DAT405 Assignment 8 – Group 67

Student Yaochen Rao – (15 hrs)
Student Wanqiu Wang – (15 hrs)

October 24, 2022

Problem 1

A:

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

Answer A):
Since there are different understandings of the parameters in the question, we use the following figure to explain our understandings in different cases. And on the basis of different understanding to answer.

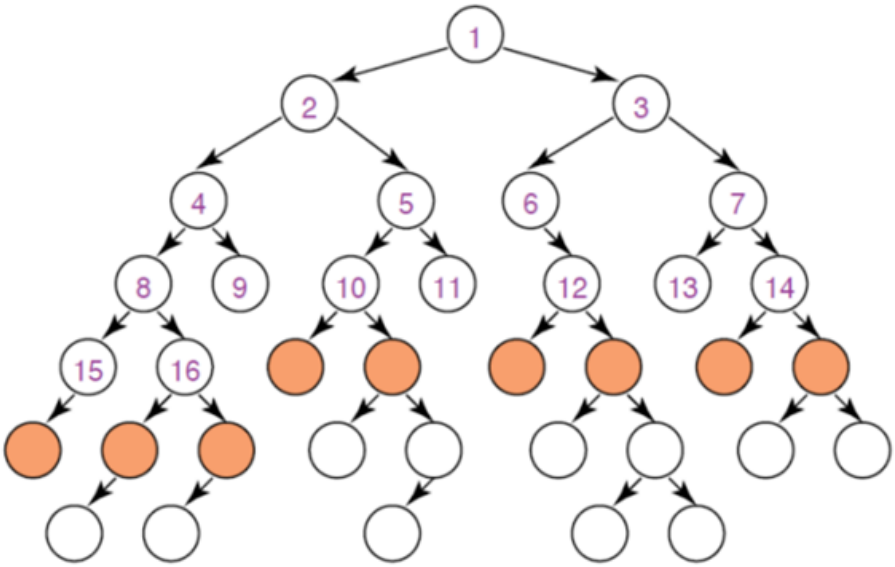


Table 1: An example of BFS

Because 'd' is the maximum number of children of a node in the graph, and 'r' is the length of shortest

path between the initial state and a goal.
So, if we want to get the maximum number of BFS, it means the BFS will search at worst case. We need the visited all nodes before reaching the end nodes. And all nodes have the maximum number of children.
1)if we assume the 'r' means the number of edges between the initial node and goal node. Like the picture above, 'd' is 2, 'r' is 4 (1(initial state)->2, 2->5, 5->10, 10->goal state).
So the maximum number of BFS iterations is:

$$\sum_{i=0}^{r} d^i = \frac{1-d^{r+1}}{1-d}$$

2)And if we assume the 'r' means the number of nodes between the initial node and goal node. Like the picture above, 'd' is 2, 'r' is 5 (1(initial state), 2, 5, 10, goal state).
So the maximum number of BFS iterations is:

$$\sum_{i=0}^{r-1} d^i = \frac{1-d^{r}}{1-d}$$

B:

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 B):
Because 'd' is the maximum number of children of a node in the graph, and 'r' is the length of shortest path between the initial state and a goal.
So, if we want to get the maximum amount of memory required for BFS, it means the BFS will search at worst case. We need the visited all nodes before reaching the end nodes. And all nodes have the maximum number of children.
1)If We assume 'r' means the number of nodes between the initial node and goal node.
A) If we assume that we're only going to store a path from the start point to the last iteration. Like the picture above, we only store (1-2-5-10-goal, 1-3-6-12-goal, etc.) So, the maximum amount of memory required for BFS is

$$r * d^{r-1}$$

B) If we assume that we store the path after each iteration. Like the picture above, we only store (1, 1-2, 1-3, 1-2-4, etc.)
So, the maximum amount of memory required for BFS is

$$\sum_{i=0}^{r-1} (i+1) * d^i$$

2)If We assume 'r' means the number of edges between the initial node and goal node.
A) The assumption is same as 1) A, the maximum amount of memory required for BFS is

$$(r+1) * d^r$$

B) The assumption is same as 1) B, the maximum amount of memory required for BFS is

$$\sum_{i=0}^{r} (i+1) * d^i$$

Problem 2

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 2):

