Name - Shreya Panikkassery
Roll no - 9582
TE Comps A

Exp5 - Postlab.

1.) Explain the time complexity of the A* Algorithm.

→ The time complexity of A* depends on the quality of the heuristic function. In a worst case, the algorithm can be $O(b^d)$. Where b is the branching factor - the average number of edges from each node, and d is the number of nodes on the resulting path.

2.) What are the limitations of A* Algorithm?

→ The A* algorithm, while renowned for its efficiency, has certain limitations to consider:

a) Computational cost:

A* can be computationally expensive, especially in scenarios with:

Expensive search spaces: When dealing with a vast number of possible paths, the exploration process can become resource-intensive.

High branching factors - If each node in the search space has many potential neighbors, the algorithm needs to evaluate numerous options, increasing computation time.

b) Reliance on Heuristics:

A* heavily relies on the quality of the heuristic used to estimate the distance to the goal.

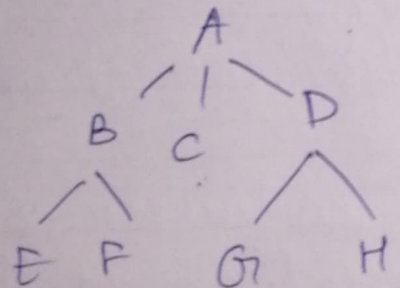
Poor heuristic: A poorly designed heuristic function used to estimate the distance to the goal.
Domain specific: Designing an effective heuristic often requires significant domain knowledge, making it less versatile for problems with diverse characteristics.

c) Limited Applicability:
A* may struggle with specific types of search spaces.
Dynamic environments: A* assumes a static environment where the cost of moving between nodes remains constant. This can be problematic in real-world scenarios with dynamic elments.

3) Discuss A*, BFS, DFS, Dijkstra's Algorithm, or traversing.

BFS –
1) Algorithm: for searching tree or graph data structure
2) It starts at the root, explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.
3) Uses queue data structure.

example.

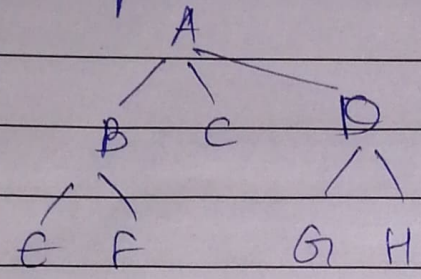


Traversing order:

A → B → C → D → E → F → G → H

DFS:
→ Another algorithm for traversing or searching
→ It starts at root, explores as far as possible along each branch before backtracking.
→ Uses stack data structure.

Example:



Traversing order:

A → B → E → F → C → D → G → H

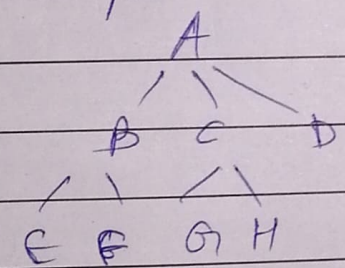
Dijkstra's Algorithm.

→ It is used to find the shortest path from single source vertex to all other vertices in a weighted graph.

→ It maintains a set of vertices whose shortest distance from source is known.

→ It iteratively selects the vertex with smallest tentative distance from the source vertex and updates the distances

Example:



If we apply with source node A, it would compute the shortest path.

A* Algorithm:

→ Used to find the shortest path in a graph.

→ Uses heuristic to estimate the cost from the current node to the goal, combined with the actual cost of reaching that node from the start.

A* keeps track of the total estimated cost of reaching a node (f-~~node~~ score) where

fscore = hscore + g-score
(heuristic estimate of current to goal) (cost of current node)

example:
We have grid map, where each cell represents a node in the graph, we need to find the shortest path from the start cell to the target cell. A* algorithm can be applied.