Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по теме «Унарные и бинарные операции над графами»

Выполнили:

Студенты группы 21ВВ2

Назаров Е.А.

Макаров И.С.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза, 2022

**Цель работы:**

Исследование унарных, бинарных операций над графами и приобретение практических навыков решения задач с использованием основ теории графов.

**Лабораторные работы:**

**Задание:**

**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 https://lh5.googleusercontent.com/L_5gfz8876Ab2u0x_bXk2V2HkE9jl2Rw_yMDOwONwHywiAO_7hOKeD8IaEuHHpuCyI5btpgBBisHfpggXgTnrZh2FMN9vpZ1SbB1IM8RStloXJXVVCIsbcj0vmTkrSoKDaRNCkRjuUTYsk2N0WK26Aizk85gmlk0dTOOGRMeh6IZpnAMaC3LWe1mxfvAm3PEc-RAEw *G*2

б) пересечения  *G* = *G*1 https://lh3.googleusercontent.com/s8omopK4mw7F2NmG1bjJyrgjLEPWUWssAttwd3a1oAD2Kb_mdOdrfOL9BWjRgZJxjwN4iaaOgcxbq4-07fLn3KFzywp68Seks0vCaGMEHmZpLB6Ozmpusftp1W3bC2vfoteTctyaS6d15iVf_6_Ys7iycjlPLQz0D12wc8AY0D8I53ncxLRVVhHic0Med3IZAcnMTA *G*2

в) кольцевой суммы *G* = *G*1 https://lh4.googleusercontent.com/icQUnFEbZ5A5kDEair-FW6aoFkMqcsH6rldELs5wOEg4t_yjDUAP6jodBG_1Wod84NugOOVU3-saWc2zZXuuCh_coC_KHq3s7f5693UncsXwnqDW4tu2fnn_hZHSHZv70qfcdPqBcUIQYwtQhT3WTiF_IMY-1TMMOE_Yfb-zCC9ymYK-FDLppz7qnml0wx8_GlidgQ *G*2

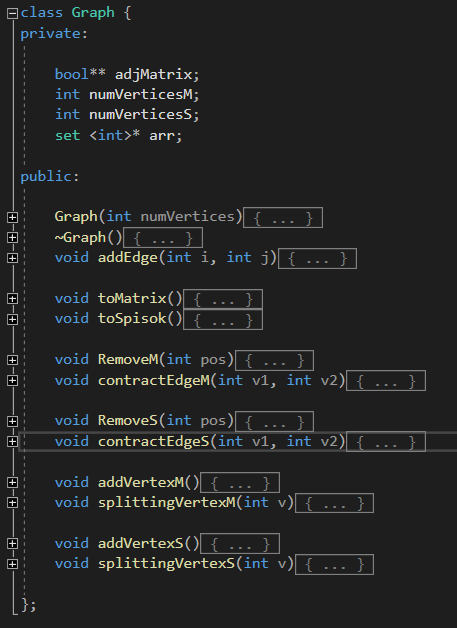
Результат выполнения операции выведите на экран.

**Задание 4 \***

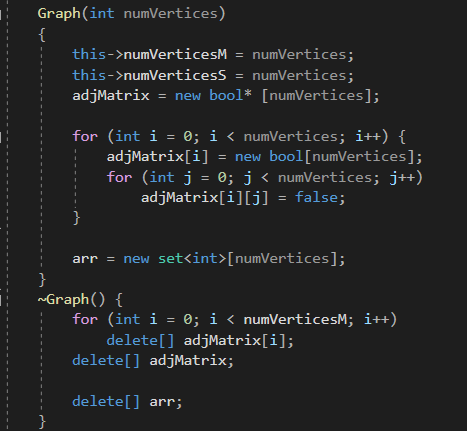
1. Для матричной формы представления графов выполните операцию декартова произведения графов *G = G*1X *G*2.

Результат выполнения операции выведите на экран.

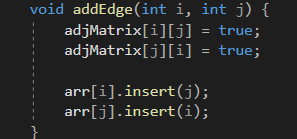
**Пояснительный текст к программе:**



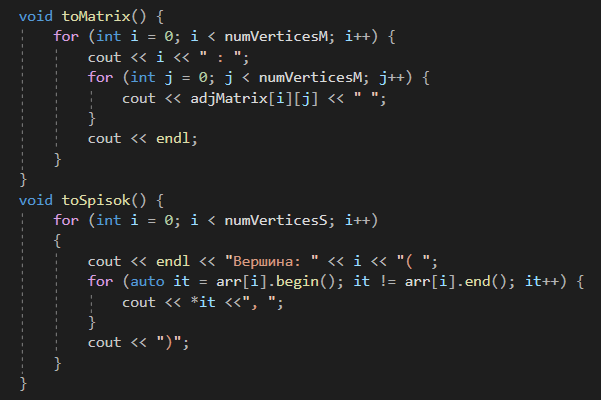
В первых двух заданиях использовался класс Graph. В public части показаны все реализованные функции.



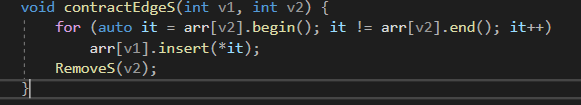
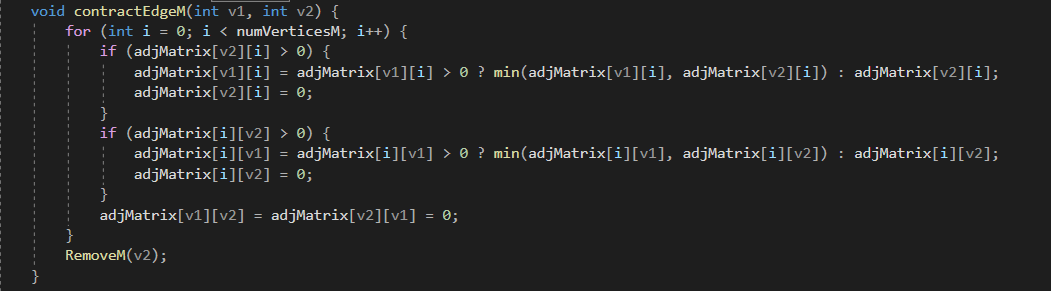
Реализация конструктора и деструктора в классе. Задание и освобождения динамических структур данных для работы. В двухмерном динамическом массиве хранится информация о графе в виде матрицы смежности, в динамическом списке set(STL библиотека), хранится информация о графе в виде списка смежности.



Функция добавления вершин и смежных с ней вершинами. Добавление реализовано в виде неориентированных графов.

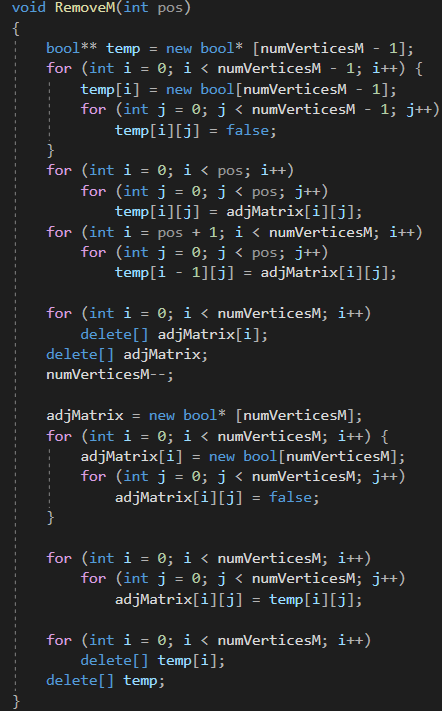
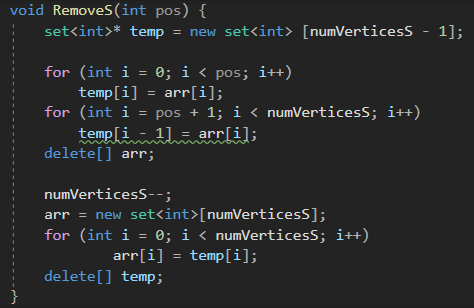


Функции вывода информации о графе в виде матрицы смежности и списка смежности.

