Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Отчет**

по лабораторной работе №6

по курсу “ Логика и основы алгоритмизации в инженерных задачах”

на тему “Поиск расстояний во взвешенном графе”

Выполнили

студенты группы 22ВВВ3:

Бормотов А.А.

Кочетков А.М.

Кузьмин Д.В.

Приняли

Акифьев И.В.

Юрова О.В.

Пенза 2023

### **Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Выведите сгенерированные матрицы на экран.
2. \* Для указанных графов преобразуйте представление матриц смежности в списки смежности. Выведите полученные списки на экран.

### **Задание 2**

1. Для матричной формы представления графов выполните операцию:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

1. \* Для представления графов в виде списков смежности выполните операцию:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

**Задание 3**

1. Для матричной формы представления графов выполните операцию:

а) объединения *G* = *G*1  *G*2

б) пересечения *G* = *G*1  *G*2

в) кольцевой суммы *G* = *G*1  *G*2

Результат выполнения операции выведите на экран.

**Задание 4 \***

1. Для матричной формы представления графов выполните операцию декартова произведения графов *G = G*1X *G*2.

Результат выполнения операции выведите на экран.

**Листинг**

**#include <iostream>**

**#include <vector>**

**#include <cstdlib>**

**#include <time.h>**

**#include <list>**

**#include <string>**

**//#include <bits/stdc++.h>**

**using namespace std;**

**/// use this:**

**/// int size = (\*size\_old - 1);**

**/// \*size\_old += 1;**

**/// and this: nodes->push\_back(\*size\_old+1);**

**// initiates a Graph of specified size in a matrix form, fills it with random numbers.**

**// no looping nodes**

**vector<vector<int>> create\_Graph(int size, vector<int>\* nodes) {**

**for (int i = 0; i < size; i++) nodes->push\_back(i + 1);**

**vector<vector<int>> matrix(size, vector<int>(size));**

**for (int i = 0; i != size; i++) {**

**for (int j = i; j != size; j++) {**

**if (j == i) {**

**matrix[i][j] = 0;**

**continue;**

**}**

**matrix[i][j] = rand() % 2;**

**matrix[j][i] = matrix[i][j];**

**}**

**}**

**return matrix;**

**}**

**// displays it. Uses a vector with all of the node 'numbers' to properly display it**

**void display\_Graph(vector<vector<int>> matrix, int size, string name, vector<int> nodes) {**

**cout << "Adjacency matrix for graph " << name << ":" << endl;**

**for (int i = 0; i < size; i++) {**

**cout << nodes[i] << " | ";**

**for (int j = 0; j < size; j++) {**

**cout << matrix[i][j] << " ";**

**}**

**cout << endl;**

**}**

**cout << " ";**

**for (int k = 0; k < size; k++) cout << "\_" << " ";**

**cout << endl << " ";**

**for (int k = 0; k < size; k++) cout << nodes[k] << " ";**

**cout << endl << endl;**

**}**

**list<list<int>> convert\_to\_adj\_list(vector<vector<int>> matrix, int size) {**

**list<list<int>> graph;**

**list<int> temp\_list;**

**for (int i = 0; i != size; i++) {**

**for (int j = 0; j != size; j++) {**

**if (matrix[i][j] == 1) {**

**temp\_list.push\_back(j);**

**}**

**}**

**graph.push\_back(temp\_list);**

**temp\_list.clear();**

**}**

**return graph;**

**}**

**void display\_adj\_list(list<list<int>> adj\_list, int size, string name, vector<int> nodes) {**

**int counter = 0;**

**cout << "Adjacency list for graph " << name << ":" << endl;**

**for (auto iterator = adj\_list.begin(); iterator != adj\_list.end(); ++iterator) {**

**cout << nodes[counter];**

**counter++;**

**list<int>::iterator sub\_iterator;**

**list<int>& sub\_pointer = \*iterator;**

**for (sub\_iterator = sub\_pointer.begin(); sub\_iterator != sub\_pointer.end(); sub\_iterator++) {**

**cout << "-> " << (\*sub\_iterator) + 1;**

**}**

**cout << endl;**

**}**

**cout << endl;**

**}**

**vector<vector<int>> close\_nodes(vector<vector<int>> matrix, int\* size\_old, int node1, int node2, vector<int>\* nodes) {**

**node1--;**

**node2--;**

**int size = (\*size\_old - 1);**

**\*size\_old -= 1;**

**vector<int> new\_node(size);**

**//creating new node by 'closing' two specified**

**// because we used new\_node[i] it was messing up position for new closed node**

**// I can add another counter - which is easy, or try to insert thenew nums i**

**// can't I just omit using 'j' and also the first if, and just check the second if and do a .push\_back intead of 'new\_node[j] = 1' ???**

**int j = 0;**

**for (int i = 0; i <= size; i++) {**

**if (i == node1 || i == node2) continue;**

**if (matrix[node1][i] || matrix[node2][i]) new\_node[j] = 1;**

**j++;**

**/\***

**if (!matrix[node1][i] && !matrix[node2][i]) {**

**//new\_node[j] = 0;**

**j++;**

**}**

**else {**

**new\_node[j] = 1;**

**j++;**

**}**

**\*/**

**}**

**// should there be 'size-1' ???**

**if (matrix[node1][node2]) new\_node[size - 1] = 1; //adding loop if it exists**

**matrix.erase(matrix.begin() + node2);**

**matrix.erase(matrix.begin() + node1);**

**for (j = 0; j != size - 1; j++) {**

**matrix[j].erase(matrix[j].begin() + node2);**

**matrix[j].erase(matrix[j].begin() + node1);**

**matrix[j].push\_back(new\_node[j]);**

**}**

**matrix.push\_back(new\_node);**

**//there we delete the nodes**

**//int temp = max(node1, node2);**

**//int mini = min(node1, node2);**

**nodes->erase(nodes->begin() + node2);**

**nodes->erase(nodes->begin() + node1);**

**nodes->push\_back(size + 2);**

**return matrix;**

**}**

**// rename it, cuz it is used for borh vectors and lists**

**// shouldve used:**

**// nodes.erase(nodes.begin() + node2);**

**// instead of all this iterator nonesense**

**//maybe there needs to be size-1 or sth...**

**list<list<int>> close\_nodes\_adj(list<list<int>> graph, int\* size\_old, int node1, int node2, vector<int>\* nodes) {**

**node1--;**

**node2--;**

**int size = \*size\_old;**

**\*size\_old -= 1;**

**nodes->erase(nodes->begin() + node2);**

**nodes->erase(nodes->begin() + node1);**

**nodes->push\_back(size + 1);**

**list<int> new\_sub\_list;**

**list<list<int>>::iterator it = graph.begin();**

**advance(it, node1);**

**list<int>& sub\_pointer = \*it;**

**list<int>::iterator sub\_iterator;**

**int check = 0;**

**for (sub\_iterator = sub\_pointer.begin(); sub\_iterator != sub\_pointer.end(); sub\_iterator++) {**

**new\_sub\_list.push\_back(\*sub\_iterator);**

**if (\*sub\_iterator == node2) check = 1;**

**}**

**it = graph.begin();**

**advance(it, node2);**

**sub\_pointer = \*it;**

**new\_sub\_list.merge(sub\_pointer); // by mergin we are also deleting the list number 'node2'**

**new\_sub\_list.unique();**

**new\_sub\_list.remove(node1);**

**new\_sub\_list.remove(node2);**

**if (check == 1) new\_sub\_list.push\_back(size);**

**//here we are proly traversing throught the whole list of lists, but technically**

**// we can only search for lists with indexes from 'new\_sub\_list' and do not even look in the rest of them**

**// i did the 'new sub list' traversing but exactly zero line of this code is working properly**

**for (auto iterator = new\_sub\_list.begin(); (iterator != new\_sub\_list.end()) && \*iterator != size; ++iterator) {**

**it = graph.begin();**

**advance(it, \*iterator);**

**//looks like we pass teh link but not the actual memory pointer or something.**

**// So it changes everything only in sub pointer**

**sub\_pointer = \*it;**

**sub\_pointer.push\_back(size);**

**sub\_pointer.remove(node1);**

**sub\_pointer.remove(node2);**

**\*it = sub\_pointer;**

**// for (sub\_iterator = sub\_pointer.begin(); sub\_iterator != sub\_pointer.end(); sub\_iterator++) {**

**// cout<< (\*sub\_iterator) << " is erased" << endl;**

**// if (\*sub\_iterator == node1 || \*sub\_iterator == node2) sub\_pointer.erase(sub\_iterator);**

**// //cout<< (\*sub\_iterator) << "\_" << endl;**

**// }**

**}**

**it = graph.begin();**

**advance(it, node2);**

**graph.erase(it);**

**it = graph.begin();**

**advance(it, node1);**

**graph.erase(it);**

**graph.push\_back(new\_sub\_list);**

**//РІРµР·РґРµ РіРґРµ РІСЃС‚СЂРµС‡Р°РµС‚СЃСЏ node1 or node2 РјС‹:**

**//СѓРґР°Р»СЏРµРј РёС…, РѕСЃС‚Р°Р»СЊРЅРѕРµ РІСЃРµ СЃРґРІРёРіР°РµС‚СЃСЏ Рё РІ РєРѕРЅС†Рµ РґРѕР±Р°РІР»СЏРµРј РЅРѕРІСѓСЋ РЅР°РґРѕ Рє СЃРїРёСЃРєСѓ**

