Question 3.4

1.(a) BFS

If "I" is the destination node

Depth 1

A -> B

A -> E

A -> F

Depth 2

A -> B -> C

A -> E -> G

A -> F -> H

Depth 3

It is for the A -> B -> C path

A -> B -> C -> D

It is for A -> E -> G

A -> E -> G -> I (if it directly go to I which is the destination and then it will finish the search)

Otherwise

A -> E -> G -> L

It is for A -> F -> H

A -> F -> H -> I (Destination)

Depth 4

A -> E -> G -> L -> I (Destination)

Hence the success path including

1. A -> E -> G -> I

2. A -> F -> H -> I

3.A -> E -> G -> L -> I

It would result in stuck in loop

IF A -> B -> C -> D -> B

If A -> E -> A

If A -> F -> H -> G -> F

If A -> E -> G -> L -> E

If A -> E -> F -> H -> G -> L -> E

If A -> F -> G -> L -> E -> F

1.(b) DFS

Total depth is from 1 to 3

A -> B depth 1

A -> B -> C depth 2

A -> B -> C -> D depth 3

A -> E depth 1

A -> E -> F depth 2

OR A -> E -> G depth 2

IF A -> E -> F

A -> E -> F -> H depth 3

In A -> E -> F -> H, there are two situations

A -> E -> F -> H -> G depth 4

OR A -> E -> F -> H -> I (Destination && depth 4)

IF A -> E -> F -> H -> G, there are three situations

A -> E -> F -> H -> G -> F (get stuck && depth 5)

OR A -> E -> F -> H -> G -> I (Destination && depth 5)

OR A -> E -> F -> H -> G -> L (depth 5)

In A -> E -> F -> H -> G -> L, there are two situations

A -> E -> F -> H -> G -> L -> E (Stuck && depth 6)

A -> E -> F -> H -> G -> L -> I (Destination && depth 6)

If A -> E -> G, there are three situations

A -> E -> G -> L (depth 3)

OR A -> E -> G -> F (depth 3)

OR A -> E -> G -> I (which is the destination && depth 3)

IF A -> E -> G -> L, there are two situations

A -> E -> G -> L -> E (stuck && depth 4)

OR A -> E -> G -> L -> I (Destination && depth 4)

IF A -> E -> G -> F, there are three situations

A -> E -> G -> F -> H (depth 4)

A -> E -> G -> F -> H -> G (Stuck && depth 5)

OR A -> E -> G -> F -> H -> I (Destination && depth 5)

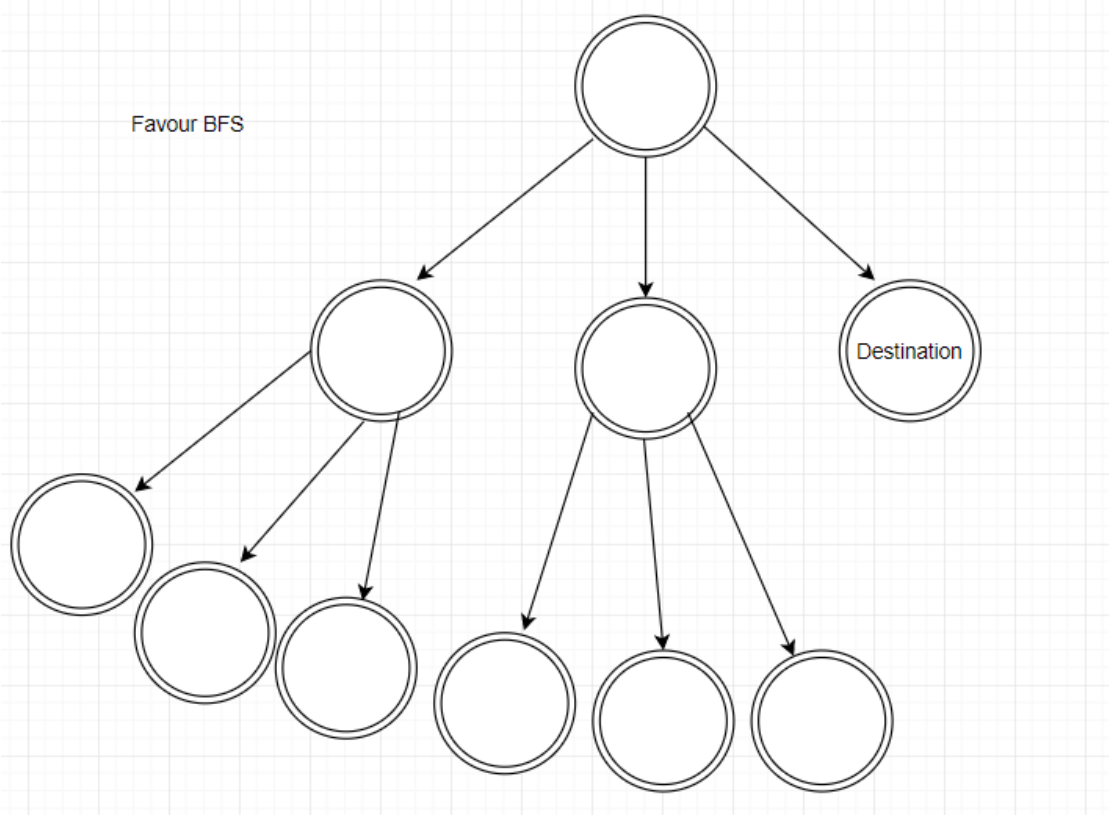
IF A -> F (depth 1)

