Artificial Intelligence

ASSIGNMENT #1

Eisha Baig | 200901015 | Bscs-01(B)

Objective:

To understand the basics of search algorithms and implement a simple search algorithm in Python.

Source: Arad

Destination: Bucharest

TASK:

1.Research the Depth-First Search (DFS) & Breadth First Search (BFS)Algorithms.2.Apply DFS and BFS on Romanian example3.The program should take as input a graph and represents an adjacency list along withsource and goal nodes.4.The program should output the shortest path or optimize path from source to the goalnode.5.Also highlight which of the algorithm outperform other.

WORKING/PROCEDURE:

DFS

1. def dfs_shortest_path(graph, start, end, path=none, shortest_path=none): defines a function dfs_shortest_path that takes four arguments: graph - the graph to be searched, start - the starting node, end - the target node, path - the current path being searched (defaulted to none), and shortest_path - the current shortest path found so far (defaulted to none).
2. checks if path is none. if path is none, then the current start node is the first node of the path.
3. checks if shortest_path is none. if shortest_path is none, then no shortest path has been found yet.
4. if start == end: checks if the current start node is equal to the end node. if the start node is equal to the end node, then we have found a path from start to end.
5. if not shortest_path or len(path) < len(shortest_path): checks the function checks if the path from the start node to the end node (i.e., the path currently being explored by the algorithm) is shorter than the current shortest path that has been found so far. If this is true, then the current path is stored as the new shortest path.else: if the current start node is not equal to the end node, then we need to search for the target node by recursively.
6. for neighbor in graph[start]: iterates through each neighbor of the current start node.
7. if neighbor not in path: checks if the current neighbor node is not already in the current path.

BFS

1. queue = [(start, [start])]: A priority queue is created, this queue will be used to keep track of the nodes to visit next in the order of their priority.
2. visited = set(): An empty set is created to keep track of the nodes that have been visited.
3. while queue:: A while loop is started which will keep running as long as there are nodes in the queue.
4. node, path = heapq.heappop(queue): The node and its path with the minimum priority are extracted from the queue using heappop() method from the heapq module. This ensures that the nodes with the smallest path are visited first.
5. if node == goal:: If the node being visited is the goal node, the path to the goal is returned.
6. visited.add(node): If the node is not the goal node, it is marked as visited by adding it to the visited set.
7. for neighbor in graph[node]:: For each neighbor of the current node that has not been visited yet, we do the following:
8. if neighbor not in visited:: If the neighbor has not been visited yet, we add it to the queue with its path updated to include the current neighbor.
9. heapq.heappush(queue, (neighbor, path + [neighbor])): The neighbor is added to the queue with its priority based on the length of the current path to it. The heappush() method from the heapq module ensures that the nodes are inserted into the queue in the order of their priority.
10. At the end of the loop, if the goal node is not found in the queue, it means that there is no path from the start to the goal, so None is returned.

RESULT:

```
The shortest path from Arad to Bucharest using DFS is: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
The shortest path from Arad to Bucharest using BFS is: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
```