**LABORATORY ASSIGNMENT 6**

**NAME: AKSHAT SRIVASTAV**

**REGISTRATION ID: 19BCE0811**

**Data Structure and algorithms :**

**Q). Implement the ‘Breadth First Search (BFS) Algorithm’ in C/C++/C#/JAVA/Python/ MATLAB/R Programming language.**

[NOTE: The Graph must be represented using Adjacency List]

**Algorithm:**

**bfs(vertices, start)**

**Input** − The list of vertices, and the start vertex.

**Output** − Traverse all of the nodes, if the graph is connected.

Begin

   define an empty queue que

   at first mark all nodes status as unvisited

   add the start vertex into the que

   while que is not empty, do

      delete item from que and set to u

      display the vertex u

      for all vertices 1 adjacent with u, do

         if vertices[i] is unvisited, then

            mark vertices[i] as temporarily visited

            add v into the queue

         mark

      done

      mark u as completely visited

   done

End

**C++ Code:**

#include<iostream>

#include<queue>

#define NODE 6

using namespace std;

class node {

   public:

      int val;

      int state; //status

};

int graph[NODE][NODE] = {

   {0, 1, 1, 1, 0, 0},

   {1, 0, 0, 1, 1, 0},

   {1, 0, 0, 1, 0, 1},

   {1, 1, 1, 0, 1, 1},

   {0, 1, 0, 1, 0, 1},

   {0, 0, 1, 1, 1, 0}

};

void bfs(node \*vert, node s) {

   node u;

   int i, j;

   queue<node> que;

   for(i = 0; i<NODE; i++) {

      vert[i].state = 0; //not visited

   }

   vert[s.val].state = 1;//visited

   que.push(s); //insert starting node

   while(!que.empty()) {

      u = que.front(); //delete from queue and print

      que.pop();

      cout << char(u.val+'A') << " ";

      for(i = 0; i<NODE; i++) {

         if(graph[i][u.val]) {

            //when the node is non-visited

            if(vert[i].state == 0) {

               vert[i].state = 1;

               que.push(vert[i]);

            }

         }

      }

      u.state = 2;//completed for node u

   }

}

int main() {

   node vertices[NODE];

   node start;

   char s;

   for(int i = 0; i<NODE; i++) {

      vertices[i].val = i;

   }

   s = 'B';//starting vertex B

   start.val = s-'A';

   cout << "BFS Traversal: ";

   bfs(vertices, start);

   cout << endl;

}

**Output:**

BFS Traversal: B A D E C F

**Therefore what the above code does is:**

Input : The Adjacency matrix of the graph.

A B C D E F

A 0 1 1 1 0 0

B 1 0 0 1 1 0

C 1 0 0 1 0 1

D 1 1 1 0 1 1

E 0 1 0 1 0 1

F 0 0 1 1 1 0

Output : BFS Traversal: B A D E C F