A picture containing text

Description automatically generated



Lab Journal: 12

Date: 15 / 12 / 24

Student : Abdul Rafay

Enrollment : 01-131232-004

Department of Software Engineering

Bahria University, Islamabad

Data Structures & Algorithms Lab (Spring-2024)

Teacher: RAHEELA AMBRIN

**Comments:**

**Signature**

### Code:

All the code files are uploaded on GitHub: <https://github.com/CharlieFour/DSA_Lab>

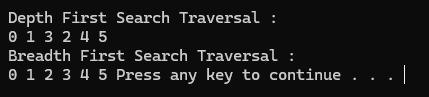
You can check out the code on GitHub in Lab\_12 folder

### Code:

#### Task\_01:

#include<iostream>  
#include<vector>  
#include<queue>  
using namespace std;  
  
void DFS(int v, vector<bool>& visited, const vector<vector<int>>& graph)  
{  
    visited[v] = true;  
    cout << v << " ";  
  
    for(int neighbor : graph[v])  
    {  
        if(!visited[neighbor])  
        {  
            DFS(neighbor, visited, graph);  
        }  
    }  
}  
  
void BSF(int start, const vector<vector<int>>& graph)  
{  
    vector<bool> visited(graph.size(), false);  
    queue<int> q;  
  
    q.push(start);  
    visited[start] = true;  
  
    while(!q.empty())  
    {  
        int current = q.front();  
        q.pop();  
        cout << current << " ";  
  
        for(int neighbor : graph[current])  
        {  
            if(!visited[neighbor])  
            {  
                visited[neighbor] = true;  
                q.push(neighbor);  
            }  
        }  
    }  
}  
  
int main()  
{  
    vector<vector<int>> graph  
    {  
        {1, 2},                                                          
        {3},                                                          
        {4, 5},                                                             
        {},  
        {},  
        {}  
    };  
    vector<bool> visited(graph.size(), false);  
    cout << "Depth First Search Traversal :" << endl;  
    DFS(0, visited, graph);  
  
    cout << "\nBreadth First Search Traversal :" << endl;  
    BSF(0, graph);  
  
    system("pause");  
    return 0;  
}

### Screen Shots:



#### Task\_02

#include <iostream>  
#include <vector>  
#include <queue>  
#include <climits>  
  
using namespace std;  
  
class Node  
{  
public:  
    int vertexNumber;  
    vector<pair<int, int>> children;  
  
    Node(int vertexNumber)  
    {  
        this->vertexNumber = vertexNumber;  
    }  
  
    void add\_child(int vNumber, int weight)  
    {  
        children.push\_back({vNumber, weight});  
    }  
};  
  
vector<int> dijkstraDist(const vector<Node\*>& graph, int source, vector<int>& path)  
{  
    int n = graph.size();  
    vector<int> dist(n, INT\_MAX);  
    vector<bool> visited(n, false);  
    dist[source] = 0;  
    path[source] = -1;  
  
    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;  
    pq.push({0, source});  
  
    while(!pq.empty())  
    {  
        int u = pq.top().second;  
        pq.pop();  
  
        if (visited[u]) continue;  
        visited[u] = true;  
  
        for(auto& child : graph[u]->children)  
        {  
            int v = child.first;  
            int weight = child.second;  
  
            if(dist[u] + weight < dist[v])  
            {  
                dist[v] = dist[u] + weight;  
                path[v] = u;  
                pq.push({dist[v], v});  
            }  
        }  
    }  
    return dist;  
}  
  
void printPath(const vector<int>& path, int destination)  
{  
    if(destination == -1)  
    {  
        return;  
    }  
    printPath(path, path[destination]);  
    cout << destination << " ";  
}  
  
int main()  
{  
    int n = 4;  
    vector<Node\*> graph;  
  
    for(int i = 0; i < n; i++)  
    {  
        graph.push\_back(new Node(i));  
    }  
  
    graph[0]->add\_child(1, 1);  
    graph[0]->add\_child(2, 4);  
    graph[1]->add\_child(2, 2);  
    graph[1]->add\_child(3, 6);  
    graph[2]->add\_child(3, 3);  
  
    int source = 0;  
    vector<int> path(n);  
    vector<int> dist = dijkstraDist(graph, source, path);  
  
    for (int i = 0; i < n; i++)  
    {  
        if (dist[i] == INT\_MAX)  
        {  
            cout << "Vertex " << i << " is not reachable from source " << source << ".\n";  
        }  
        else  
        {  
            cout << "Distance from Node " << source << " to Node " << i << ": " << dist[i] << "\n";  
            cout << "Path: " << source << " ";  
          
            printPath(path, i);  
            cout << "\n";  
        }  
    }  
  
    for(auto node : graph)  
    {  
        delete node;  
    }  
  
    system("pause");  
    return 0;  
}

### Screen Shots:

