NAME : Jigar Shailesh Siddhpura

SAP ID : 60004210155

DIV: B 2

BRANCH : COMPS

EXPERIMENT 2 : Identify and analyze uninformed search Algorithm to solve the problem.

Implement BFS/DFS search algorithms to reach goal state.

Theory :

BFS:

Breadth-first search is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. BFS or Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighbouring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.
There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Algorithm:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS =1) and set their STATUS = 2

(waiting state)

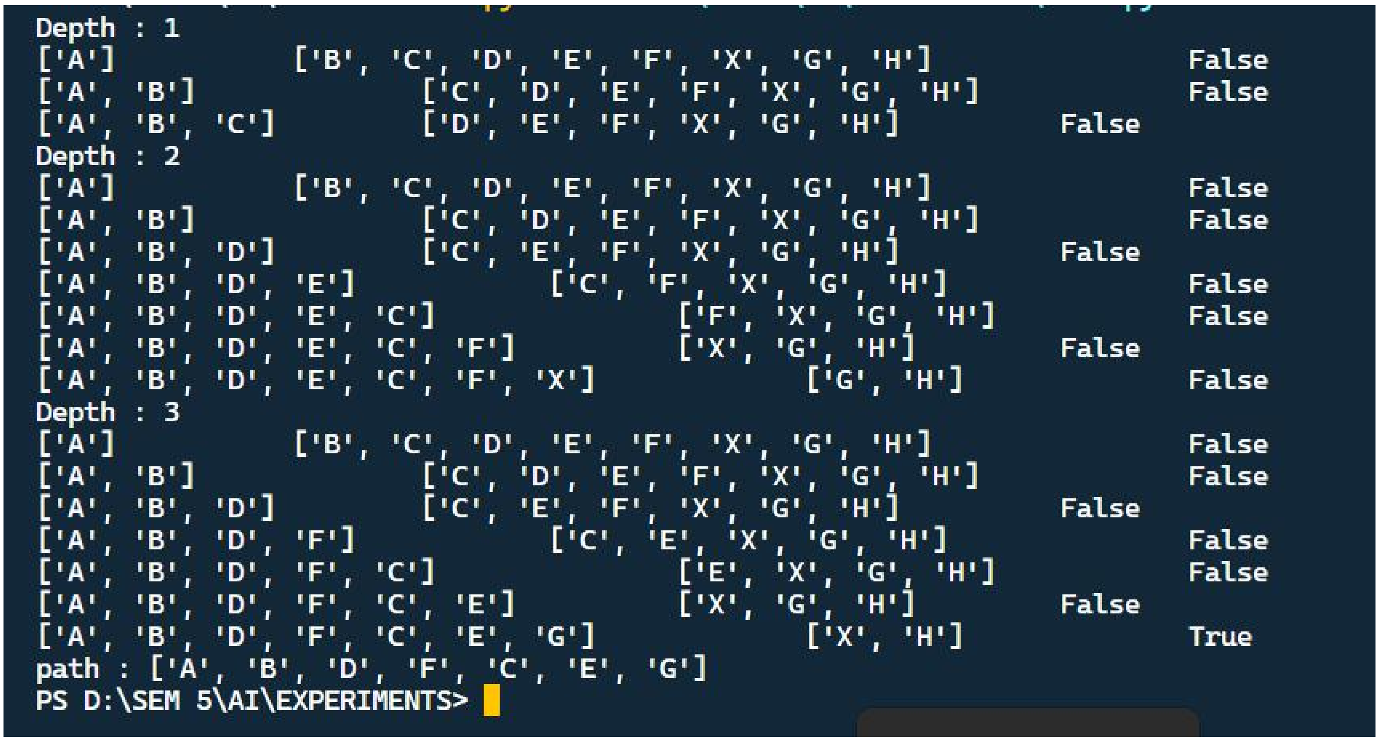
[END OF LOOP]

Step 6: EXIT

Code :

```python
graph = {
 'A': ['B', 'C'],
 'B': ['A', 'D'],
 'C': ['A', 'D', 'E'],
 'D': ['B', 'C', 'F'],
 'E': ['C', 'F', 'G'],
 'F': ['D', 'E', 'H'],
 'G': ['E', 'H'],
 'H': ['F', 'G']
}

def bfs(graph, current_node, goal_node, visited, path, queue):
    visited.append(current_node)
    queue.append(current_node)
    not_visited = [i for i in graph.keys() if i not in visited]
    print(f"{visited}\t\t{not_visited}\t\tFalse")
    while queue:
        node = queue.pop(0)

        path.append(node)
        if node == goal_node:
            not_visited = [i for i in graph.keys() if i not in visited]
            print(f"{visited}\t\t{not_visited}\t\tTrue")
            print(f"path : {path}")
            break

        for neighbour_node in graph[node]:
            if neighbour_node not in visited:
                visited.append(neighbour_node)
                queue.append(neighbour_node)
                not_visited = [i for i in graph.keys() if i not in visited]
                print(f"{visited}\t\t{not_visited}\t\tFalse")

print("\n")
print(f"Nodes in Graph : {[i for i in graph.keys()]}")
start_node = input("Enter start node : ")
goal_node = input("Enter goal node : ")
visited = []
queue = []
path = []
print("<------------------------------Breadth-First Search------------------------------>")
print("Visited\t\tNot Visted\t\tGoal state")
print("<------------------------------------------------------------------------------>")
bfs(graph, start_node, goal_node,visited,path,queue)
```

Output:



```
Depth : 1
['A']           ['B', 'C', 'D', 'E', 'F', 'X', 'G', 'H']                False
['A', 'B']              ['C', 'D', 'E', 'F', 'X', 'G', 'H']             False
['A', 'B', 'C']         ['D', 'E', 'F', 'X', 'G', 'H']          False
Depth : 2
['A']           ['B', 'C', 'D', 'E', 'F', 'X', 'G', 'H']                False
['A', 'B']              ['C', 'D', 'E', 'F', 'X', 'G', 'H']             False
['A', 'B', 'D']         ['C', 'E', 'F', 'X', 'G', 'H']          False
['A', 'B', 'D', 'E']            ['C', 'F', 'X', 'G', 'H']               False
['A', 'B', 'D', 'E', 'C']               ['F', 'X', 'G', 'H']            False
['A', 'B', 'D', 'E', 'C', 'F']          ['X', 'G', 'H']         False
['A', 'B', 'D', 'E', 'C', 'F', 'X']             ['G', 'H']              False
Depth : 3
['A']           ['B', 'C', 'D', 'E', 'F', 'X', 'G', 'H']                False
['A', 'B']              ['C', 'D', 'E', 'F', 'X', 'G', 'H']             False
['A', 'B', 'D']         ['C', 'E', 'F', 'X', 'G', 'H']          False
['A', 'B', 'D', 'F']            ['C', 'E', 'X', 'G', 'H']               False
['A', 'B', 'D', 'F', 'C']               ['E', 'X', 'G', 'H']            False
['A', 'B', 'D', 'F', 'C', 'E']          ['X', 'G', 'H']         False
['A', 'B', 'D', 'F', 'C', 'E', 'G']             ['X', 'H']              True
path : ['A', 'B', 'D', 'F', 'C', 'E', 'G']
PS D:\SEM 5\AI\EXPERIMENTS>
```

Theory :

DFS:

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking
It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children. Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

