Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «унарные и бинарные операции над графами»

Выполнил:

студенты группы 22ВВВ1

Матросов А.М.

Пяткин М.Э.

Принял:

к.т.н. доцент Юрова О.В.

к.э.н. доцент Акифьев И.В.

Пенза 2023

**Общие сведения**

Все унарные операции над графами можно объединить в две группы. Первую группу составляют операции, с помощью которых из исходного графа *G*1*,* можно построить граф *G*2 с меньшим числом элементов. В группу входят операции удаления ребра или вершины, отождествления вершин, стягивание ребра. Вторую группу составляют операции, позволяющие строить графы с большим числом элементов. В группу входят операции расщепления вершин, добавления ребра.

*Отождествление вершин.* В графе *G*1 выделяются вершины *и,v.* Определяют окружение *Q*1 вершины *u*,и окружение *Q*2 вершины *v,* вычисляют их объединение *Q* = *Q1* * Q2.* Затем над графом *G*1 выполняются следующие преобразования:

* из графа *G*1 удаляют вершины *u,* *v (H*1 *= G*1 *- u - v);*
* к графу *Н*1присоединяют новую вершину *z (H*1 *= H*1 *+z);*
* вершину *z* соединяют ребром с каждой из вершин *w*1*Q*

*(G*2 *= H*1 *+ zwi*, *i =* 1,2,3*,…).*

*Стягивание ребра.* Данная операция является операцией отождествления смежных вершин *и, v* в графе *G*1.

Наиболее важными бинарными операциями являются: объединение, пересечение, декартово произведение и кольцевая сумма.

*Объединение.* Граф *G* называется объединением или наложением графов *G*1 и *G*2, если *VG = V*1*V*2*; UG = U*1* U*2 (рис. 1).

**U**

*v*1

*v*2

*v*3

*v*4

*v*3

*v*4

*v*5

*v*2

*v*1

*v*3

*v*4

*v*5

Рис. 1. Объединение графов *G*1, *G*2

Объединение графов *G*1 и *G*2 называется дизъюнктным, если *V*1*V*2 *= *. При дизъюнктном объединении никакие два из объединяемых графов не должны иметь общих вершин.

*Пересечение.* Граф *G* называется пересечением графов *G*1, *G*2,если *VG = V*1*V*2и *UG = U*1*U*2 (риc.2). Операция "пересечения" записывается следующим образом: *G = G*1*G*2*.*

**∩**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*6

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

*v*3

*v*2

Рис.2. Пересечение графов *G*1, *G*2*.*

*Декартово произведение.* Граф *G* называется декартовым произведением графов *G*1 и *G*2 если *VG* = *V*1*V*2 —декартово произведение множеств вершин графов *G*1, *G*2, а множество ребер *U*c задается следующим образом: вершины (*zi*, *vk*) и (*zj*, *vl*) смежны в графе *G* тогда и только тогда, когда *zi* = *zj*(*i* = *j*), a *v*k и *vl* смежны в *G*2 или *vk* = *vl*(*k* = *l*), смежны в графе *G*1 (см. рис.3).

**X**

*z*1

*z*2

*v*1

*v*3

*v*2

*z*1*v*1

*z*1*v*2

*z*1*v*3

*z*2*v*1

*z*2*v*2

*z*2*v*3

Рис. 3. Декартово произведение графов *G*1, *G*2

*Кольцевая сумма* графов представляет граф, который не имеет изолированных вершин и состоит из ребер, присутствующих либо в первом исходном графе, либо во втором. Кольцевая сумма определяется следующим соотношением: *G* = *G*1  *G*2 (рис.4).

**⊕**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

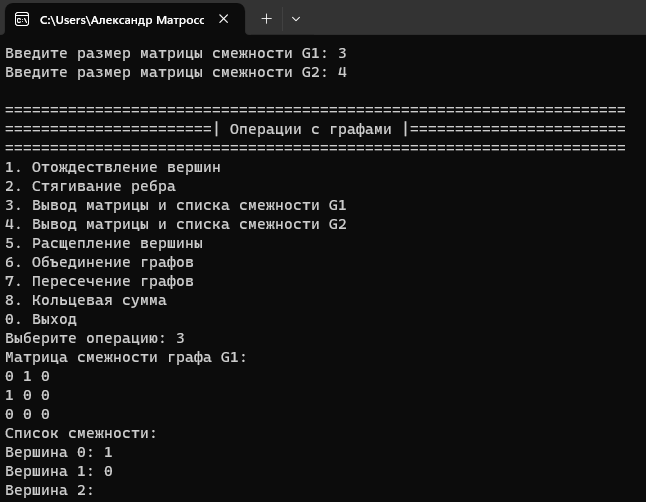
*v*3

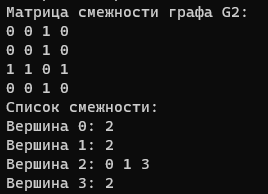
*v*2

Рис.4. Кольцевая сумма графов *G1, G2*

### 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:

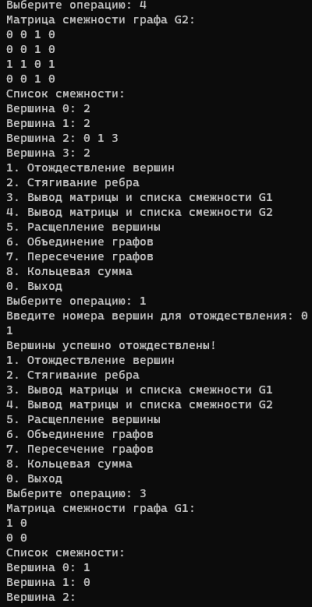
а) отождествления вершин

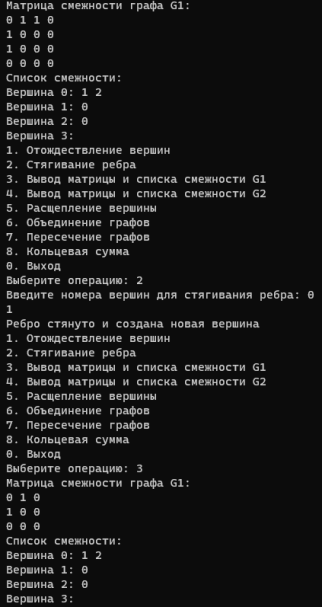
б) стягивания ребра

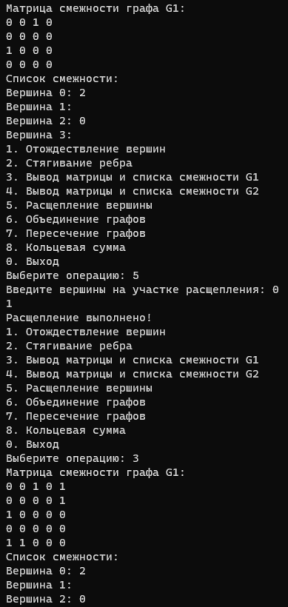
в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.







**Задание 3**

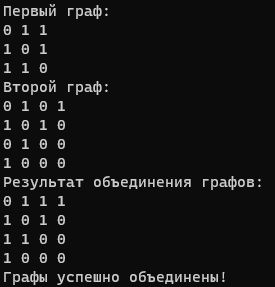
1. Для матричной формы представления графов выполните операцию:

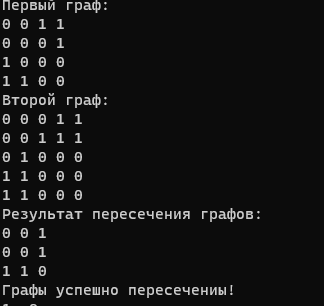
а) объединения *G* = *G*1  *G*2

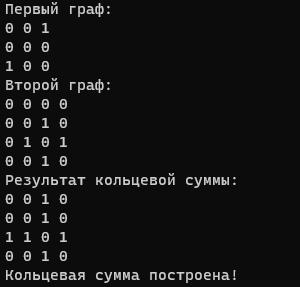
б) пересечения *G* = *G*1  *G*2

в) кольцевой суммы *G* = *G*1  *G*2

Результат выполнения операции выведите на экран.







### Вывод

В ходе выполнения лабораторной работы была реализована программа, генерирующая матрицу смежности для неориентированного графа и список смежности.

Была реализована программа, выполняющая такие операции, как стягивание ребра, отождествление вершин, расщепление вершины, объединение графов, пересечение графов и кольцевая сумма.

### Листинг

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

using namespace std;

// Функция для генерации случайного числа в заданном диапазоне

int getRandomNumber(int min, int max) {

return min + rand() % (max - min + 1);

}

// Функция для удаления вершины и инцидентных ей ребер из матрицы смежности

void removeVertex(vector<vector<int>>& matrix, int vertex) {

int size = matrix.size();

// Удаляем вершину и ее ребра

matrix.erase(matrix.begin() + vertex);

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

matrix[i].erase(matrix[i].begin() + vertex);

}

}

// Функция для отождествления двух вершин графа

void identifyVertices(vector<vector<int>>& matrix, int vertex1, int vertex2) {

int size = matrix.size();

// Отождествляем вершины vertex1 и vertex2

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

matrix[vertex1][i] = matrix[vertex1][i] || matrix[vertex2][i];

}

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

matrix[i][vertex1] = matrix[i][vertex1] || matrix[i][vertex2];

}

// Удаляем вершину vertex2

removeVertex(matrix, vertex2);

}

void contractEdge(vector<vector<int>>& matrix, int vertex1, int vertex2) {

int size = matrix.size();

matrix[vertex1][vertex2] = 0;

matrix[vertex2][vertex1] = 0;

identifyVertices(matrix, vertex1, vertex2);

}

void abiba(vector<vector<int>>& matrix, int vertex1, int vertex2) {

int size = matrix.size();

// Создаем новую матрицу смежности с дополнительной вершиной

vector<vector<int>> newMatrix(size + 1, vector<int>(size + 1, 0));

// Копируем значения из старой матрицы в новую матрицу

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

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

newMatrix[i][j] = matrix[i][j];

}

}

// Соединяем новую вершину с указанными вершинами

newMatrix[vertex1][size] = newMatrix[size][vertex1] = 1;

newMatrix[vertex2][size] = newMatrix[size][vertex2] = 1;

// Заменяем старую матрицу на новую матрицу

matrix = newMatrix;

}

// Функция для вывода матрицы смежности на консоль

void printMatrix(const vector<vector<int>>& matrix) {

int size = matrix.size();

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

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

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

}

cout << endl;

}

}

// Функция для объединения двух графов через логическое ИЛИ

void mergeGraphs(vector<vector<int>>& matrix1, const vector<vector<int>>& matrix2) {

int size1 = matrix1.size();

int size2 = matrix2.size();

// Создаем новую матрицу смежности с размером, равным максимальному размеру графов

int maxSize = max(size1, size2);

vector<vector<int>> newMatrix(maxSize, vector<int>(maxSize, 0));

// Копируем значения из первой матрицы в новую матрицу

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

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

newMatrix[i][j] = matrix1[i][j];

}

}

// Добавляем вершины и ребра из второй матрицы в новую матрицу

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

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

newMatrix[i][j] = newMatrix[i][j] || matrix2[i][j];

}

}

cout << "Первый граф:" << endl;

printMatrix(matrix1);

cout << "Второй граф:" << endl;

printMatrix(matrix2);

cout << "Результат объединения графов:" << endl;

printMatrix(newMatrix);

cout << "Графы успешно объединены!" << endl;

// Заменяем первую матрицу на новую матрицу

matrix1 = newMatrix;

}

void intersectionGraphs(vector<vector<int>>& matrix1, const vector<vector<int>>& matrix2) {

int size1 = matrix1.size();

int size2 = matrix2.size();

// Создаем новую матрицу смежности с размером, равным максимальному размеру графов

int maxSize = max(size1, size2);

vector<vector<int>> newMatrix(maxSize, vector<int>(maxSize, 0));

// Копируем значения из первой матрицы в новую матрицу

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

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

newMatrix[i][j] = matrix1[i][j];

}

}

// Добавляем вершины и ребра из второй матрицы в новую матрицу

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

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

newMatrix[i][j] = newMatrix[i][j] && matrix2[i][j];

}

}

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

bool isolated = true;

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

if (newMatrix[i][j] != 0 || newMatrix[j][i] != 0) {

isolated = false;

break;

}

}

if (isolated) {

removeVertex(newMatrix, i);

maxSize--;

i--;

}

}

cout << "Первый граф:" << endl;

printMatrix(matrix1);

cout << "Второй граф:" << endl;

printMatrix(matrix2);

cout << "Результат пересечения графов:" << endl;

printMatrix(newMatrix);

cout << "Графы успешно пересечениы!" << endl;

// Заменяем первую матрицу на новую матрицу

matrix1 = newMatrix;

}

void XORGraphs(vector<vector<int>>& matrix1, const vector<vector<int>>& matrix2) {

int size1 = matrix1.size();

int size2 = matrix2.size();

// Создаем новую матрицу смежности с размером, равным максимальному размеру графов

int maxSize = max(size1, size2);

vector<vector<int>> newMatrix(maxSize, vector<int>(maxSize, 0));

// Копируем значения из первой матрицы в новую матрицу

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

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

newMatrix[i][j] = matrix1[i][j];

