Shaheed Zulfikar Ali Bhutto Institute of Science & Technology

COMPUTER SCIENCE DEPARTMENT

Total Marks: 7.5

Obtained Marks: _____

DATA STRUCTURE AND ALGORITHM

Lab Report # 12

Submitted To: Mam Tehreem

Submitted By: Hammad Qureshi .

Reg. Numbers: 2112114

Question no 1:
Write a program to find the node 1 using BFS approach

Write a program to find the node 1 using DFS approach
Code:

Bfs Code

```
#include <bits/stdc++.h>
using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph {
   int V; // No. of vertices

   // Pointer to an array containing adjacency
   // lists
   vector<list<int> > adj;

public:
   Graph(int V); // Constructor

   // function to add an edge to graph
   void addEdge(int v, int w);

   // prints BFS traversal from a given source s
   void BFS(int s);
};
```

```
Graph::Graph(int V)
{
   this->V = V;
   adj.resize(V);
}

void Graph::addEdge(int v, int w)
{
   adj[v].push_back(w); // Add w to v's list.
}

void Graph::BFS(int s)
{
   // Mark all the vertices as not visited
   vector<bool> visited;
   visited.resize(V, false);

   // Create a queue for BFS
   list<int> queue;

   // Mark the current node as visited and enqueue it
   visited[s] = true;
   queue.push_back(s);

   while (!queue.empty()) {
      // Dequeue a vertex from queue and print it
      s = queue.front();
      cout << s << " ";
      queue.pop_front();

      // Get all adjacent vertices of the dequeued
      // vertex s. If a adjacent has not been visited,
```

```
        // then mark it visited and enqueue it
        for (auto adjecent : adj[s]) {
            if (!visited[adjecent]) {
                visited[adjecent] = true;
                queue.push_back(adjecent);
            }
        }
    }
}

// Driver program to test methods of graph class
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 0) \n";
    g.BFS(0);

    return 0;
}
```

Dfs Code

```
#include <iostream>
#define MAX 5
```

```
const int stackSize = MAX;
using namespace std;


class Stack {

 private:
  int top;
  int arr[stackSize];

 public:

  Stack() {
   top = -1;
  }

 void push(int value) {

  if (top + 1 >= stackSize) {
   cout << "Stack Overflow" << endl;
  } else {
   top = top + 1;
   arr[top] = value;
  }

 }

 int pop() {

  int stackPopVal;
```

```
  if (top <= -1) {
    cout << "Stack underflow" << endl;
  } else {
    stackPopVal = arr[top];
    top--;
  }
  return stackPopVal;

}

bool isStachEmpty() {

  if (top == -1) {
     return true;
  } else {
     return false;
  }
}

int peek() {

  return arr[top];
}




void display() {
  if (top >= 0) {
    cout << "Stack elements are:";
    for (int i = top; i >= 0; i--)
      cout << arr[i] << " ";
    cout << endl;
```

```
   } else
     cout << "Stack is empty";
  }

};


int adjMatrix[MAX][MAX];

struct Vertex {
char label;
bool visited;
};



class DepthFirstSearch{

public:
struct Vertex* lstVertices[MAX];
int vertexCount = 0;

DepthFirstSearch() {
 int i,j;
 for(i = 0; i< MAX; i++) {
  for(j = 0; j< MAX; j++) {
     adjMatrix[i][j] = 0;
    }
  }

}
```

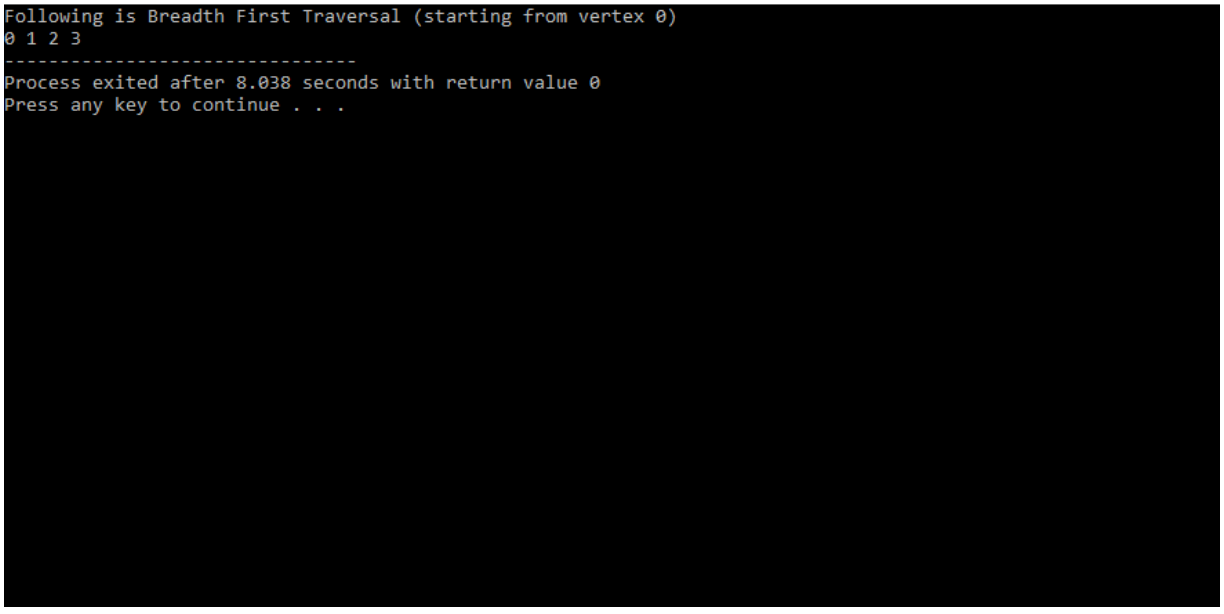
```
void addVertex(char label) {
 Vertex *vertex = new Vertex;
 vertex->label = label;
 vertex->visited = false;
 lstVertices[vertexCount++] = vertex;
}

void addEdge(int edgeStart,int edgeEnd) {
 adjMatrix[edgeStart][edgeEnd] = 1;
 adjMatrix[edgeEnd][edgeStart] = 1;
}

//display the vertex
void displayVertex(int vertexIndex) {
 cout<<lstVertices[vertexIndex]->label<<" ";
}

int getAdjUnvisitedVertex(int vertexIndex) {
   int i;
   for(i = 0; i<vertexCount; i++) {
      if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited ==
false)
      return i;
   }
   return -1;
}



void DFS(){
 Stack objStack;
 lstVertices[0]->visited = true;
```

```
displayVertex(0);
objStack.push(0);

while(!objStack.isStachEmpty()){
int unvisitedVertexIndex = getAdjUnvisitedVertex(objStack.peek());

if(unvisitedVertexIndex == -1){
   objStack.pop();
 }else{
   lstVertices[unvisitedVertexIndex]->visited = true;
   displayVertex(unvisitedVertexIndex);
   objStack.push(unvisitedVertexIndex);
 }

}

 }

};

int main() {

DepthFirstSearch obj;


obj.addVertex('S'); // 0
obj.addVertex('A'); // 1
obj.addVertex('B'); // 2
obj.addVertex('C'); // 3
obj.addVertex('D'); // 4

obj.addEdge(0, 1); //S - A
```

```
obj.addEdge(0, 2); // S - B
obj.addEdge(0, 3); // S - C
obj.addEdge(1, 4); // A - D
obj.addEdge(2, 4); // B - D
obj.addEdge(3, 4); // C - D

cout<<"Started with index S "<<endl;
obj.DFS();

return 0;

}
```

CONSOLE SCREEN:

Bfs



Dfs

```
Started with index S
S A D B C
--------------------------------
Process exited after 7.629 seconds with return value 0
Press any key to continue . . .
```