Algorithm:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

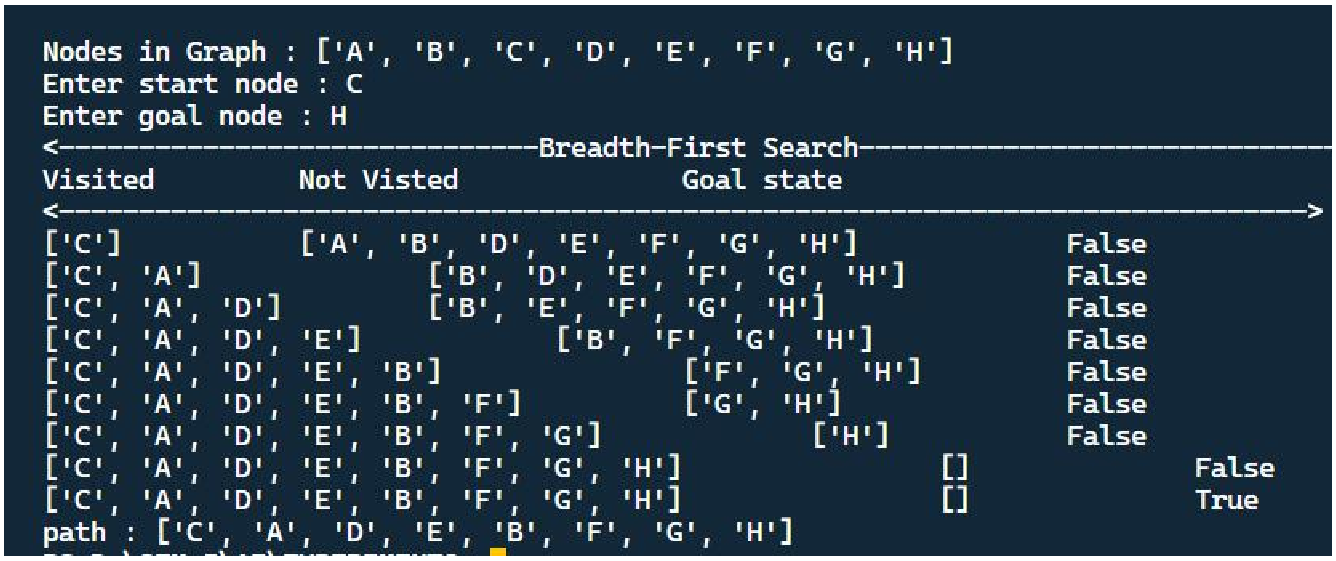
[END OF LOOP]

Step 6: EXIT

Code:

```
graph = {
  'A': ['B', 'C'],
  'B': ['A', 'D'],
  'C': ['A', 'D', 'E'],
  'D': ['F', 'C', 'B'],
  'E': ['C', 'F', 'G'],
  'F': ['D', 'E', 'H'],
  'G': ['E', 'H'],
  'H': ['F', 'G']
}

def dfs(graph, current_node, goal_node, visited, path):
    if current_node not in visited:
        path.append(current_node)
        visited.append(current_node)
        not_visited = [i for i in graph.keys() if i not in visited]
        if current_node == goal_node:
            print(f"{visited}\t\t{not_visited}\t\tTrue")
            print(f"path : {path}")
            exit()

        print(f"{visited}\t\t{not_visited}\t\tFalse")
        for i in graph[current_node]:
            dfs(graph, i, goal_node, visited, path)
print("\n")
print(f"Nodes in Graph : {[i for i in graph.keys()]}")
start_node = input("Enter start node : ")
goal_node = input("Enter goal node : ")
visited = []
path = []
print("<-----------------------------Depth-First Search----------------------------->")
print("Visited\t\tNot Visted\t\tGoal state")
print("<----------------------------------------------------------------------------->")
dfs(graph, start_node, goal_node,visited,path)
```

Output:



```
Nodes in Graph : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
Enter start node : C
Enter goal node : H
<-------------------------------Breadth-First Search-------------------------------
Visited         Not Visted              Goal state
<-------------------------------------------------------------------------------->
['C']           ['A', 'B', 'D', 'E', 'F', 'G', 'H']             False
['C', 'A']              ['B', 'D', 'E', 'F', 'G', 'H']          False
['C', 'A', 'D']         ['B', 'E', 'F', 'G', 'H']               False
['C', 'A', 'D', 'E']            ['B', 'F', 'G', 'H']            False
['C', 'A', 'D', 'E', 'B']               ['F', 'G', 'H']         False
['C', 'A', 'D', 'E', 'B', 'F']          ['G', 'H']              False
['C', 'A', 'D', 'E', 'B', 'F', 'G']             ['H']           False
['C', 'A', 'D', 'E', 'B', 'F', 'G', 'H']                []              False
['C', 'A', 'D', 'E', 'B', 'F', 'G', 'H']                []              True
path : ['C', 'A', 'D', 'E', 'B', 'F', 'G', 'H']
```

Conclusion:

Hence successfully studied Uninformed Searching Algorithms like BFS,DFS for finding the path between the start and goal node.