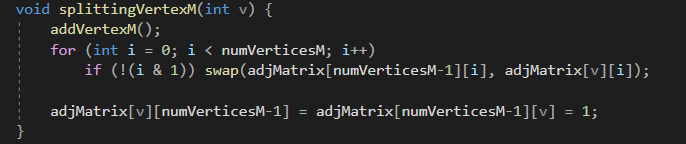


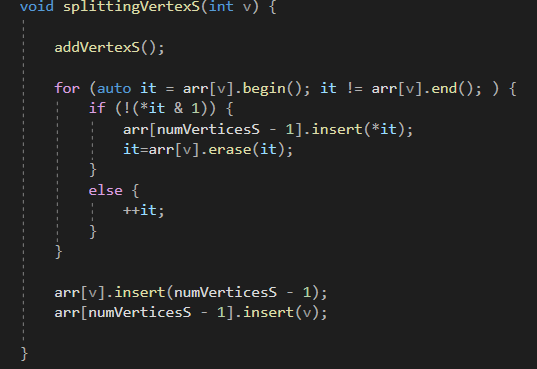
Функции для отожествления вершин и стягивания ребра для графов, представленных в виде матрице смежности и списка смежности.

Задача функции состоит в том, что всё окружение вершины, которая входит второй в функцию, присваивается первой, после чего эта вершина удаляется.

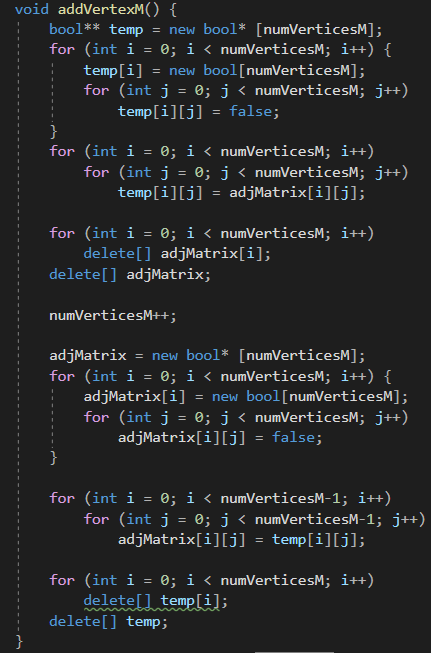
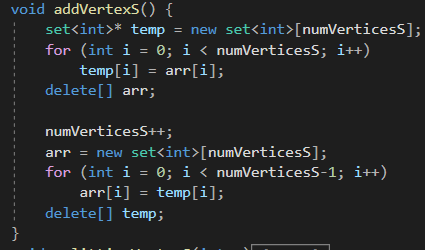


Функции служат для того, чтобы удалить вершину из графа представленном в виде матрицы смежности и списка смежности.

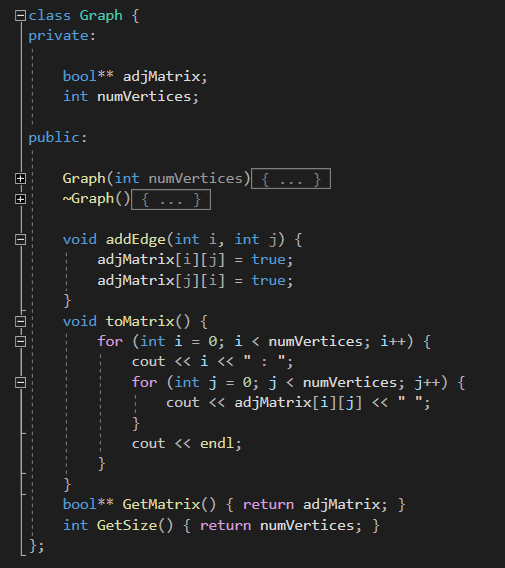




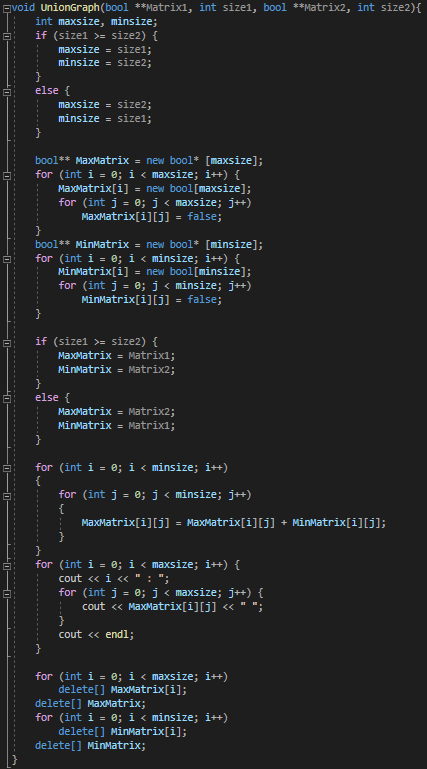
Функции для реализации операции расщепления вершины. Суть задачи состоит в том, чтобы одну вершину разбить на две и каждой в хаотичном порядке передать часть окружения первоначальной вершины.



