Lab 12(B PART)

Task #3:

#include <iostream>

using namespace std;

class UndirectedGraph

{

private:

int N; // Number of vertices

int\*\* adjacencyList; // 2D array for the adjacency list

public:

UndirectedGraph(int n) : N(n)

{

adjacencyList = new int\*[N + 1]; // Adding 1 to N to make it 1-indexed

for (int i = 1; i <= N; i++)

{

adjacencyList[i] = nullptr;

}

}

void addEdge(int x, int y) {

if (x >= 1 && x <= N && y >= 1 && y <= N) {

// Add y to the adjacency list of x

adjacencyList[x] = addNode(adjacencyList[x], y);

// Add x to the adjacency list of y

adjacencyList[y] = addNode(adjacencyList[y], x);

}

}

// Function to add a node to the adjacency list

int\* addNode(int\* head, int data)

{

int\* newNode = new int[2];

newNode[0] = data;

newNode[1] = 0;

if (head == nullptr)

{

head = newNode;

}

else {

int\* current = head;

while (current[1] != 0) {

current = current + 2;

}

current[1] = (int)newNode;

}

return head;

}

// Function to print the adjacency list

void printAdjacencyList() {

for (int i = 1; i <= N; i++) {

std::cout << "Vertex " << i << ": ";

int\* current = adjacencyList[i];

while (current != nullptr) {

std::cout << current[0] << " ";

current = (int\*)current[1];

}

cout <<endl;

}

}

};

int main()

{

int N = 5; // Number of vertices

UndirectedGraph graph(N);

// Adding edges to the graph

graph.addEdge(1, 2);

graph.addEdge(2, 3);

graph.addEdge(3, 4);

graph.addEdge(4, 5);

graph.addEdge(5, 1);

cout << "Adjacency List:" << endl;

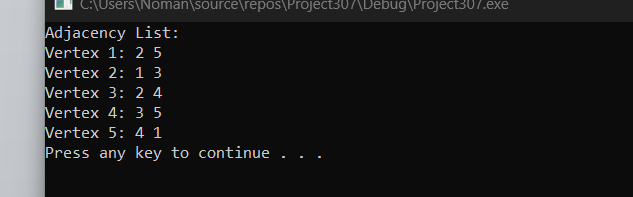
graph.printAdjacencyList();

system("pause");

return 0;

}

Output:



Task #4:

#include <iostream>

#include <string>

using namespace std;

class City

{

public:

string name; // attributes

City\* next;

int distance;

City(const string& cityName, int dist) : name(cityName), distance(dist), next(nullptr) {}// constructor

};

class Vehicle {

private:

int capacity; // Number of packages or maximum weight the vehicle can carry

Vehicle\* next;

public:

Vehicle(int capacity) : capacity(capacity), next(nullptr) {}

int getCapacity() const {

return capacity;

}

Vehicle\* getNext() const {

return next;

}

void setNext(Vehicle\* vehicle) {

next = vehicle;

}

};

class Package {

private:

std::string destination;

int size;

int weight;

Package\* next;

public:

Package(const string& destination, int size, int weight): destination(destination), size(size), weight(weight), next(nullptr) {} //using intializer

string getDestination() const

{

return destination;

}

int getSize() const {

return size;

}

int getWeight() const {

return weight;

}

Package\* getNext() const {

return next;

}

void setNext(Package\* package) {

next = package;

}

};

class DeliveryRoute {

private:

City\* route;

public:

DeliveryRoute() : route(nullptr) {}

void addCity(const string& city, int distance) {

City\* newCity = new City(city, distance);

newCity->next = route;

route = newCity;

}

const City\* getRoute() const {

return route;

}

};

class TransportationNetwork {

private:

City\* cities;

Vehicle\* vehicles;

Package\* packages;

public:

TransportationNetwork() : cities(nullptr), vehicles(nullptr), packages(nullptr) {}

void addCity(const string& city) {

City\* newCity = new City(city, 0);

newCity->next = cities;

cities = newCity;

}

void addRoad(const string& city1, const string& city2, int distance) {

City\* cityNode1 = nullptr;

City\* cityNode2 = nullptr;

for (City\* city = cities; city != nullptr; city = city->next) {

if (city->name == city1) {

cityNode1 = city;

}

if (city->name == city2) {

cityNode2 = city;

}

}

if (cityNode1 && cityNode2) {

City\* newCity1 = new City(city1, distance);

City\* newCity2 = new City(city2, distance);

newCity1->next = cityNode2->next;

cityNode2->next = newCity1;

newCity2->next = cityNode1->next;

cityNode1->next = newCity2;

}

}

void addVehicle(const Vehicle& vehicle) {

Vehicle\* newVehicle = new Vehicle(vehicle.getCapacity());

newVehicle->setNext(vehicles);

vehicles = newVehicle;

}

void addPackage(const Package& package) {

Package\* newPackage = new Package(package.getDestination(), package.getSize(), package.getWeight());

newPackage->setNext(packages);

packages = newPackage;

}

void planDeliveryRoutes() {

if (!vehicles)

{

cout << "No vehicles available. Please add vehicles to the fleet first." << std::endl;

return;

}

DeliveryRoute route;

for (Package\* pkg = packages; pkg != nullptr; pkg = pkg->getNext()) {

route.addCity(pkg->getDestination(), 0);

}

// Add the headquarters as the starting and ending point for the route

route.addCity("Headquarters", 0);

// Display the planned route for the first vehicle

cout << "Planned Route for Vehicle 1:" << endl;

const City\* city = route.getRoute();

while (city && city->next) {

int distance = city->next->distance;

cout << city->name << " -> " << city->next->name << " (" << distance << " km)" <<endl;

city = city->next;

}

}

};

int main() {

TransportationNetwork network;

network.addCity("City A");

network.addCity("City B");

network.addCity("City C");

network.addRoad("City A", "City B", 50);

network.addRoad("City A", "City C", 80);

network.addRoad("City B", "City C", 30);

network.addVehicle(Vehicle(100));

network.addVehicle(Vehicle(150));

network.addPackage(Package("City B", 2, 5));

network.addPackage(Package("City C", 1, 3));

network.addPackage(Package("City A", 3, 10));

network.planDeliveryRoutes();

system("pause");

return 0;

}

Output:

