Experiment No. __2_

Title: : Implementation of Uninformed search algorithm – BFS and DFS

Batch: **Roll No.:** **Experiment No.: 2**

Aim: To implement BFS and DFS - Uninformed search algorithm in state space

Resources needed: C / C++ / Java / Python

Theory

Intelligent agents are supposed to maximize the performance measure. The problem solving agents start with the activity of goal formulation where, it organizes the behavior by limiting the objectives that the agent is trying to achieve. Then comes the problem formulation which the process of deciding what actions and state to consider, given a goal. These (legal) actions when applied to initial state, gives us the entire state-space.

The state-space leaves the agent with several immediate options of unknown value where the agent can decide what to do next by first examining different possible sequences of actions that lead to states of known value, and then choosing the best one.

The algorithms in uninformed search category:

BFS:

These algorithms search trees of nodes, whether that tree is explicit or implicit (generated on the go). The basic principle is that a node is taken from a data structure, its successors examined and added to the data structure. By manipulating the data structure, the tree is explored in level by level order.

This method selects the deepest unexpanded node in the search tree for expansion.

Algorithm :

1. Enqueue the root node.
2. Dequeue a node and examine it.
   - If the element sought is found in this node, quit the search and return a result.
   - Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
4. If the queue is not empty, repeat from Step 2

DFS:

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking. Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

In depth-first search, the frontier acts like a last-in first-out stack. The elements are added to the stack one at a time. The one selected and taken off the frontier at any time is the last element that was added.

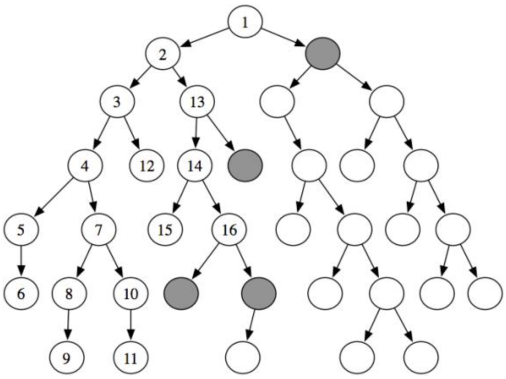

Figure – Depth First Search

Algorithm

1. If the initial state is a goal state, quit and return success.
2. Otherwise, loop until success or failure is signalled.
   - Generate a state, say E, and let it be the successor of the initial state. If there is no successor, signal failure.
   - Call Depth-First Search with E as the initial state.
   - If success is returned, signal success. Otherwise continue in this loop.

The advantage of depth-first Search is that memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.

The depth-first search is time-limited rather than space-limited.

Procedure:

1. Implement Mentioned algorithm for BFS for graph search
2. Implement the mentioned algorithm for DFS for graph search
3. Output must show the contents of Fringe and Visited nodes for each iteration of graph traversal. Also, finally it must print the path traversed.

Results: (Softcopy submission of Summary Document)

Outcomes:

Conclusion:

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

- Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Second Edition, Pearson Publication
- Elaine Rich, Kevin Knight, Artificial Intelligence, Tata McGraw Hill, 1999.