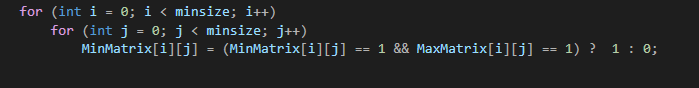
Функции служат для того, чтобы добавить вершину из графа представленном в виде матрицы смежности и списка смежности



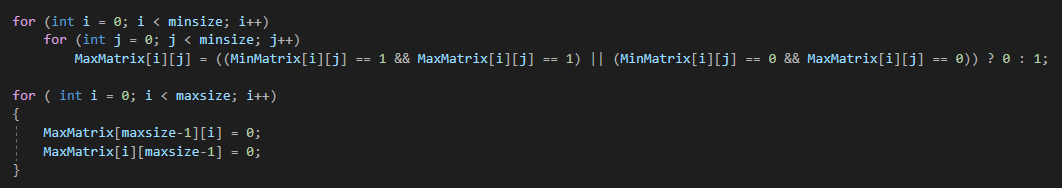
Класс Graph для реализаций 3 и 4 заданий



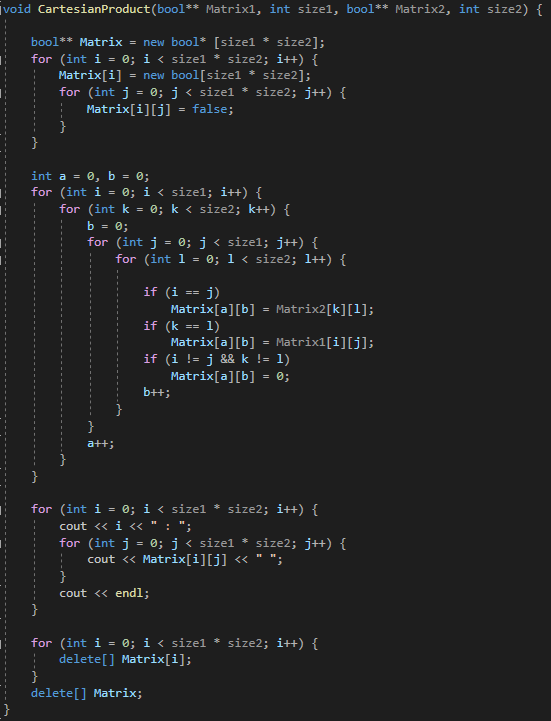
Объединение. Функция получает да вход два графа в виде матрицы смежности и размер этих матриц. После чего выбирается по размеру наименьшая и наибольшая матрица для корректной работы логики в программе. Отожествление вершин, ничто иначе как сложение матриц.



Пересечение. Эта функция тоже выбирает наибольший по размеру массив, после чего происходит операция пересечения. Если два ребра в графах совпадают, то связь этих вершин сохраняется.

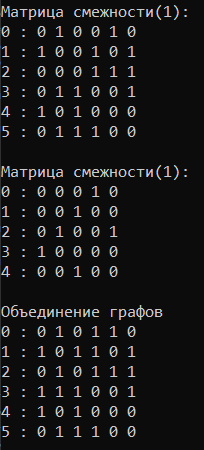
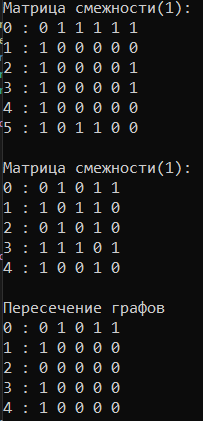
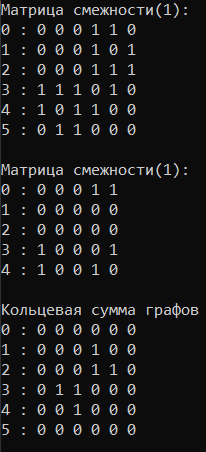


Эта функция тоже выбирает наибольший по размеру массив, после чего происходит операция кольцевой суммы. Если два ребра в графах совпадают, то связь этих вершин удаляются. Все оставшиеся ребра сохраняются.

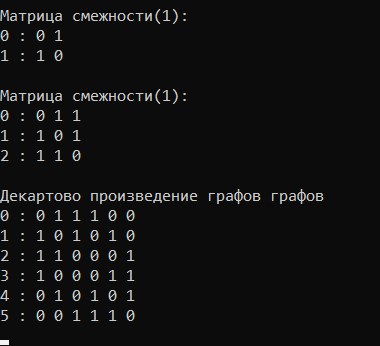


Декартово произведение графов представляет собой схожую по системе операцию- прямое произведение матриц. Прямым (декартовым) произведением графовG1(X1,Г1X1) и G2(X2,Г2X2) называется граф G(X,ГX), для которого X=X1http://www.e-biblio.ru/book/bib/02_estestv_nauki/Diskretnaj_matematika/Book/docs/piece010.files/image115.gifX2 и ГX=Г1X1http://www.e-biblio.ru/book/bib/02_estestv_nauki/Diskretnaj_matematika/Book/docs/piece010.files/image115.gifГ2X2. По этой логике и работает функция.

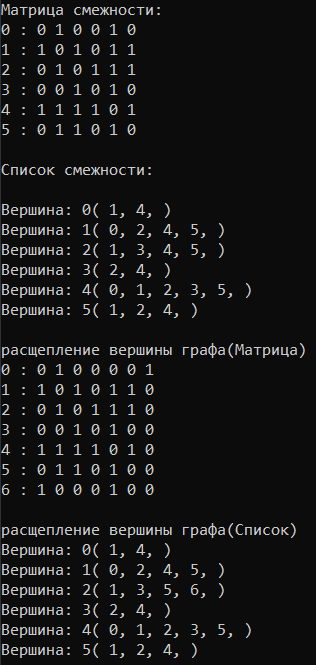
**Результат работы:**

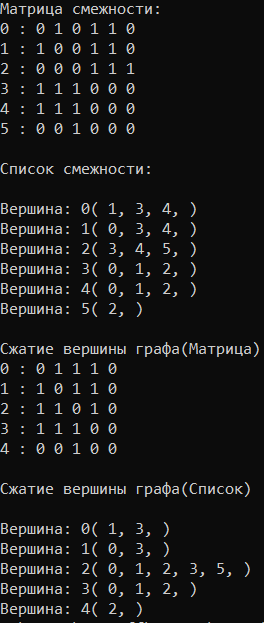
Результат выполнения простых бинарных операций над графами.



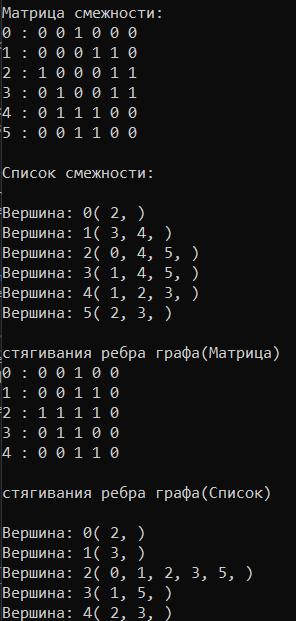
Результат выполнения декартового произведения.



Пример показан на расщеплении 2 вершины графа.



Пример показан на сжатие 2 и 4 вершин графа.



Пример показан на стягивании ребра 2 и 4 вершин графа. Ребро соединявшие вершины, становится петлей для итоговой вершины.

