ARTIFICIAL INTELLIGENCE (18CSC305J) LAB

EXPERIMENT 3

Implementation and analysis of DFS & BFS

Harsh Goel
RA1811003010185
CSE-C1

Aim:

To implement DFS and BFS algorithm for an application and analyze it. [Traversing and Searching]

Problem Description:

Breath-first search (BFS):

Breath-first search (BFS) is an algorithm used for tree traversal on graphs or tree data structures. BFS can be easily implemented using recursion and data structures like dictionaries and lists.

Depth-first search (DFS):

Depth-first search (DFS), is an algorithm for tree traversal on graph or tree data structures. It can be implemented easily using recursion and data structures like dictionaries and sets

Problem Formulation:

The rules are as follows:

BFS:

- Pick any node, visit the adjacent un-visited vertex, mark it as visited, display it, and insert it in a queue.
- If there are no remaining adjacent vertices left, remove the first vertex from the queue.
- Repeat step 1 and step 2 until the queue is empty or the desired node is found.

DFS:

- Pick any node.
- If it is un-visited, mark it as visited and recur on all its adjacent nodes.
- Repeat until all the nodes are visited, or the node to be searched is found.

Algorithm:

For both problems, inputting of graph is the same so it is explained separately.

Inputting the Graph:

1. Graph will be in the form of node & its adjacent nodes, in form of a dictionary with nodes as keys & adjacent node list as data value.
2. Input the number of nodes to be used.
3. Initialize the Dictionary.
4. Using a for loop input n keys & n adjacency node lists and assign them.
5. Return the resultant dictionary.

BFS:

1. Initialize a visited node list, to keep track of the visited nodes.
2. Function parameters will be a visited node list, inputted graph, the current node, a list to note the BFS traversal.
3. Add the current note to the visited list & update the queue as well.
4. Using a while loop traverse down the queue, take the first index of the queue & add it to the list, thereby traversing it.
5. Using the for loop explore the adjacent nodes.
6. If the neighbor is not visited the add it to the visited list & update the queue.
7. Repeat the procedure from step 3, till the queue becomes null & all the nodes are traversed
8. Return the BFS traversed list.
9. Input the node to searched & check it against the list returned from the above function & print the appropriate message.

DFS:

1. Initialize a visited node set, to keep track of the visited nodes.
2. Function parameters will be a visited node set, inputted graph, the current node, a list to note the DFS traversal.
3. If the current node is not in visited, add the current note to the visited set & update the list as well.
4. Using the for loop explore the adjacent nodes.
5. Recursively call the DFS traversal function & repeat the procedure from step 2 till all of graph is traversed.
6. Using if condition check whether the visited set has all the nodes & return the DFS traversal list.
7. Input the node to searched & check it against the list returned from the above function & print the appropriate message

Source Code:

Language-**PYTHON**

DFS:

```
def enter_graph():
    n = int(input("No of Nodes: "))
    g = {}
    for i in range(n):
        node = input("Node " + str(i) + " -> ")
        l = list(input("Adjacency nodes for node " + node + "->").split(","))
        if l == ['']:
            l = []
        g[node] = l
    print("Entered Graph:-> " + str(g))
    return g

def dfs_trav(visited, graph, node, l):
    if node not in visited:
        l.append(node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs_trav(visited, graph, neighbour, l)
    if len(visited) == len(graph.keys()):
        return l


graph = enter_graph()
visited = set()
start_node = list(graph.keys())[0]
l = dfs_trav(visited, graph, start_node, l=[])
print("DFS traversal of the graph:", str(l))
ser = input("Enter Node to be searched -> ")
if ser in l:
    print("Node exists in the given graph.")
else:
    print("Node doesn't exist in the given graph.")
```

BFS:

```
def enter_graph():
   n = int(input("No of Nodes: "))
   g = {}
   for i in range(n):
      node = input("Node " + str(i) + " -> ")
      l = list(input("Adjacency nodes for node " + node + "->").split(","))
      if l == ['']:
         l = []
      g[node] = l
   print("Entered Graph:-> " + str(g))
   return g

def bfs_trav(visited, graph, node, l):
   visited.append(node)
   queue.append(node)
   while queue:
      s = queue.pop(0)
      l.append(s)
      for neighbour in graph[s]:
         if neighbour not in visited:
            visited.append(neighbour)
            queue.append(neighbour)
   return l

visited = []
queue  = []
graph = enter_graph()
start_node = list(graph.keys())[0]
l = bfs_trav(visited, graph, start_node, l=[])
print("BFS traversal of the graph:", str(l))
ser = input("Enter Node to be searched -> ")
if ser in l:
   print("Node exists in the given graph.")
else:
   print("Node doesn't exist in the given graph.")
```

TEST CASE:

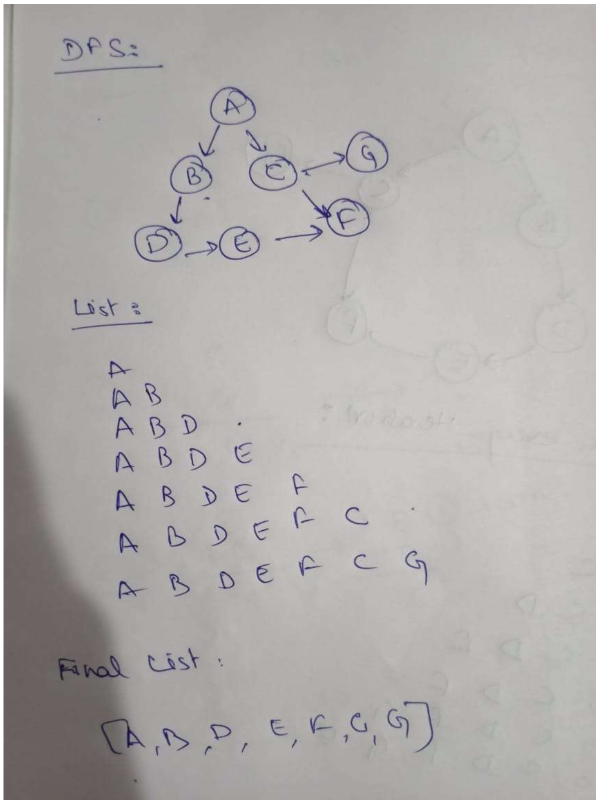
Case 1: DFS

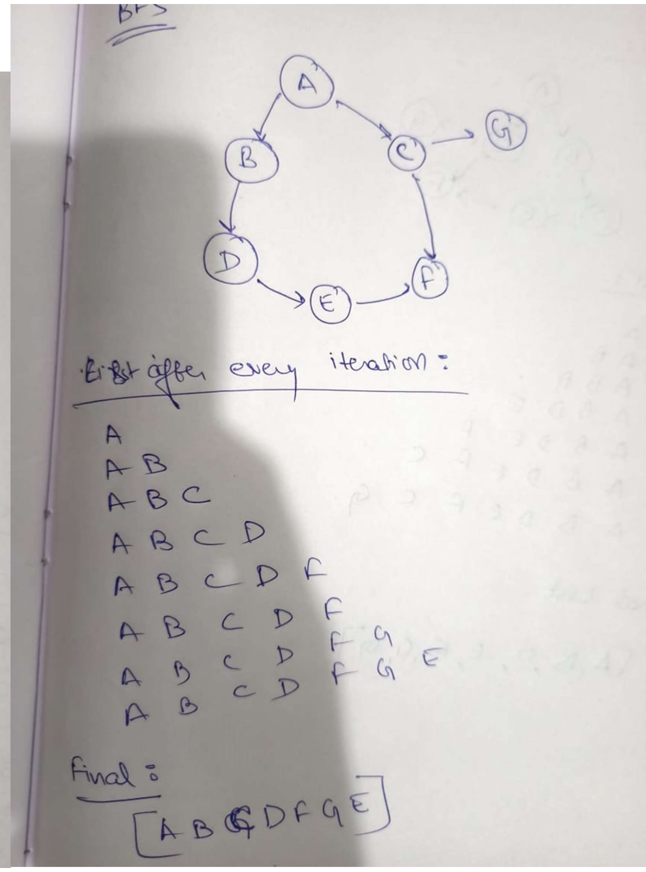
```
PS C:\Users\HARSH-PC\Desktop\college\AI\EXP_4> ./dfs.py
No of Nodes: 3
Node 0 -> A
Node 1 -> B
Adjacency nodes for node B->C
Node 2 -> C
Adjacency nodes for node C->
Entered Graph:-> {'A': ['B', 'C'], 'B': ['C'], 'C': []}
DFS traversal of the graph: ['A', 'B', 'C']
Enter Node to be searched -> A
Node exists in the given graph.
```

Case 2: BFS

```
PS C:\Users\HARSH-PC\Desktop\college\AI\EXP_4> .\bfs.py
No of Nodes: 4
Node 0 -> A
Adjacency nodes for node A->
Node 1 -> B
Adjacency nodes for node B->C
Node 2 -> C
Adjacency nodes for node C->B
Node 3 -> D
Adjacency nodes for node D->
Entered Graph:-> {'A': [], 'B': ['C'], 'C': ['B'], 'D': []}
BFS traversal of the graph: ['A']
Enter Node to be searched -> A
Node exists in the given graph.
PS C:\Users\HARSH-PC\Desktop\college\AI\EXP_4>
```

Verification:





Result: Hence, DFS and BFS methods were successfully implemented for an application and the algorithms and outputs were analyzed.