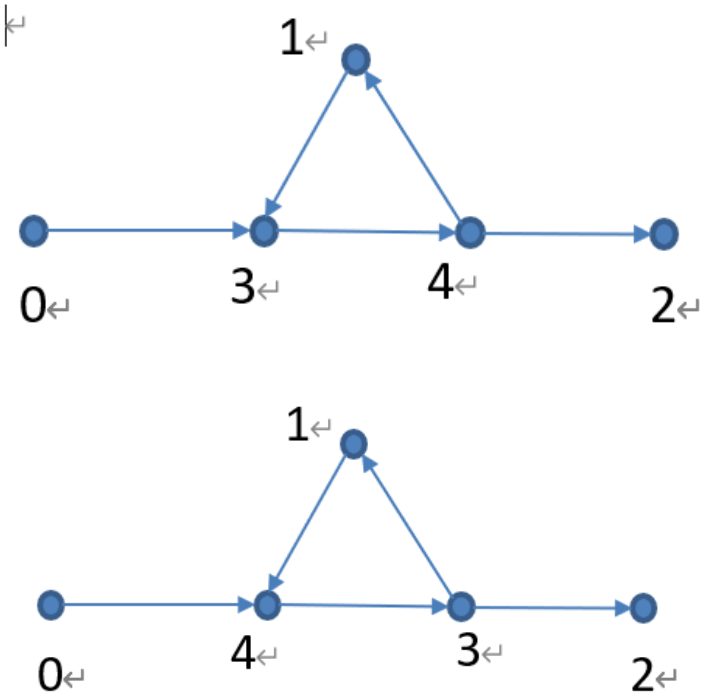


Table 2: Two situations where DFS will never reach the goal

We set the label of the topmost node to 1. Then if we use the DFS method and in the case of a tie, we chose the smaller label, DFS will never reach to the goal. All labelling of these three nodes like above, there are two cases. We can see when we get node 4(the first case) or node 4(the second case). Then in the case of tie, so we chose the smaller label (1<2), we go to top node and It's going to go clockwise around all three ends of this triangle.
How to avoid this situation:
1) We can keep track of our own path in the case of tie. So, DFS can avoid repeating the same route the next time it recycle to the same point.
2) We change DFS to IDDFS(Iterative Depending DFS), The main idea is to combine DFS and BFS, and then set the depth of the search for an initial value (for example, 1), and then search with DFS, and if you don't find the answer, you iterate down. This iterative process uses the idea of BFS search in the breadth of each layer (the inner layer can also be used with the DFS), but the outer layer is DFS.

Problem 3

A:

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.

Answer A:

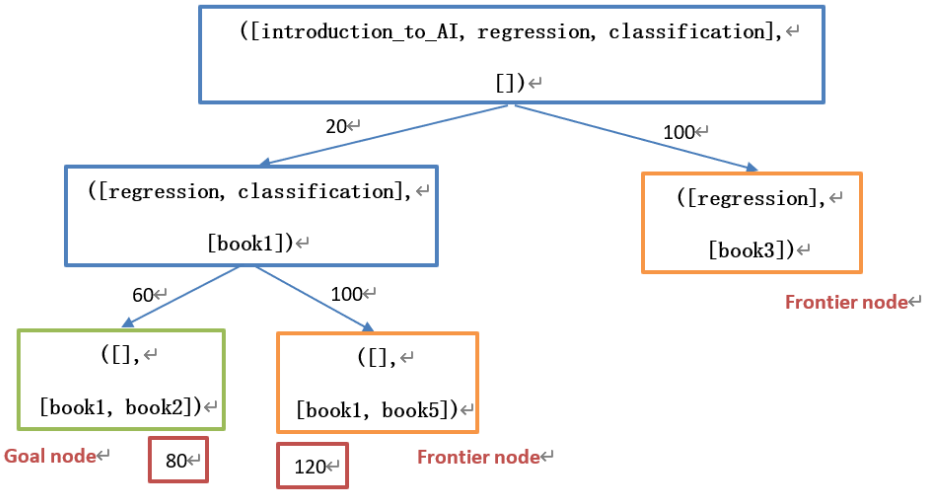


Table 3: A tree expanded for a lowest-cost-first search

Because the algorithm always selects the leftmost topic when generating child nodes of the current node.
At first, choose the leftmost topic is [introduction_to_AI], we get two child nodes about book1 and book3. And the less cost is 20, The frontier is [book3_100].
Then, choose the leftmost topic is [regression], we get two child nodes about book2 and book5. The frontier now is [[book1, book2]_80 ,[book1, book5]_120 ,[book3_100]_100] We can see all topics are covered , and the less cost is 80, so add [book1, book5] to the final frontier. The frontier when the goal is found is(([regression], [book3]), ([], [book1, book5])). The goal node is ([], [book1, book2]).

B:

B) Give a non-trivial heuristic function h that is admissible. [h(n)=0 for all n is the trivial heuristic function.]
Answer B:
A heuristic function can be defined as more topics are covered. It means that give priority to books that cover more topics and cost less. In other words, subtract the number of topics covered in the selected book from the current number of topics. The number of topics obtained later is used as one of the judgment criteria.

Like the question a: The topics which need to be cover are [introduction_to_AI, regression, classification], so book2, book3 and book5 all cover two of the topics (max topics can be covered). The cost of each one is 60, 100 and 100. Then we choose book2. Then find the book which cover the topic [introduction_to_AI] and the cost is less.

Problem 4

A

The Manhattan distance between point (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. We calculate F cost = G cost + H cost for a point's surrounding nodes. G cost is the movement cost and the H cost is the Manhattan distance. We choose the point that have the minimum F cost. The iterations are as follows:

Iteration-1

(x_k, y_k)	Previous Point	G cost	H cost	F cost
(4, 3)	START	0	4	4

Iteration-2

(x_k, y_k)	Previous Point	G cost	H cost	F cost
(4, 4)	(4, 3)	1	3	4
(5, 3)	(4, 3)	1	5	6
(4, 2)	(4, 3)	1	5	6

Choose (4, 4).

Iteration-3

(x_k, y_k)	Previous Point	G cost	H cost	F cost
(3, 4)	(4, 4)	2	2	4
(5, 3)	(4, 3)	1	5	6
(4, 2)	(4, 3)	1	5	6
(5, 4)	(4, 4)	2	4	6

Choose (3, 4).

Iteration-4

(x_k, y_k)	Previous Point	G cost	H cost	F cost
(5, 3)	(4, 3)	1	5	6
(4, 2)	(4, 3)	1	5	6
(5, 4)	(4, 4)	2	4	6

Choose (5, 3).

Iteration-5

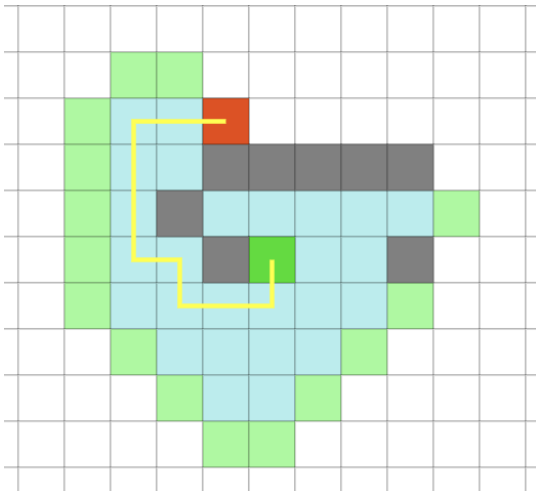
(x_k, y_k)	Previous Point	G cost	H cost	F cost
(5, 4)	(4, 4)	2	4	6
(3, 2)	(4, 2)	2	4	6
(6, 3)	(5, 3)	2	6	8
(5, 2)	(5, 3)	2	6	8
(4, 1)	(4, 2)	2	6	8

Choose (4, 2).

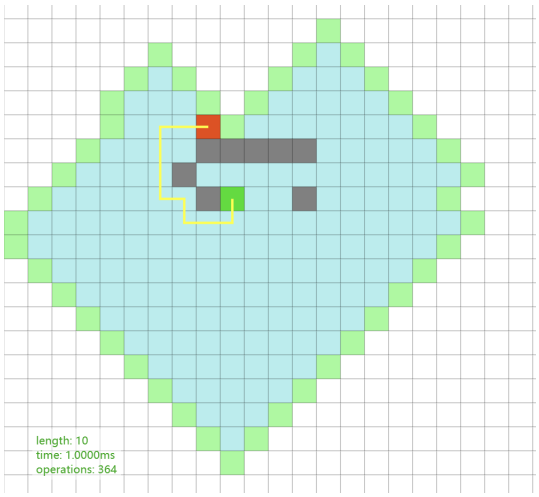
B

The graph of A*, BFS, Best-first search are as follows.