A -> F -> H, there are two situations (depth 2)

A -> F -> H -> I (Destination && depth 3)
OR A -> F -> H -> G (depth 3)
If A -> F -> H -> G (depth 3)
A -> F -> H -> G -> I (Destination && depth 4)

It would result in stuck in loop
IF A -> B -> C -> D -> B
If A -> E -> A
If A -> F -> H -> G -> F
If A -> E -> G -> L -> E
If A -> E -> F -> H -> G -> L -> E
If A -> F -> G -> L -> E -> F

Question c
It is a very good example to show favours BFS

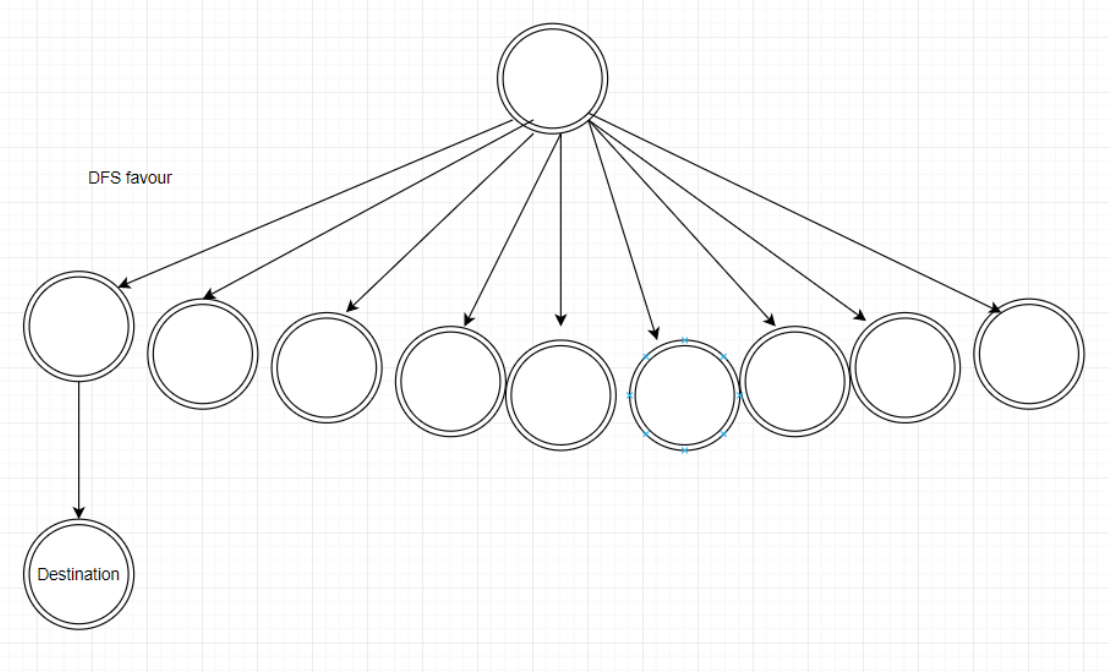


Explanation:
1. In this example, if you use the BFS, it only needs 3 explored nodes to get the target node (Destination). But if you use DFS, it would be 9 explored nodes to get the target node, which is almost triple the workload.
2. Moreover, this graph illustrates the drawbacks of the DFS directly. DFS would spend lots of time and space to explore the first starting node and its branch when the first node and its branches is

deep. But in BFS, the solution is simple and fast if the destination node is short.
3. BFS is optimal algorithm while DFS is not optimal
4. Also, in DFS search, the machine might get stuck in a loop, but not in BFS.

Question d

Favour DFS



Explanation:

1. In this example, you only need to search 2 nodes to get the result you want if you using DFS, but it needs 10 nodes if you using BFS, which is 5 times workload compare with DFS.
2. Memory space is efficiently utilized in DFS while space utilization in BFS is not effective. In this example the BFS uses lots of space and time to search for the destination node, but the DFS is fast and efficiently.
3. BFS is more suitable for searching vertices which are closer to the given source. DFS is more suitable when there are solutions away from source.

Question e

Key Differences Between BFS and DFS

1. BFS is vertex-based algorithm while DFS is an edge-based algorithm.
2. Queue data structure is used in BFS. DFS uses stack or recursion.
3. Memory space is efficiently utilized in DFS while space utilization in BFS is not effective.
4. BFS is optimal algorithm while DFS is not optimal.
5. DFS constructs narrow and long trees. As against, BFS constructs wide and short tree.
6. BFS and DFS, both of the graph searching techniques have similar running time but different space consumption, DFS takes linear space because we have to remember single path with unexplored nodes, while BFS keeps every node in memory.
7. DFS yields deeper solutions and is not optimal, but it works well when the solution is dense whereas BFS is optimal which searches the optimal goal at first.
8. The major difference between BFS and DFS is that BFS proceeds level by level while DFS follows first a path from the starting to the ending node (vertex), then another path from the start to end, and so on until all nodes are visited. Furthermore, BFS uses the queue for storing the nodes whereas DFS uses the stack for traversal of the nodes.

2.(a)

(a): optimal depth is 3, which would get an optimal solution that A -> E -> G -> I

(b)

1.The advantage of Iterative deepening search (IDS) is proceeds level by level and also deep into a path, and it is not easily to get stuck during the search. The DFS space complexity is less and it follows first a path from the starting to the ending node (vertex).

2.I tend to use the IDS if the depth of some node in the graph is very deep, because if I use the IDS which would reduce the probability to get stuck and increase the performance that shown in Question 1(c).

3. Also because the IDS would gradually increase the scale of the depth, then until it finds the target node, which means that IDS is a complete and optimal algorithm.

4. It would largely decrease the unnecessary and useless search for the unimportant node

Question 4.3

1.(a)

1.Straight paths typically look more plausible than jagged paths, particularly through open spaces

2.If each waypoint looks ahead to farthest unobstructed waypoint on the path which would result in more connections in the waypoint graph and would increase cost.

3.Bias the search toward straight paths, it would increase the cost for a segment if using it requires turning a corner. And it would reduce efficiency, because straight but unsuccessful paths will be explored preferentially.

.1.(b)

A* is often used for the common pathfinding problem in applications such as video games, google map and air route planning. The valid heuristic including

2.(a)

1. Greedy best first algorithms expand most desirable unexpanded node, usually the node with the lowest evaluation. A* search avoid expanding paths that are already expensive
2. Greedy best first algorithm is not complete (can get stuck in loops). A* search is complete
3. Greedy best first algorithm time complexity is O(b^m) for tree version, but a good heuristic can give dramatic improvement. A* search time complexity is exponential.
4. Greedy best first algorithm is not optimal, A* search is optimal
5. The greedy BFS algorithm, the evaluation function is f(n)=h(n)f(n)=h(n), that is, the greedy BFS algorithm first expands the node whose estimated distance to the goal is the smallest. So, greedy BFS does not use the "past knowledge",
6. In the case of the A* algorithm, the evaluation function is f(n)=g(n)+h(n), where is an admissible heuristic function. The "star", often denoted by an asterisk, *, refers to the fact that A* uses an admissible heuristic function, which essentially means that A* is optimal,
7. In summary, greedy BFS is not complete, not optimal, has a time complexity of O(b^m) and a space complexity which can be polynomial. A* is complete, optimal, and it has a time and space complexity of O(b^m). So, in general, A* uses more memory than greedy BFS. A* becomes impractical when the search space is huge. However, A* also guarantees that the found path between the starting node and the goal node is the optimal one and that the algorithm eventually terminates. Greedy BFS, on the other hand, uses less memory, but does not provide the optimality and completeness guarantees of A*. So, which algorithm is the "best" depends on the context, but both are "best"-first searches.

2.(b)

Solution:

1.Using f(n) = h(n) instead of f(n) = g(n) + h(n) in A* algorithm.

2. Because the greedy BFS evaluation function is f(n)=h(n), and A* algorithm is f(n)=g(n) +h(n). If we want to change A* into greedy BFS, then just delete the g(n) in the f(n) = g(n) + h(n), which means that the algorithm just need to evaluate which paths with lowest cost then go to it, instead of evaluating and predict the cost of each path.

Question 5.4

1. Yes, I am
2. Because there are 16 different slots, 8 orientation, hence the total number of actions is 8*16*8 = 1024 during one term.
3.

Main challenge:

1. The alpha-beta pruning method is not feasible in most of the cases to find out the whole game tree, a depth limit needs to be set.
2. Evaluations of the utility of a node are usually not exact but crude estimates of the value of a position and as a result large errors could be associated with them.
3. Alpha-Beta is designed to select a good move but it also calculates the values of all legal moves. A better method maybe to use what is called the utility of a node expansion. In this way a good search algorithm could select a node that had a high utility to expand (these will hopefully lead to better moves). This could lead to a faster decision by searching through a smaller decision space. An extension on those abilities would be the use of another technique called goal-directed reasoning. This technique focuses on having a certain goal in mind like capturing the queen in chess. So far no one has successfully combined these techniques into a fully functional system.

Whether the implementation is practical?

1. No, the time cost is too large. It would take a long time to run.
2. For example, in first turn, there are 1024 different actions could be chosen, the second turn is 960 actions and so on.
3. But in my view, it is a way that better than the minimax algorithms.