Experiment 7

StudentName: Harsh Kumar **UID:22BCS15754**
Branch: BE CSE **Section: 22BCS_FL_IOT-603 B**
Semester: 5th **Date of Performance: 16-9-2024**
Subject Name: AP **Subject Code: 22CSP-314**

1. **Aim:** Breadth First Search: Shortest Reach.

2. **Objective:** You will be given a number of queries. For each query, you will be given a list of edges describing an undirected graph. After you create a representation of the graph, you must determine and report the shortest distance to each of the other nodes from a given starting position using the breadth-first search algorithm (BFS). Return an array of distances from the start node in node number order. If a node is unreachable, return -1 for that node.

3. **Algorithm:**
   1) Create an adjacency list from the given edges.
   2) Initialize a queue and push the source node s into it.
   3) Initialize a distance vector with size n, setting all elements to -1, and set the distance of the source node to 0.
   4) Perform BFS: for each node in the queue, visit its unvisited neighbors and update their distances by adding 6 to the current node's distance.
   5) Exclude the source node's distance from the result and prepare the answer vector.
   6) Return the answer vector containing the distances of all nodes except the source.

4. Implementation/Code:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
void bfs(vector<vector<int>>& adj, int s,
                vector<bool>& visited)
{
    queue<int> q;

    visited[s] = true;
    q.push(s);
    while (!q.empty()) {

        int curr = q.front();
        q.pop();
        cout << curr << " ";

        for (int x : adj[curr]) {
            if (!visited[x]) {
                visited[x] = true;
                q.push(x);
            }
        }
    }
}

void addEdge(vector<vector<int>>& adj,
```

```
                    int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}

int main()
{
    int V = 5;

    vector<vector<int>> adj(V);
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 24);
    addEdge(adj, 1, 32);
    addEdge(adj, 1, 44);
    addEdge(adj, 2, 84);

    vector<bool> visited(V, false);

    cout << "BFS starting from 0 : \n";
    bfs(adj, 0, visited);

    return 0;
      }
```

5. (a) Output:

```
/tmp/KPKkwy7FdE.o
BFS starting from 0 :
0 1 24 32 44

=== Code Execution Successful ===
```

6. **Time Complexity:** O(V+E)

7. **Learning Outcomes**:

1) Learn how to represent a graph using an adjacency list and how to process edges to create that list.
2) Understand the BFS algorithm to calculate the shortest distance in an unweighted graph from a starting node to other nodes.
3) Learn how to utilize a queue for BFS traversal to explore nodes level by level.
4) Learn how to represent and handle cases where certain nodes are unreachable by marking distances as -1.

1.(ii) Aim: Depth First Traversal (or DFS) .

2.(ii) Objective: The algorithm starts from a given source and explores all reachable vertices from the given source. It is similar to Preorder Tree Traversal where we visit the root, then recur for its children. In a graph, there maybe loops. So we use an extra visited array to make sure that we do not process a vertex again.

3.(ii) Implementation/Code:

```
#include <iostream>
#include <vector>
using namespace std;


void DFSRec(vector<vector<int>> &adj, vector<bool> &visited, int s){

    visited[s] = true;
    cout << s << " ";

    for (int i : adj[s])
        if (visited[i] == false)
            DFSRec(adj, visited, i);
}

void DFS(vector<vector<int>> &adj, int s){
    vector<bool> visited(adj.size(), false);
    DFSRec(adj, visited, s);
}

void addEdge(vector<vector<int>> &adj, int s, int t){
    adj[s].push_back(t);
    adj[t].push_back(s);
}

int main(){
    int V = 5;
    vector<vector<int>> adj(V);
```

```
    vector<vector<int>> edges={{1, 2},{1, 0},{2, 0},{2, 3},{2, 4}};
    for (auto &e : edges)
        addEdge(adj, e[0], e[1]);

    int s = 1;
    cout << "DFS from source: " << s << endl;
    DFS(adj, s);

    return 0;
}
```

4.(ii)Output:

```
DFS from source: 1
1 2 0 3 4

...Program finished with exit code 0
Press ENTER to exit console.
```