**Листинг:**

**Lab5.1**

#include <iostream>

#include <set>

#include <iterator>

using namespace std;

class Graph {

private:

bool\*\* adjMatrix;

int numVerticesM;

int numVerticesS;

set <int>\* arr;

public:

Graph(int numVertices)

{

this->numVerticesM = numVertices;

this->numVerticesS = numVertices;

adjMatrix = new bool\* [numVertices];

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

adjMatrix[i] = new bool[numVertices];

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

adjMatrix[i][j] = false;

}

arr = new set<int>[numVertices];

}

~Graph() {

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

delete[] adjMatrix[i];

delete[] adjMatrix;

delete[] arr;

}

void addEdge(int i, int j) {

adjMatrix[i][j] = true;

adjMatrix[j][i] = true;

arr[i].insert(j);

arr[j].insert(i);

}

void toMatrix() {

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

cout << i << " : ";

for (int j = 0; j < numVerticesM; j++) {

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

}

cout << endl;

}

}

void toSpisok() {

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

{

cout << endl << "Вершина: " << i << "( ";

for (auto it = arr[i].begin(); it != arr[i].end(); it++) {

cout << \*it <<", ";

}

cout << ")";

}

}

void RemoveM(int pos)

{

bool\*\* temp = new bool\* [numVerticesM - 1];

for (int i = 0; i < numVerticesM - 1; i++) {

temp[i] = new bool[numVerticesM - 1];

for (int j = 0; j < numVerticesM - 1; j++)

temp[i][j] = false;

}

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

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

temp[i][j] = adjMatrix[i][j];

for (int i = pos + 1; i < numVerticesM; i++)

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

temp[i - 1][j] = adjMatrix[i][j];

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

delete[] adjMatrix[i];

delete[] adjMatrix;

numVerticesM--;

adjMatrix = new bool\* [numVerticesM];

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

adjMatrix[i] = new bool[numVerticesM];

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

adjMatrix[i][j] = false;

}

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

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

adjMatrix[i][j] = temp[i][j];

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

delete[] temp[i];

delete[] temp;

}

void contractEdgeM(int v1, int v2) {

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

if (adjMatrix[v2][i] > 0) {

adjMatrix[v1][i] = adjMatrix[v1][i] > 0 ? min(adjMatrix[v1][i], adjMatrix[v2][i]) : adjMatrix[v2][i];

adjMatrix[v2][i] = 0;

}

if (adjMatrix[i][v2] > 0) {

adjMatrix[i][v1] = adjMatrix[i][v1] > 0 ? min(adjMatrix[i][v1], adjMatrix[i][v2]) : adjMatrix[i][v2];

adjMatrix[i][v2] = 0;

}

adjMatrix[v1][v2] = adjMatrix[v2][v1] = 0;

}

RemoveM(v2);

}

void RemoveS(int pos) {

set<int>\* temp = new set<int> [numVerticesS - 1];

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

temp[i] = arr[i];

for (int i = pos + 1; i < numVerticesS; i++)

temp[i - 1] = arr[i];

delete[] arr;

numVerticesS--;

arr = new set<int>[numVerticesS];

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

arr[i] = temp[i];

delete[] temp;

}

void contractEdgeS(int v1, int v2) {

for (auto it = arr[v2].begin(); it != arr[v2].end(); it++)

arr[v1].insert(\*it);

RemoveS(v2);

}

void addVertexM() {

bool\*\* temp = new bool\* [numVerticesM];

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

temp[i] = new bool[numVerticesM];

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

temp[i][j] = false;

}

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

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

temp[i][j] = adjMatrix[i][j];

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

delete[] adjMatrix[i];

delete[] adjMatrix;

numVerticesM++;

adjMatrix = new bool\* [numVerticesM];

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

adjMatrix[i] = new bool[numVerticesM];

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

adjMatrix[i][j] = false;

}

for (int i = 0; i < numVerticesM-1; i++)

for (int j = 0; j < numVerticesM-1; j++)

adjMatrix[i][j] = temp[i][j];

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

delete[] temp[i];

delete[] temp;

}

void splittingVertexM(int v) {

addVertexM();

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

if (!(i & 1)) swap(adjMatrix[numVerticesM-1][i], adjMatrix[v][i]);

adjMatrix[v][numVerticesM-1] = adjMatrix[numVerticesM-1][v] = 1;

}

void addVertexS() {

set<int>\* temp = new set<int>[numVerticesS];

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

temp[i] = arr[i];

delete[] arr;

numVerticesS++;

arr = new set<int>[numVerticesS];

for (int i = 0; i < numVerticesS-1; i++)

arr[i] = temp[i];

delete[] temp;

}

void splittingVertexS(int v) {

addVertexS();

for (auto it = arr[v].begin(); it != arr[v].end(); ) {

if (!(\*it & 1)) {

arr[numVerticesS - 1].insert(\*it);

it=arr[v].erase(it);

}

else {

++it;

}

}

arr[v].insert(numVerticesS - 1);

arr[numVerticesS - 1].insert(v);

}

};

int main()

{

srand(time(NULL));

setlocale(0, "");

int z = 6;

Graph c(z);

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

{

for (int j = i + 1; j < z; j++)

{

if (bool(rand() % 2))

c.addEdge(i, j);

}

}

cout << endl << "Матрица смежности: \n";

c.toMatrix();

cout << endl << "Cписок смежности: \n";

c.toSpisok();

//cout << endl << endl << "Сжатие вершины-ребра графа(Матрица)"<<endl;

//c.contractEdgeM(2, 4);

//c.toMatrix();

//cout << endl << endl << "расщеплениe вершины графа(Матрица)"<<endl;

//c.splittingVertexM(0);

//c.toMatrix();

//cout << endl << "Сжатие вершины-ребра графа(Список)"<<endl;

//c.contractEdgeS(2, 4);

//c.toSpisok();

//cout << endl << endl << "расщеплениe вершины графа(Список)" << endl;

//c.splittingVertexS(2);

//c.toSpisok();

return 0;

}

**Lab5.2**

#include <iostream>

using namespace std;

class Graph {

private:

bool\*\* adjMatrix;

int numVertices;

public:

Graph(int numVertices)

{

this->numVertices = numVertices;

adjMatrix = new bool\* [numVertices];

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

adjMatrix[i] = new bool[numVertices];

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

adjMatrix[i][j] = false;

}

}

~Graph() {

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

delete[] adjMatrix[i];

delete[] adjMatrix;

}

void addEdge(int i, int j) {

adjMatrix[i][j] = true;

adjMatrix[j][i] = true;

}

void toMatrix() {

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

cout << i << " : ";

for (int j = 0; j < numVertices; j++) {

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

}

cout << endl;

}

}

bool\*\* GetMatrix() { return adjMatrix; }

int GetSize() { return numVertices; }

};

void UnionGraph(bool\*\* Matrix1, int size1, bool\*\* Matrix2, int size2) {

int maxsize, minsize;

if (size1 >= size2) {

maxsize = size1;

minsize = size2;

}

else {

maxsize = size2;

minsize = size1;

}

bool\*\* MaxMatrix = new bool\* [maxsize];

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

MaxMatrix[i] = new bool[maxsize];

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

MaxMatrix[i][j] = false;

}

bool\*\* MinMatrix = new bool\* [minsize];

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

MinMatrix[i] = new bool[minsize];

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

MinMatrix[i][j] = false;

}

if (size1 >= size2) {

MaxMatrix = Matrix1;

MinMatrix = Matrix2;

}

else {

MaxMatrix = Matrix2;

MinMatrix = Matrix1;

}

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

{

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

{

MaxMatrix[i][j] = MaxMatrix[i][j] + MinMatrix[i][j];

}

}

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

cout << i << " : ";

for (int j = 0; j < maxsize; j++) {

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

}

cout << endl;

}

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

delete[] MaxMatrix[i];

delete[] MaxMatrix;

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

delete[] MinMatrix[i];

delete[] MinMatrix;

}

void IntersectionCraph(bool\*\* Matrix1, int size1, bool\*\* Matrix2, int size2) {

int maxsize, minsize;

if (size1 >= size2) {

maxsize = size1;

minsize = size2;

}

else {

maxsize = size2;

minsize = size1;

}

bool\*\* MaxMatrix = new bool\* [maxsize];

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

MaxMatrix[i] = new bool[maxsize];

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

MaxMatrix[i][j] = false;

}

bool\*\* MinMatrix = new bool\* [minsize];

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

MinMatrix[i] = new bool[minsize];

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

MinMatrix[i][j] = false;

}

if (size1 >= size2) {

MaxMatrix = Matrix1;

MinMatrix = Matrix2;

}

else {

MaxMatrix = Matrix2;

MinMatrix = Matrix1;

}

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

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

MinMatrix[i][j] = (MinMatrix[i][j] == 1 && MaxMatrix[i][j] == 1) ? 1 : 0;

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

cout << i << " : ";

for (int j = 0; j < minsize; j++) {

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

}

cout << endl;

}

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

delete[] MaxMatrix[i];

delete[] MaxMatrix;

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

delete[] MinMatrix[i];

delete[] MinMatrix;

}

void RingSumGraph(bool\*\* Matrix1, int size1, bool\*\* Matrix2, int size2) {

int maxsize, minsize;

if (size1 >= size2) {

maxsize = size1;

minsize = size2;

}

else {

maxsize = size2;

minsize = size1;

}

bool\*\* MaxMatrix = new bool\* [maxsize];

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

MaxMatrix[i] = new bool[maxsize];

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

MaxMatrix[i][j] = false;

}

bool\*\* MinMatrix = new bool\* [minsize];

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

MinMatrix[i] = new bool[minsize];

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

MinMatrix[i][j] = false;

}

if (size1 >= size2) {

MaxMatrix = Matrix1;

MinMatrix = Matrix2;

}

else {

MaxMatrix = Matrix2;

MinMatrix = Matrix1;

}

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

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

MaxMatrix[i][j] = ((MinMatrix[i][j] == 1 && MaxMatrix[i][j] == 1) || (MinMatrix[i][j] == 0 && MaxMatrix[i][j] == 0)) ? 0 : 1;

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

{

MaxMatrix[maxsize - 1][i] = 0;

MaxMatrix[i][maxsize - 1] = 0;

}

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

cout << i << " : ";

for (int j = 0; j < maxsize; j++) {

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

}

cout << endl;

}

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

delete[] MaxMatrix[i];

delete[] MaxMatrix;

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

delete[] MinMatrix[i];

delete[] MinMatrix;

}

void CartesianProduct(bool\*\* Matrix1, int size1, bool\*\* Matrix2, int size2) {

bool\*\* Matrix = new bool\* [size1 \* size2];

for (int i = 0; i < size1 \* size2; i++) {

Matrix[i] = new bool[size1 \* size2];

for (int j = 0; j < size1 \* size2; j++) {

Matrix[i][j] = false;

}

}

int a = 0, b = 0;

for (int i = 0; i < size1; i++) {

for (int k = 0; k < size2; k++) {

b = 0;

for (int j = 0; j < size1; j++) {

for (int l = 0; l < size2; l++) {

if (i == j)

Matrix[a][b] = Matrix2[k][l];

if (k == l)

Matrix[a][b] = Matrix1[i][j];

if (i != j && k != l)

Matrix[a][b] = 0;

b++;

}

}

a++;

}

}

for (int i = 0; i < size1 \* size2; i++) {

cout << i << " : ";

for (int j = 0; j < size1 \* size2; j++) {

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

}

cout << endl;

}

for (int i = 0; i < size1 \* size2; i++) {

delete[] Matrix[i];

}

delete[] Matrix;

}

int main(void)

{

srand(time(NULL));

setlocale(0, "");

int z = 2, x = 3;

Graph c1(z);

Graph c2(x);

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

{

for (int j = i + 1; j < z; j++)

{

if (bool(rand() % 2))

c1.addEdge(i, j);

}

}

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

{

for (int j = i + 1; j < x; j++)

{

if (bool(rand() % 2))

c2.addEdge(i, j);

}

}

cout << endl << "Матрица смежности(1): \n";

c1.toMatrix();

cout << endl << "Матрица смежности(1): \n";

c2.toMatrix();

//cout << endl << "Объединение графов"<<endl;

//UnionGraph(c1.GetMatrix(), c1.GetSize(), c2.GetMatrix(), c2.GetSize());

//cout << endl << "Пересечение графов" << endl;

//IntersectionCraph(c1.GetMatrix(), c1.GetSize(), c2.GetMatrix(), c2.GetSize());

//cout << endl << "Кольцевая сумма графов" << endl;

//RingSumGraph(c1.GetMatrix(), c1.GetSize(), c2.GetMatrix(), c2.GetSize());

cout << endl << "Декартово произведение графов графов" << endl;

CartesianProduct(c1.GetMatrix(), c1.GetSize(), c2.GetMatrix(), c2.GetSize());

cin.get();

return 0;

}

**Вывод:** Мы научились выполнять унарные, бинарные операций над графами и приобрели практические навыки решения задач с использованием основ теории графов.