LAB 2

1. In the vacuum cleaner domain in part 1, what were the states and actions? What is the branching factor?

   There exist mainly *three states* in the vacuum cleaner domain in part 1: the *initial state,* the *goal state* and in-between states called *problem states*. The different actions performed are:

   - No Operation
   - Suck Dirt
   - Go North
   - Go South
   - Go East
   - Go West

   Branching Factor

   In computing, tree data structures, and game theory, the branching factor is the number of children at each node, the outdegree, if this value is not uniform, an average branching factor can be calculated. The higher the branching factor, the faster this "explosion" occurs. For the problem given the branching factor is 4.

2. What is the difference between Breadth First Search and Uniform Cost Search in a domain where the cost of action is 1?

   Breadth First Search is *optimal* if all the step costs are the same. For any step-cost function, Uniform Cost Search expands the node with *least path cost*. Other than the ordering of queue there is one more important difference between Breadth First Search and Uniform Cost Search. In Breadth First Search, the goal test on the node is performed when it is *first generated*. But in Uniform Cost Search, goal test on the node is performed when it is *selected for expansion*. This is because the first node generated could be a sub optimal-path. Breadth First Search is *complete* but Uniform Cost Search completeness is guaranteed only if *the cost of every step is some positive number*. But *when the cost is 1 there is no difference* between uniform cost search and Breadth first search.

3. Suppose that h1 and h2 are admissible heuristics (used in for example A*). Which of the following are also admissible?
   a) (h1+h2)/2
   b) 2h1
   c) max (h1, h2)

Max (h1, h2) and (h1+h2)/2 are admissible because taking the maximum of admissible heuristics is again admissible and also as h1 and h2 are admissible the average of them cannot be more than h1 or h2.

4. If one uses A* to search for a path to one specific square in the vacuum domain, what could the heuristic function (h) be? What could the cost function (g) be? Is your choice of (h) an admissible heuristic? Explain why.

The heuristic function (h) tells A* an estimate of the minimum cost from any vertex n to the goal. A heuristic function that can be used is the *straight line distance between nodes* as this cannot be greater than the actual distance to the node.

A cost function tells the cost or expense in moving from the initial state to the goal state. A cost function that can be used here is the number of squares covered in moving from initial to goal state.

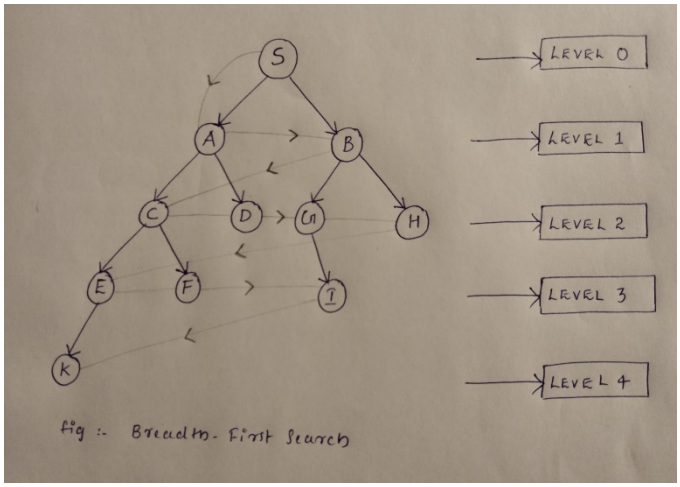
5. Draw and explain. Choose your three favorite search algorithms and apply them to any problem domain (it might be a good idea to use a domain where you can identify a good heuristic function). Draw the search tree for them, and explain how they proceed in the searching. Also include the memory usage. You can attach a hand-made drawing.

Breadth First Search

- Breadth First Search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called Breadth First Search.
- Breadth First Search algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The Breadth First Search algorithm is an example of a general-graph search algorithm.
- Breadth First Search implemented using FIFO queue data structure.

Example:

In the below tree structure, we have shown the traversing of the tree using Breadth First Search Algorithm form the root node S to goal node K. Breadth First Search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:



S --- >A --- >B --- >C --- >D --- >G --- >H --- >E --- >F --- >I --- >K

Time Complexity: Time complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.....+ b^d = O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

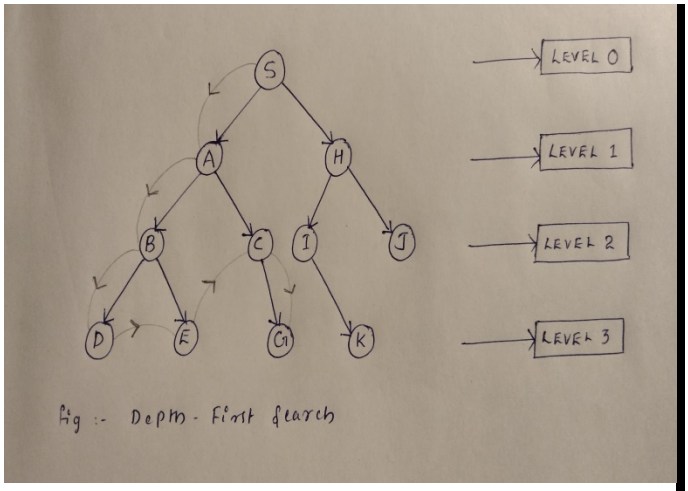
Depth First Search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Example:

In the below search tree, we have shown the flow of Depth First Search and it will follow the order as:

Root node --- >Left node --- >Right node



It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n) = 1 + n^2 + n^3 + \ldots\ldots\ldots + n^m = O(n^m)$

Where m = maximum depth of any node and this can be much larger than d (shallowest solution depth).

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is O (bm).

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cose to reach to the goal node.

Uniform Cost Search Algorithm

Uniform cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform cost search expands nodes according to their path costs from the root node. It can be used to solve any graph or tree where the optimal cost is in demand. A uniform cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

Example:

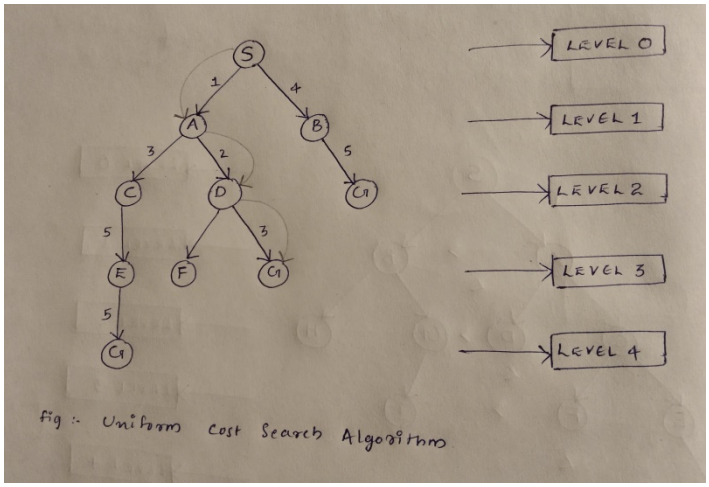


fig :- Uniform cost Search Algorithm

Completeness: Uniform cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity: Let C* is the cost of the optimal solution and ε is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε. Hence the worst-case time complexity of Uniform Cost Search is **O ($b^{1 + \lfloor C^*/\varepsilon \rfloor}$)/**.

Space Complexity: The same logic is for space complexity so the worst-case complexity of Uniform Cost Search is **O ($b^{1 + \lfloor C^*/\varepsilon \rfloor}$)**.

Optimal: Uniform Cost Search is always optimal as it only selects a path with the lowest path cost.

6. Look at all the offline search algorithms presented in chapter 3 plus A* search (i.e. Best-first search, Breadth-first search, Uniform-cost search, Depth-first search, Iterative deepening search, Bidirectional search, Greedy best-first search and A* search). Are they complete? Are they optimal? Explain why!

Breadth First Search

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Uniform Cost Search

Completeness: Uniform cost search is complete, such as if there is a solution, UCS will find it.

Optimal: Uniform Cost Search is always optimal as it only selects a path with the lowest path cost

Depth First Search

Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

*Optimal***:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cose to reach to the goal node.

Iterative Deepening Search

Completeness: This algorithm is complete if the branching factor is finite.

Optimal: Iterative Deepening Search Algorithm is optimal if path cost is a non-decreasing function of the depth of the node.

Bidirectional Search

Completeness: Bidirectional Search is complete if we use BFS in both searches.

Optimal: Bidirectional Search is optimal.

Greedy Best First Search

Completeness: Greedy Best First Search is incomplete, even if the given state space is finite.

Optimal: Greedy Best First Search Algorithm is not optimal.

A* Search

Completeness: A* Algorithm is complete as long as:

- Branching factor is finite
- Cost at every action is fixed

Optimal: A* Search Algorithm is optimal if it follows below two conditions:

- Admissible: the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- Consistency: second required condition is consistency for only A* graph search.

  If the heuristic function is admissible, then A* tree search will always find the least cost path.

7. Assume that you had to go back and do Lab 1/Task 2 once more (if you did not use search already). Remember that the agent did not have perfect knowledge of the environment but had to explore it incrementally. Which of the search algorithms you have learned would be most suited in this situation to guide the agent's execution? What would you search for? Give an example.

A* search with the admissible heuristics of straight line distance would have been optimal for find the path from state to goal nodes as this would have found the cost optimal path by exploring the least number of nodes.