**Representation of a Graph (Adjacency List)**

#include <iostream>

#include <vector>

using namespace std;

class Graph {

int V; // Number of vertices

vector<vector<int>> adjList; // Adjacency list

public:

Graph(int V) {

this->V = V;

adjList.resize(V);

}

void addEdge(int u, int v) {

adjList[u].push\_back(v);

adjList[v].push\_back(u); // Remove this for a directed graph

}

void printGraph() {

for (int i = 0; i < V; ++i) {

cout << "Vertex " << i << " -> ";

for (int v : adjList[i])

cout << v << " ";

cout << endl;

}

}

};

int main() {

Graph g(5); // Graph with 5 vertices

g.addEdge(0, 1);

g.addEdge(0, 4);

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(3, 4);

g.printGraph();

return 0;

}

**2. Directed Graph (Adjacency Matrix)**

#include <iostream>

using namespace std;

class DirectedGraph {

int V;

int\*\* adjMatrix;

public:

DirectedGraph(int V) {

this->V = V;

adjMatrix = new int\*[V];

for (int i = 0; i < V; ++i) {

adjMatrix[i] = new int[V];

for (int j = 0; j < V; ++j)

adjMatrix[i][j] = 0;

}

}

void addEdge(int u, int v) {

adjMatrix[u][v] = 1; // For a weighted graph, use weights instead of 1

}

void printGraph() {

for (int i = 0; i < V; ++i) {

for (int j = 0; j < V; ++j)

cout << adjMatrix[i][j] << " ";

cout << endl;

}

}

~DirectedGraph() {

for (int i = 0; i < V; ++i)

delete[] adjMatrix[i];

delete[] adjMatrix;

}

};

int main() {

DirectedGraph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 3);

g.printGraph();

return 0;

}

**3. Weighted Graph**

#include <iostream>

#include <vector>

using namespace std;

class WeightedGraph {

int V;

vector<vector<pair<int, int>>> adjList;

public:

WeightedGraph(int V) {

this->V = V;

adjList.resize(V);

}

void addEdge(int u, int v, int weight) {

adjList[u].push\_back({v, weight});

adjList[v].push\_back({u, weight}); // Remove for a directed graph

}

void printGraph() {

for (int i = 0; i < V; ++i) {

cout << "Vertex " << i << " -> ";

for (auto& edge : adjList[i])

cout << "(" << edge.first << ", " << edge.second << ") ";

cout << endl;

}

}

};

int main() {

WeightedGraph g(5);

g.addEdge(0, 1, 10);

g.addEdge(0, 4, 20);

g.addEdge(1, 2, 30);

g.addEdge(1, 3, 40);

g.addEdge(1, 4, 50);

g.addEdge(3, 4, 60);

g.printGraph();

return 0;

}