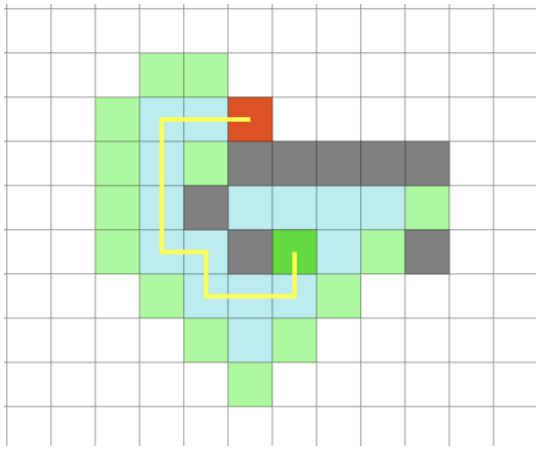
A*



BFS



Best-First Search.



Algorithm	Length	Time	Operations
A*	10	0.2ms	71
BFS	10	0.4ms	364
Best-First Search	10	0.2ms	48

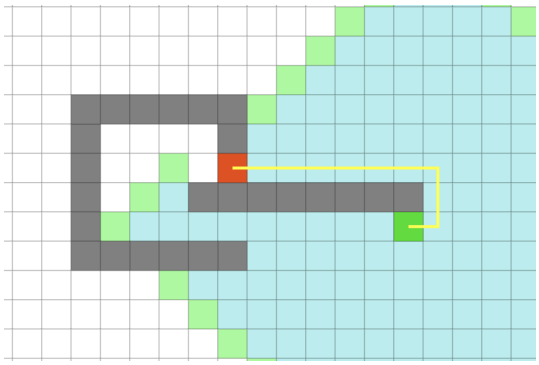
From above we can see that the Best-First Search algorithm is the most efficient one, it performs 48 operations and reached the goal. BFS, however, is the slowest one, it performs 364 operations.

The reason why BFS is slower than the other two algorithms is because it doesn't have prior knowledge about the problem, which means it will systematically expand and check all nodes in the graph to find the results, this is much more time consuming.

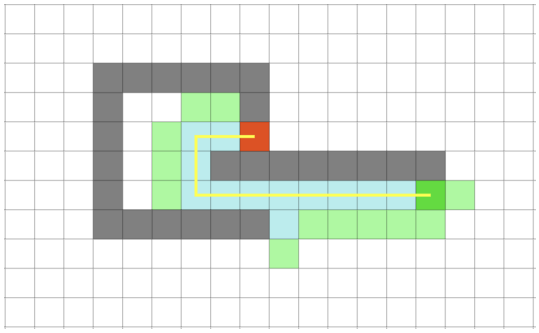
The difference between Best First Search and A* algorithm is in Best First Search $f(n) = h(n)$, while in A* algorithm, $f(n) = g(n) + h(n)$. This means the Best First Search only consider the node which have the smallest distance (heuristic value) to the goal, it doesn't consider past knowledge. For the A* algorithm, it takes the past knowledge into account, which means it will not only consider next node that have the smallest distance (heuristic value) to the goal, but also consider the cost of getting to that node. In this case, it is easy to see that when encounter several obstacles, A* algorithm will have higher cost compared to Best First Search, this leads to the result that it has more operations than Best First Search.

C

BFS (length=10, operations=378)



Best-First Search (length=12, operations=40)



Above is the example of BFS find the shortest path rather than Best-First Search. Best First Search will go to the left side because it is depend on the heuristic function. At the starting point, going to right will have bad heuristic so it won't do that. Best First Search thought it goes to the right way but ignore the fact that we set up obstacles at the corner to reach the goal. However, for BFS, it focus on scan the whole nodes and find the shortest path.

Problem 5

A

The purpose to MDP is to find a strategy that makes it possible to choose a certain action in a certain state—an action that maximizes the gain. The purpose of the generic search problem is to find the path between the starting point and the goal. Generic search problem can be converted to MDP, which can be achieved by converting the environment of the search problem into a state space and defining action functions. Furthermore, the probability distribution of these actions should be between 0 and 1. Thus, we can say the generic search problem is a subset of the MDP, which means any generic search problem can be described as MDP and MDP is a generalization of the generic search problem.

B

The advantage of value iteration over A* is it can always find the optimal path from starting point to the goal. Because for the value iteration, it must converge to one point. This conclusion is derived from the algorithmic principle of value iteration, and we can reach this conclusion by proving that the Bellman expectation equation converges to a unique v_π. However, for the A*, the path it finds is not guaranteed to be the shortest path. If there is a reachable path between the origin and the goal, then using the A* algorithm will necessarily result in a reachable path, but not necessarily the shortest path.

And the disadvantage of value iteration is it takes more time to compute compared to A*, because value iteration needs to compute the whole state space while A* only needs to compute part of the state space. So value iteration is less efficient than A*.