**//СЃР°Рј СЃРїРёСЃРєРё СЃ РёРЅРґРµРєСЃР°РјРё node1, node2 С‚РѕР¶Рµ СѓРґР°Р»СЏРµРј || list.remove()!**

**//РёР»Рё РµС‰Рµ - new\_sub\_list СЃРѕРґРµСЂР¶РёС‚ СЃРїРёСЃРѕРє РІСЃРµС… РЅРѕРґ РєРѕС‚РѕСЂС‹Рµ РЅР°РґРѕ РёР·РјРµРЅРёС‚СЊ,**

**// РѕСЃС‚Р°Р»СЊРЅС‹Рµ - РїРѕС„РёРі**

**return graph;**

**}**

**vector<vector<int>> contract\_nodes(vector<vector<int>> graph, int\* size\_old, int node1, int node2, vector<int>\* nodes) {**

**node1--;**

**node2--;**

**int size = \*size\_old;**

**\*size\_old -= 1;**

**nodes->erase(nodes->begin() + node2);**

**for (auto i = 0; i < size + 1; i++) {**

**// if we ecnounter ref to 'node2' we make it a ref to 'node1'**

**if (graph[i][node2] == 1) {**

**graph[i][node1] = 1;**

**}**

**graph[i].erase(graph[i].begin() + node2); // erase ref to 'node2' completely**

**if (graph[node1][i] == 1) continue;**

**if (graph[node2][i] == 1) {**

**graph[node1][i] = 1;**

**continue;**

**}**

**}**

**graph[node1][node1] = 0;**

**graph.erase(graph.begin() + node2);**

**return graph;**

**}**

**list<list<int>> contract\_nodes\_adj(list<list<int>> graph, int\* size\_old, int node1, int node2, vector<int>\* nodes) {**

**node1--;**

**node2--;**

**\*size\_old -= 1;**

**int temp\_check = 0;**

**// get the list for node1**

**list<list<int>>::iterator it = graph.begin();**

**advance(it, node1);**

**list<int>& sub\_pointer = \*it;**

**// get the list for node2**

**it = graph.begin();**

**advance(it, node2);**

**list<int>& sub\_pointer2 = \*it;**

**sub\_pointer.merge(sub\_pointer2);**

**sub\_pointer.unique();**

**sub\_pointer.remove(node2);**

**sub\_pointer.remove(node1);**

**/// some problem with 'sub\_pointer2' it's a &list, should be just a list**

**///**

**list<int> contracted\_sub\_list = sub\_pointer;**

**for (auto iterator1 = contracted\_sub\_list.begin(); iterator1 != contracted\_sub\_list.end(); ++iterator1) {**

**it = graph.begin();**

**//ther was \*iterator1**

**advance(it, \*iterator1);**

**sub\_pointer2 = \*it;**

**//proly can add an if() to test if node is even present. If it's not in the line,**

**//then we don't have to do anything with the list at all and just continue; and move on.**

**list<int> temp\_list = sub\_pointer2;**

**for (list<int>::iterator iter = sub\_pointer2.begin(); iter != sub\_pointer2.end(); ++iter) {**

**/// THERE IS AN INFINTE CYCLE IN CASE OF [3, 4] IT DELETES THE 4 (NODE2) AND THE SIZE OF LIST CHANGES AND ITER MESSES UP**

**/// I HAVE NO IDEA FOR NOW HOW TO FIX THIS**

**if (\*iter == node1) temp\_check = 1;**

**if (\*iter == node2) {**

**if (temp\_check == 1) {**

**temp\_list.remove(node2);**

**}**

**else {**

**temp\_list.remove(node2);**

**temp\_list.push\_back(node1);**

**temp\_list.sort();**

**}**

**}**

**//\*it = sub\_pointer2;**

**}**

**sub\_pointer2 = temp\_list;**

**\*it = sub\_pointer2;**

**temp\_check = 0;**

**}**

**it = graph.begin();**

**advance(it, node2);**

**graph.erase(it);**

**nodes->erase(nodes->begin() + node2);**

**return graph;**

**}**

**vector<vector<int>> split\_nodes(vector<vector<int>> graph, int\* size\_old, int node1, vector<int>\* nodes) {**

**nodes->push\_back(\*size\_old + 1);**

**int size = (\*size\_old - 1); // cuz notation starts from zero**

**\*size\_old += 1;**

**node1--;**

**vector<int> new\_node = graph[node1];**

**new\_node[node1] = 1; // add a connection to splitted node**

**new\_node.push\_back(0); //new node is not looping**

**for (auto i = 0; i <= size; i++) {**

**if (new\_node[i] == 1) {**

**graph[i].push\_back(1);**

**}**

**else {**

**graph[i].push\_back(0);**

**}**

**}**

**graph.push\_back(new\_node);**

**return graph;**

**}**

**list<list<int>> split\_nodes\_adj(list<list<int>> graph, int\* size\_old, int node1, vector<int>\* nodes) {**

**nodes->push\_back(\*size\_old + 1); //because we use pointer we also use '->'**

**int size = (\*size\_old - 1); // cuz notation starts from zero**

**\*size\_old += 1;**

**node1--;**

**list<list<int>>::iterator it = graph.begin();**

**advance(it, node1);**

**list<int> new\_node = \*it;**

**new\_node.push\_back(node1);**

**new\_node.sort();**

**list<int>& sub\_pointer = \*it;**

**for (auto iter = new\_node.begin(); iter != new\_node.end(); ++iter) {**

**it = graph.begin();**

**advance(it, \*iter);**

**(\*it).push\_back(size + 1);**

**}**

**graph.push\_back(new\_node);**

**return graph;**

**}**

**vector<vector<int>> merge\_graphs(vector<vector<int>> graph1, vector<vector<int>> graph2, int size1, int size2, vector<int>\* nodes, int\* new\_size) {**

**int j = 0, i = 0;**

**int size = max(size1, size2);**

**vector<vector<int>> new\_graph(size, vector<int>(size));**

**for (int i = 0; i < size; i++) nodes->push\_back(i + 1);**

**\*new\_size = size;**

**/// РµРїС‚Р° РіСЂР°С„ Р¶Рµ РЅРµ РЅР°РїСЂР°РІР»РµРЅРЅС‹Р№ Рё РЅРµРІР·РІРµС€РµРЅРЅС‹Р№ С‚СѓС‚ РјРѕР¶РЅРѕ РІ РІС‚РѕСЂРѕРј С†РёРєР»Рµ СЃРґРµР»Р°С‚СЊ i = j РёР»Рё С‡РµС‚ С‚РёРїР° С‚РѕРіРѕ РІСЂРѕРґРµ**

**int mini = min(size1, size2);**

**for (i = 0; i != mini; i++) {**

**for (j = 0; j != mini; j++) {**

**new\_graph[i][j] = (graph1[i][j] | graph2[i][j]);**

**}**

**}**

**//if we have two different matrix we then just copy everything what's left to new graph**

**if (size1 != size2) {**

**for (i = 0; i != size; i++) {**

**for (j = mini; j != size; j++) {**

**// add a check to determin which graph is bigger..**

**if (size == size1) {**

**new\_graph[i][j] = graph1[i][j];**

**new\_graph[j][i] = new\_graph[i][j];**

**}**

**else {**

**new\_graph[i][j] = graph2[i][j];**

**new\_graph[j][i] = new\_graph[i][j];**

**}**

**}**

**}**

**}**

**return new\_graph;**

**}**

**vector<vector<int>> intersection(vector<vector<int>> graph1, vector<vector<int>> graph2, int size1, int size2, vector<int>\* nodes, int\* new\_size) {**

**int size = min(size1, size2);**

**vector<vector<int>> new\_graph(size, vector<int>(size));**

**for (int i = 0; i < size; i++) nodes->push\_back(i + 1);**

**\*new\_size = size;**

**for (int i = 0; i != size; i++) {**

**for (int j = i; j != size; j++) {**

**new\_graph[i][j] = (graph1[i][j] & graph2[i][j]);**

**new\_graph[j][i] = new\_graph[i][j];**

**}**

**}**

**return new\_graph;**

**}**

**vector<vector<int>> ring\_sum(vector<vector<int>> graph1, vector<vector<int>> graph2, int size1, int size2, vector<int>\* nodes, int\* new\_size) {**

**int size = max(size1, size2);**

**vector<vector<int>> new\_graph(size, vector<int>(size));**

**int mini = min(size1, size2);**

**for (int i = 0; i != mini; i++) {**

**for (int j = i; j != mini; j++) {**

**new\_graph[i][j] = (graph1[i][j] ^ graph2[i][j]); // this is a XOR (РѕС‚СЂРёС†Р°РЅРёРµ РР›Р)**

**new\_graph[j][i] = new\_graph[i][j];**

**}**

**}**

**// if graphs are not the same size, this code executes:**

**// it just fills evertyhting with leftovers of the bigger graph**

**if (size1 != size2) {**

**for (int i = 0; i != size; i++) {**

**for (int j = mini; j != size; j++) {**

**if (size == size1) {**

**new\_graph[i][j] = graph1[i][j];**

**new\_graph[j][i] = new\_graph[i][j];**

**}**

**else {**

**new\_graph[i][j] = graph2[i][j];**

**new\_graph[j][i] = new\_graph[i][j];**

**}**

**}**

**}**

**}**

**int temp\_size = size;**

**for (int i = 0; i < size; i++) nodes->push\_back(i + 1);**

**// I think we delete the leftover nodes and such here...**

**for (int i = 0; i != temp\_size; i++) {**

**// if node's connections are all zero, like, it is separeted from the graph..**

**if ((equal(new\_graph[i].begin() + 1, new\_graph[i].end(), new\_graph[i].begin())) && (new\_graph[i][0]) == 0) {**

**//we erase all traces of this node**

**for (int j = 0; j != temp\_size; j++) new\_graph[j].erase(new\_graph[j].begin() + i);**

**new\_graph.erase(new\_graph.begin() + i);**

**nodes->erase(nodes->begin() + i);**

**// now that we have a new graph, we reset the counter and check everything once again**

**// it might be actually useless, but idk...**

**temp\_size--;**

**i = 0;**

**}**

**}**

**\*new\_size = new\_graph.size();**

**return new\_graph;**

**}**

**vector<vector<int>> Cartesian\_product(vector<vector<int>> graph1, vector<vector<int>> graph2, int size1, int size2, vector<int>\* nodes, int\* new\_size) {**

**int x = 0, y = 0;**

**int temp = size1 \* size2;**

**vector<vector<int>> new\_graph(temp, vector<int>(temp));**

**for (int a = 0; a != size1; a++) {**

**for (int b = 0; b != size1; b++) {**

**for (int c = 0; c != size2; c++) {**

**for (int d = 0; d != size2; d++) {**

**if (((a == c) && (graph2[b][d] == 1)) || ((b == d) && (graph1[a][c] == 1))) {**

**new\_graph[x][y] = 1;**

**}**

**else {**

**new\_graph[x][y] = 0;**

**}**

**//new\_graph[x][y] = graph1[a][b] \* graph2[c][d];**

**y++;**

**}**

**}**

**x++;**

**y = 0;**

**}**

**}**

**\*new\_size = new\_graph.size();**

**for (int i = 0; i < \*new\_size; i++) nodes->push\_back(i + 1);**

**return new\_graph;**

**}**

**void save\_changes(vector<vector<int>> graph, int size, vector<int> nodes, vector<vector<int>>\* g, int\* s, vector<int>\* n) {**

**\*g = graph;**

**\*s = size;**

**\*n = nodes;**

**}**

**void save\_changes(list<list<int>> graph, int size, vector<int> nodes, list<list<int>>\* g, int\* s, vector<int>\* n) {**

**\*g = graph;**

**\*s = size;**

**\*n = nodes;**

**}**

**int main() {**

**///DEBUG**

**/\***

**vector<vector<int>> debug\_matrix ={{0,1,0,1,1},//1**

**{1,0,1,0,0},//2**

**{0,1,0,0,1},//3**

**{1,0,0,0,1},//4**

**{1,0,1,1,0},//5**

**};**

**list<list<int>> debug\_list = convert\_to\_adj\_list(debug\_matrix, 5);**

**\*/**

**///DEBUG**

**srand(time(0)); //for random, always has to be set at zero, uses system time**

**int size1, size2, choice, node1, node2; //choice; //input;**

**char input;**

**vector<int> nodesG1, nodesG2;**

**// this might go in a function actually**

**cout << "Enter the amount of first Graph's nodes" << endl;**

**cin >> size1;**

**vector<vector<int>> graph1 = create\_Graph(size1, &nodesG1);**

**cout << "Enter the amount of second Graph's nodes" << endl;**

**cin >> size2;**

**vector<vector<int>> graph2 = create\_Graph(size2, &nodesG2);**

**display\_Graph(graph1, size1, "no.1", nodesG1);**

**display\_Graph(graph2, size2, "no.2", nodesG2);**

**list<list<int>> adj\_list1 = convert\_to\_adj\_list(graph1, size1);**

**list<list<int>> adj\_list2 = convert\_to\_adj\_list(graph2, size2);**

**display\_adj\_list(adj\_list1, size1, "no.1", nodesG1);**

**display\_adj\_list(adj\_list2, size2, "no.2", nodesG2);**

**// I am not sure, but I'll just copy the values, and if the user chooses to save teh changes, then it'll overwrite them..**

**while (1) {**

**int t\_size1 = size1;**

**int t\_size2 = size2;**

**vector<vector<int>> t\_graph1 = graph1;**

**vector<vector<int>> t\_graph2 = graph2;**

**vector<int> t\_nodesG1 = nodesG1;**

**vector<int> t\_nodesG2 = nodesG2;**

**list<list<int>> t\_list1 = adj\_list1;**

**list<list<int>> t\_list2 = adj\_list2;**

**//template for new graph**

**int G3\_size;**

**vector<int> G3\_nodes;**

**vector<vector<int>> G3;**

**cout << "Choose, which operation you would like to perform:" << endl;**

**cout << "1 - closing nodes (matrix form)" << endl << "2 - closing nodes (adj. list form)" << endl;**

**cout << "3 - contracting nodes (matrix form)" << endl << "4 - contracting nodes (adj. list form)" << endl;**

**cout << "5 - splitting node (matrix form)" << endl << "6 - splitting node (adj. list form)" << endl << "----" << endl;**

**cout << "7 - merge graphs" << endl << "8 - intersect graphs" << endl << "9 - ring sum for graphs" << endl << "----" << endl;**

**cout << "c - cartesian product of graphs" << endl << "----" << endl;**

**//cout << "r - generate new graphs" << endl << endl;**

**cout << "q - quit the program" << endl << endl;**

**cin >> input;**

**switch (input) {**

**//1**

**case '1':**

**cout << "Choose the graph for the operation (1 or 2):" << endl;**

**cin >> choice;**

**cout << "Which nodes should be closed?" << endl;**

**cin >> node1;**

**cin >> node2;**

**if (node1 > node2) swap(node1, node2); //we ensure that node1 < node2. It's just so we don't do this later**

**if (choice == 1) {**

**t\_graph1 = close\_nodes(t\_graph1, &t\_size1, node1, node2, &t\_nodesG1);**

**display\_Graph(t\_graph1, t\_size1, "no.1, after operation 'closing' ", t\_nodesG1);**

**}**

**else {**

**t\_graph2 = close\_nodes(t\_graph2, &t\_size2, node1, node2, &t\_nodesG2);**

**display\_Graph(t\_graph2, t\_size2, "no.2, after operation 'closing' ", t\_nodesG2);**

**};**

**cout << "Would you like to save the changes? (Y / N)" << endl;**

**cin >> input;**

**switch (input) {**

**case 'y':**

**if (choice == 1) save\_changes(t\_graph1, t\_size1, t\_nodesG1, &graph1, &size1, &nodesG1);**

**else save\_changes(t\_graph2, t\_size2, t\_nodesG2, &graph2, &size2, &nodesG2);**

**case 'n':**

**break;**

**}**

**break;**

**//2**

**case '2':**

**cout << "Choose the graph for the operation (1 or 2):" << endl;**

**cin >> choice;**

**cout << "Which nodes should be closed?" << endl;**

**cin >> node1;**

**cin >> node2;**

**if (node1 > node2) swap(node1, node2); //we ensure that node1 < node2. It's just so we don't do this later**

**if (choice == 1) {**

**t\_list1 = close\_nodes\_adj(t\_list1, &t\_size1, node1, node2, &t\_nodesG1);**

**display\_adj\_list(t\_list1, t\_size1, "no.1, after operation 'closing' ", t\_nodesG1);**

**}**

**else {**

**t\_list2 = close\_nodes\_adj(t\_list2, &t\_size2, node1, node2, &t\_nodesG2);**

**display\_adj\_list(t\_list2, t\_size2, "no.2, after operation 'closing' ", t\_nodesG2);**

**};**

**cout << "Would you like to save the changes? (Y / N)" << endl;**

**cin >> input;**

**switch (input) {**

**case 'y':**

**if (choice == 1) save\_changes(t\_list1, t\_size1, t\_nodesG1, &adj\_list1, &size1, &nodesG1);**

**else save\_changes(t\_list2, t\_size2, t\_nodesG2, &adj\_list2, &size2, &nodesG2);**

**case 'n':**

**break;**

**}**

**break;**

**//3**

**case '3':**

**cout << "Choose the graph for the operation (1 or 2):" << endl;**

**cin >> choice;**

**cout << "Which nodes should be closed?" << endl;**

**cin >> node1;**

**cin >> node2;**

**if (node1 > node2) swap(node1, node2); //we ensure that node1 < node2. It's just so we don't do this later**

**if (choice == 1) {**

**t\_graph1 = contract\_nodes(t\_graph1, &t\_size1, node1, node2, &t\_nodesG1);**

**display\_Graph(t\_graph1, t\_size1, "no.1, after operation 'contraction' ", t\_nodesG1);**

**}**

**else {**

**t\_graph2 = contract\_nodes(t\_graph2, &t\_size2, node1, node2, &t\_nodesG2);**

**display\_Graph(t\_graph2, t\_size2, "no.2, after operation 'contraction' ", t\_nodesG2);**

**};**

**cout << "Would you like to save the changes? (Y / N)" << endl;**

**cin >> input;**

**switch (input) {**

**case 'y':**

**if (choice == 1) save\_changes(t\_graph1, t\_size1, t\_nodesG1, &graph1, &size1, &nodesG1);**

**else save\_changes(t\_graph2, t\_size2, t\_nodesG2, &graph2, &size2, &nodesG2);**

**case 'n':**

**break;**

**}**

**break;**

**//4**

**case '4':**

**cout << "Choose the graph for the operation (1 or 2):" << endl;**

**cin >> choice;**

**cout << "Which nodes should be contracted?" << endl;**

**cin >> node1;**

**cin >> node2;**

**if (node1 > node2) swap(node1, node2); //we ensure that node1 < node2. It's just so we don't do this later**

**if (choice == 1) {**

**t\_list1 = contract\_nodes\_adj(t\_list1, &t\_size1, node1, node2, &t\_nodesG1);**

**display\_adj\_list(t\_list1, t\_size1, "no.1, after operation 'contraction' ", t\_nodesG1);**

**}**

**else {**

**t\_list2 = close\_nodes\_adj(t\_list2, &t\_size2, node1, node2, &t\_nodesG2);**

**display\_adj\_list(t\_list2, t\_size2, "no.2, after operation 'contraction' ", t\_nodesG2);**

**};**

**cout << "Would you like to save the changes? (Y / N)" << endl;**

**cin >> input;**

**switch (input) {**

**case 'y':**

**if (choice == 1) save\_changes(t\_list1, t\_size1, t\_nodesG1, &adj\_list1, &size1, &nodesG1);**

**else save\_changes(t\_list2, t\_size2, t\_nodesG2, &adj\_list2, &size2, &nodesG2);**

**case 'n':**

**break;**

**}**

**break;**

**//5**

**case '5':**

**cout << "Choose the graph for the operation (1 or 2):" << endl;**

**cin >> choice;**

**cout << "Which node should be splitted?" << endl;**

**cin >> node1;**

**if (choice == 1) {**

**t\_graph1 = split\_nodes(t\_graph1, &t\_size1, node1, &t\_nodesG1);**

**display\_Graph(t\_graph1, t\_size1, "no.1, after operation 'splitting' ", t\_nodesG1);**

**}**

**else {**

**t\_graph2 = split\_nodes(t\_graph2, &t\_size2, node1, &t\_nodesG2);**

**display\_Graph(t\_graph2, t\_size2, "no.2, after operation 'splitting' ", t\_nodesG2);**

**};**

**cout << "Would you like to save the changes? (Y / N)" << endl;**

**cin >> input;**

**switch (input) {**

**case 'y':**

**if (choice == 1) save\_changes(t\_graph1, t\_size1, t\_nodesG1, &graph1, &size1, &nodesG1);**

**else save\_changes(t\_graph2, t\_size2, t\_nodesG2, &graph2, &size2, &nodesG2);**

**case 'n':**

**break;**

**}**

**break;**

**//6**

**case '6':**

**cout << "Choose the graph for the operation (1 or 2):" << endl;**

**cin >> choice;**

**cout << "Which node should be splitted?" << endl;**

**cin >> node1;**

**if (choice == 1) {**

**t\_list1 = split\_nodes\_adj(t\_list1, &t\_size1, node1, &t\_nodesG1);**

**display\_adj\_list(t\_list1, t\_size1, "no.1, after operation 'splitting' ", t\_nodesG1);**

**}**

**else {**

**t\_list2 = split\_nodes\_adj(t\_list2, &t\_size2, node1, &t\_nodesG2);**

**display\_adj\_list(t\_list2, t\_size2, "no.2, after operation 'splitting' ", t\_nodesG2);**

**};**

**cout << "Would you like to save the changes? (Y / N)" << endl;**

**cin >> input;**

**switch (input) {**

**case 'y':**

**if (choice == 1) save\_changes(t\_list1, t\_size1, t\_nodesG1, &adj\_list1, &size1, &nodesG1);**

**else save\_changes(t\_list2, t\_size2, t\_nodesG2, &adj\_list2, &size2, &nodesG2);**

**case 'n':**

**break;**

**}**

**break;**

**//7**

**case '7':**

**G3 = merge\_graphs(t\_graph1, t\_graph2, t\_size1, t\_size2, &G3\_nodes, &G3\_size);**

**display\_Graph(G3, G3\_size, "no.3, after merging two other graphs", G3\_nodes);**

**cout << endl;**

**break;**

**//8**

**case '8':**

**G3 = intersection(t\_graph1, t\_graph2, t\_size1, t\_size2, &G3\_nodes, &G3\_size);**

**display\_Graph(G3, G3\_size, "no.3, after intersecting two other graphs", G3\_nodes);**

**cout << endl;**

**break;**

**//9**

**case '9':**

**G3 = ring\_sum(t\_graph1, t\_graph2, t\_size1, t\_size2, &G3\_nodes, &G3\_size);**

**display\_Graph(G3, G3\_size, "no.3, which is a ring sum of two other graphs", G3\_nodes);**

**cout << endl;**

**break;**

**//10**

**case 'c':**

**G3 = Cartesian\_product(t\_graph1, t\_graph2, t\_size1, t\_size2, &G3\_nodes, &G3\_size);**

**display\_Graph(G3, G3\_size, "no.3, which is a Cartesian product of two other graphs", G3\_nodes);**

**cout << endl;**

**break;**

**case 'q':**

**cout << "Terminating the process...";**

**return 0;**

**//case 'r':**

**default:**

**cout << "Incorrect input, try again" << endl << endl;**

**fflush(stdin);**

**//cin >> ws;**

**//cin.clear();**

**//cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');**

**break;**

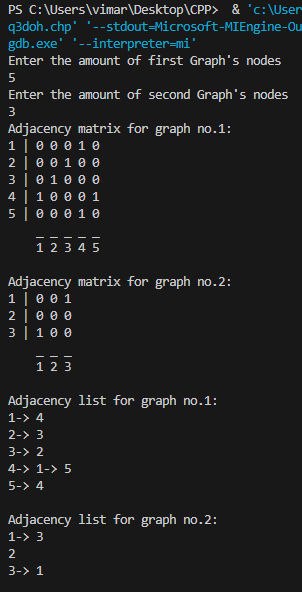
**}**

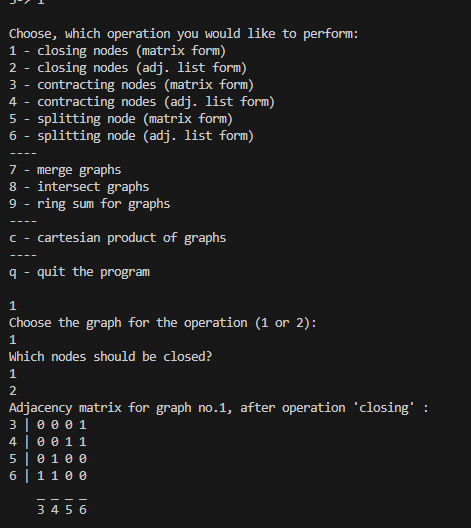
**}**

**return 0;**

**}**

**Результаты работы программы**





**Вывод**

В ходе выполнения заданий был успешно написан код, осуществляющий унарные и бинарные операции над графами и их вершинами.