U20CS005

Tutorial-6

PAGE NO.
DATE / /

1. "Depth First Search uses LIFO technique'. Justify the statement. Also provide the reason how backtracking will help improving space complexity in DFS?

→ - The DFS algorithm starts with root node and explores the child node of current node, thus it expands the deepest node of the current froniter ~~of stacks~~ before backtracking. To implement this, the algorithm uses a stack data structure to keep track of the nodes visited and the nodes that need to be visited. The last node added to the stack is the first one to explored, hence DFS uses LIFO approach.

- Backtracking will help improving the space complexity of DFS because when DFS visits a node and its adjacent nodes, it marks the visited nodes in a visited array or sets a flag in the node itself. However, this approach can lead to high space complexity when the graph is large as all visited nodes have to be stored in memory.

Backtracking is a technique that involves undoing the ~~same~~ changes made to the state of the algorithm when it reaches a dead end. In case of DFS, this means removing the node from stack & making it an unvisited. By doing this, the algorithm can free up by removing nodes from the visited ~~sta~~ set that are no longer needed.

2. What do you mean by diameter of a state space search? In which uninformed search it is used? Explain it in detail.

PAGE NO.
DATE / /

→ - The diameter of a state space search is the length of the longest path between any starting state to states in the search space.

- Depth limited search algorithm uses diameter. It restricts the depth of the search, stopping the algorithm from going further than the set limit. All nodes at depth 'd' are recognised as that they do not have any successors. This algorithm is similar to DFS but DLS runs only till the diameter specified in the algorithm as DFS may run into problem known as the DFS bottleneck where it gets stuck exploring one branch of search tree & does not explore other branches, if the diameter is very large. Therefore, the diameter of a state space search is an important parameter to consider while using DFS i.e DLS if diameter constraint is provided in algorithm.

3. Compare & contrast the following algorithm. Also comment on their space complexity.
a) Breadth first search & Uniform Cost Search
b) IDA* and RBFS.

→ a) BFS & UCS

- BFS and UCS both are the search algorithm and are category of uninformed search. Both algorithm uses queue data structure and explores all successing node breadth wise. The main difference between these algorithms is the way they explore the search space at each depth level.
- BFS explores the search space in layers. It

starts at the root node & visit all the nodes at the same depth before moving on to the next level. BFS is guaranteed to find the shortest path to the goal node. ~~If the pa~~ BFS stores all the visited nodes in a ~~priority~~ queue, which can be memory intensive, especially for large search spaces. The space complexity of BFS is $O(b^d)$.

- UCS on other hand, explores the search space by considering the cost of the path to each node. It always expands the node with the lowest path cost. UCS ~~all~~ also guaranteed to find the shortest path to the goal node. UCS also stores all visited node but in priority queue, thus the space complexity can be improved by using data structure like heap. Space complexity of UCS is $O(b^{1+\lceil C/\varepsilon \rceil})$.

b) IDA* & RBFS

- IDA* & RBFS both are the informed search algorithm used to find the optimal path from the start node to the goal node.
- IDA* is the modified version of A* that does not use a priority queue. Instead, it uses an iterative deepening strategy to limit the search space. It starts with a threshold & expands the node within that threshold. If it does not find a solution, it increases the threshold & repeats the process until a solution is found. IDA* is memory-efficient as it only stores the current path & threshold, & its space complexity is $O(bd)$.
- RBFS is a variant of BFS that uses a limited amount of memory. It explores the search space by expanding the node with the lowest

heuristic value & if it encounters a node with higher heuristic value, it temporarily stores the current state & explores the new node. Once it has finished exploring new node, it returns to the previous state & continues the search. RBFS also has a space complexity of O(bd) as it stores the current path & a limited number of nodes in memory.