Ai Application & Ethics(TC-7)

Lab 1: - Program to implement iterative deepening DFS and BFS

Name: - Utkarsh Bhangale

PRN: - 20200802124

DS1

1.DFS

Iterative Deepening Depth-First Search (DFS) is a graph traversal algorithm that combines the benefits of depth-first search (DFS) with a breadth-first search (BFS)-like search strategy. It explores nodes at increasing depths, starting from depth 0 and incrementing the depth limit in each iteration until the goal is found or all nodes have been explored. IDDFS ensures that it explores nodes in the same order as BFS but uses less memory.

Algorithm

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited The purpose of the algorithm is to mark each vertex as visited while avoiding cycles. The DFS algorithm works as follows:
3. Start by putting any one of the graph's vertices on top of a stack.
4. Take the top item of the stack and add it to the visited list.
5. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
6. Keep repeating steps 2 and 3 until the stack is empty.

In [1]:
```
#Program to implement iterative deepening DFS
# Using a Python dictionary to act as an adjacency list
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set() # Set to keep track of visited nodes of graph.

ted, graph, node):  #function for dfs
```

Loading [MathJax]/extensions/Safe.js

```
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

```
Following is the Depth-First Search
5
3
2
4
8
7
```

2.BFS

Breadth-First Search (BFS) is a graph traversal algorithm used to systematically explore and visit all the nodes or vertices of a graph in a level-by-level manner, starting from a specified source node. It begins by visiting the source node, then explores all its neighbors before moving on to their neighbors, effectively traversing the graph in breadth-first order. BFS is commonly employed in various applications, including shortest path calculations and searching for connected components in a graph, and it ensures that it finds the shortest path in unweighted graphs.

Algorithm

1. SET STATUS = 1 (ready state) for each node in G
2. Enqueue the starting node A and set its STATUS = 2 (waiting state)
3. Repeat Steps 4 and 5 until QUEUE is empty
4. Dequeue a node N. Process it and set its STATUS = 3 (Processed state).
5. Enqueue all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2

(Waiting state)

[END OF LOOP]

In [2]:
```
#Program to implement iterative deepening BFS

graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue
```

Loading [MathJax]/extensions/Safe.js

```
def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

```
Following is the Breadth-First Search
5 3 7 2 4 8
```

Counclusion

We have successfully implemented iterative deepning BFS & DFS.