Page No. 1
Date

Assignment No: 1

Title :- DFS and BFS using recursive algorithm.

Problem Statement :- Implement depth first search algorithm and Breadth first search algorithm, use an undirected graph and develop a recursive algorithm for searching all the vertics of a graph. or tree data structure.

Objective :-

* To understand the concept BFS and DFS search techniques.
* To implement BFS and DFS search technique using recursive function on undirected graph or tree.

Theory :-

Depth-First search (DFS) :-

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structure One start at the root (selecting some arbitary node as the root for a graph). and explore as far as possible along each branch before backtracking.

Scanned with OKEN Scanner

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhausting searches of all the nodes by going ahead, if possible, else by backtracking.

Here, the world backtrack means that when you are moving forward and there are no more nodes along the current paths, 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.

Algorithm:-

Create a recursive function that takes the index of the node and a visited array.

1. Mark the current node as visited and print the node.
2. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

Scanned with OKEN Scanner

Dr.D.Y.PATIL
Varale Campus

Complexity Analysis:
- Time complexity: O(V+E), where V is the number of vertices and E is the number of edge in the graph.
- Space Complexity: O(V), since an extra visited array of size V is required.

Pseudocode of recursive BFS:-

```
DFS(adjacent[][], source, visited[], key) {
if(source == key) return true // We found the key visited[source] = True.

FOR node in adjacent[source]:
  IF visited[node] == false:
    DFS(adjacent, node, visited)
  END IF
END FOR.
return false // If it reaches here, then
             all nodes have been explored
             // and we still haven't found
             the key.
```

Breadth-first Search (BFS):-

Breadth-first search (BFS) is an algorithm

Scanned with OKEN Scanner

for traversing or searching tree or graph datastructures. It starts at the tree root (or some arbitary node of a graph, sometimes referred to as a 'search key') and explores the neighbour nodes first before moving to the next level neighbours.

Algorithm:-

1) Start by putting any one of the graph vertices at the back of a queue.
2) Take the front item of the queue and add it to the visited list.
3) Create a list of that vertex's adjacent nodes. Add the one's which aren't in the visited list to the back of the queue.
4) Keep repeating step 2 & 3 until the queue is empty.

Complexity Analysis:-

- Time complexity: The worst case breadth First search has to consider all paths to all possible nodes the time complexity of breadth-first search is $O(|E| + |V|)$ where $|V|$ and $E$ is the cardinality of set of vertices and edges respectively.

Scanned with OKEN Scanner

• Space Complexity :- The space complexity can also be mentioned as O(|V|) where |V| is the cardinality of the set of vertices.

Pseudocode:

```
recursiveBFS(Graph, Queue q, boolean[], visited,
int key) {
if(q.isEmpty())
 return "Not found";

// pop front node from queue and print
it
int v = q.poll();
if(v==-key) return "found";

for(Node u in graph.get(v))
{
 if(!visited[u])
{
   visited[u]=true;
   q.add(u);
}
}

recursiveBFS(graph, q, visited key);
}
Queue q=new Queue();
q.add(s)
```

Scanned with OKEN Scanner

recursiveBFS(graph, q, visited, key);

Conclusion :- We have implemented BFS and DFS using recursive algorithm for undirected graph.

Date 28/08/24

| Coding Efficiency | Viva | Timely Completion | Total | Dated Sign of Course In-charge |
|---|---|---|---|---|
| 5 | 3 | 2 | 10 | |
| | | | | |

| TS | PR | UC | VA | RV | Total |
|---|---|---|---|---|---|
| (2) | (2) | (2) | (2) | (2) | (10) |
| 02 | 01 | 01 | 02 | 02 | 08 |

10

Scanned with OKEN Scanner