**Name:Vishal Indradev Chauhan**

**PRN:2020BTECS00090**

**Grp:S6**

**Implementation of Graph**

Implementation of Graph, Graph Traversal-BFS & DFS.

// A simple representation of graph using STL

#include <bits/stdc++.h>

using namespace std;

// A utility function to add an edge in an

// undirected graph.

void addEdge(vector<int> adj[], int u, int v)

{

    adj[u].push\_back(v);

    adj[v].push\_back(u);

}

// A utility function to print the adjacency list

// representation of graph

void printGraph(vector<int> adj[], int V)

{

    for (int v = 0; v < V; ++v)

    {

        cout << "\n Adjacency list of vertex "

             << v << "\n head ";

        for (auto x : adj[v])

            cout << "-> " << x;

        printf("\n");

    }

}

// Driver code

int main()

{

    int V = 5;

    vector<int> adj[V];

    addEdge(adj, 0, 1);

    addEdge(adj, 0, 4);

    addEdge(adj, 1, 2);

    addEdge(adj, 1, 3);

    addEdge(adj, 1, 4);

    addEdge(adj, 2, 3);

    addEdge(adj, 3, 4);

    printGraph(adj, V);

    return 0;

}

OUTPUT:

Adjacency list of vertex 0

head -> 1-> 4

Adjacency list of vertex 1

head -> 0-> 2-> 3-> 4

Adjacency list of vertex 2

head -> 1-> 3

Adjacency list of vertex 3

head -> 1-> 2-> 4

Adjacency list of vertex 4

head -> 0-> 1-> 3

II-Graph Traversal-BFS

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

queue<int> q;

vector<int> graph[4]{

    {0, 1, 0, 1},

    {1, 0, 0, 1},

    {0, 1, 1, 1},

    {1, 1, 1, 0}};

void graph\_traversal()

{

    for (int i = 0; i < 4; i++)

    {

        for (auto x = graph[i].begin(); x != graph[i].end(); x++)

        {

            cout << \*x << " ";

        }

        cout << endl;

    }

}

int main()

{

    int visited\_arr[4] = {0};

    int x=3;

    q.push(x);

    cout << x << " ";

    visited\_arr[x] = 1;

    while (!q.empty())

    {

         q.pop();

        for (int a = 0; a < 4; a++)

        {

            if (graph[x][a] == 1 && visited\_arr[a] == 0)

            {

                cout << a << " ";

                visited\_arr[a] = 1;

                q.push(a);

            }

        }

    }

}

OUTPUT:

3 0 1 2

PS E:\DSA\College\graph>

III-Graph Traversal- DFS

#include <bits/stdc++.h>

using namespace std;

class Graph {

public:

    map<int, bool> visited;

    map<int, list<int> > adj;

    // function to add an edge to graph

    void addEdge(int v, int w);

    // DFS traversal of the vertices

    // reachable from v

    void DFS(int v);

};

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::DFS(int v)

{

    // Mark the current node as visited and

    // print it

    visited[v] = true;

    cout << v << " ";

    // Recur for all the vertices adjacent

    // to this vertex

    list<int>::iterator i;

    for (i = adj[v].begin(); i != adj[v].end(); ++i)

        if (!visited[\*i])

            DFS(\*i);

}

// Driver code

int main()

{

    // Create a graph given in the above diagram

    Graph g;

    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 Depth First Traversal"

            " (starting from vertex 0) \n";

    g.DFS(0);

    return 0;

}

OUTPUT:

Following is Depth First Traversal (starting from vertex 0)

0 1 2 3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*