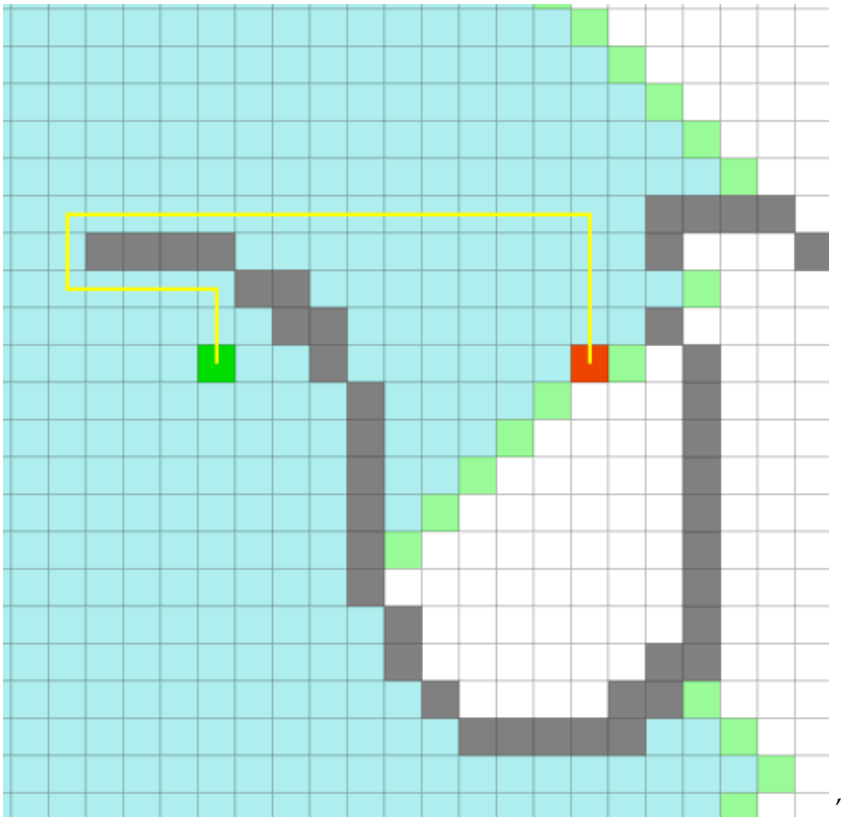


Figure 2: A BFS algorithm that finds a length of 26 units to the goal.

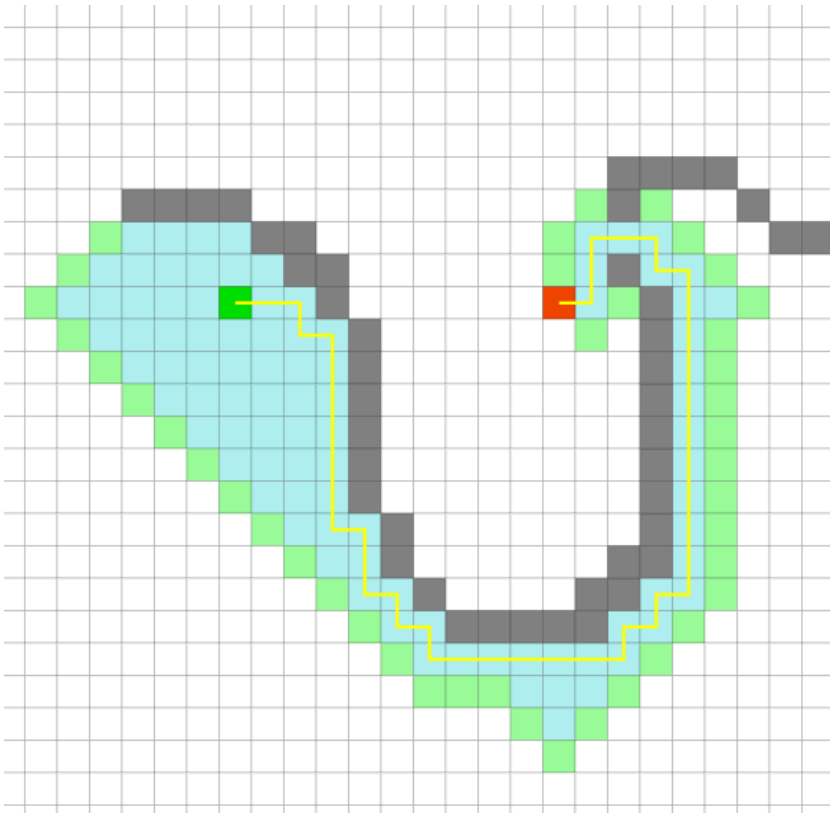


Figure 3: A Best-First search algorithm that finds a length of 44 units to the goal.

The reason Breadth-first search finds a shorter path in this case is because it explores all nodes at a given distance from the starting point before moving to the next distance level. In contrast, Greedy best-first search may skip over some nodes at a given distance if they have a higher heuristic value, and may end up exploring nodes that are farther away from the starting point. In this scenario, the shortest path happens to be at a lower distance level, and so BFS is able to find it more efficiently.