}

}

// Добавляем вершины и ребра из второй матрицы в новую матрицу

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

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

newMatrix[i][j] = newMatrix[i][j] xor matrix2[i][j];

}

}

cout << "Первый граф:" << endl;

printMatrix(matrix1);

cout << "Второй граф:" << endl;

printMatrix(matrix2);

cout << "Результат кольцевой суммы:" << endl;

printMatrix(newMatrix);

cout << "Кольцевая сумма построена!" << endl;

// Заменяем первую матрицу на новую матрицу

matrix1 = newMatrix;

}

vector<vector<int>> generateAdjacencyList(const vector<vector<int>>& adjacencyMatrix) {

int size = adjacencyMatrix.size();

vector<vector<int>> adjacencyList(size);

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

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

if (adjacencyMatrix[i][j] != 0) {

adjacencyList[i].push\_back(j);

}

}

}

return adjacencyList;

}

void printAdjacencyList(const vector<vector<int>>& adjacencyList) {

int size = adjacencyList.size();

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

cout << "Вершина " << i << ": ";

for (int j = 0; j < adjacencyList[i].size(); j++) {

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

}

cout << endl;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(0));

int size;

cout << "Введите размер матрицы смежности G1: ";

cin >> size;

// Создаем матрицу смежности и заполняем случайными числами

vector<vector<int>> adjacencyMatrix(size, vector<int>(size, 0));

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

for (int j = i + 1; j < size; j++) {

int randomNum = getRandomNumber(0, 1);

adjacencyMatrix[i][j] = randomNum;

adjacencyMatrix[j][i] = randomNum;

}

}

int size1;

cout << "Введите размер матрицы смежности G2: ";

cin >> size1;

vector<vector<int>> adjacencyMatrix1(size1, vector<int>(size1, 0));

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

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

if (i != j) {

int randomNum = getRandomNumber(0, 1);

adjacencyMatrix1[i][j] = randomNum;

adjacencyMatrix1[j][i] = randomNum; // Заполняем симметричные элементы

}

}

}

vector<vector<int>> adjacencyList = generateAdjacencyList(adjacencyMatrix);

vector<vector<int>> adjacencyList1 = generateAdjacencyList(adjacencyMatrix1);

int choice;

cout << endl;

cout << "=====================================================================" << endl;

cout << "=======================| Операции с графами |========================" << endl;

cout << "=====================================================================" << endl;

do {

cout << "1. Отождествление вершин" << endl;

cout << "2. Стягивание ребра" << endl;

cout << "3. Вывод матрицы и списка смежности G1" << endl;

cout << "4. Вывод матрицы и списка смежности G2" << endl;

cout << "5. Расщепление вершины" << endl;

cout << "6. Объединение графов" << endl;

cout << "7. Пересечение графов" << endl;

cout << "8. Кольцевая сумма" << endl;

cout << "0. Выход" << endl;

cout << "Выберите операцию: ";

cin >> choice;

switch (choice) {

case 1: {

int vertex1, vertex2;

cout << "Введите номера вершин для отождествления: ";

cin >> vertex1 >> vertex2;

if (vertex1 >= 0 && vertex1 < size && vertex2 >= 0 && vertex2 < size) {

identifyVertices(adjacencyMatrix, vertex1, vertex2);

size--;

cout << "Вершины успешно отождествлены!" << endl;

}

else {

cout << "Некорректные номера вершин!" << endl;

}

break;

}

case 2: {

int vertex1, vertex2;

cout << "Введите номера вершин для стягивания ребра: ";

cin >> vertex1 >> vertex2;

if (vertex1 >= 0 && vertex1 < size && vertex2 >= 0 && vertex2 < size) {

contractEdge(adjacencyMatrix, vertex1, vertex2);

size++;

cout << "Ребро стянуто и создана новая вершина" << endl;

}

else {

cout << "Некорректные номера вершин!" << endl;

}

break;

}

case 3: {

cout << "Матрица смежности графа G1:" << endl;

printMatrix(adjacencyMatrix);

cout << "Список смежности:" << endl;

printAdjacencyList(adjacencyList);

break;

}

case 4: {

cout << "Матрица смежности графа G2:" << endl;

printMatrix(adjacencyMatrix1);

cout << "Список смежности:" << endl;

printAdjacencyList(adjacencyList1);

break;

}

case 5: {

int vertex1, vertex2;

cout << "Введите вершины на участке расщепления: ";

cin >> vertex1 >> vertex2;

if (vertex1 >= 0 && vertex1 < size && vertex2 >= 0 && vertex2 < size) {

abiba(adjacencyMatrix, vertex1, vertex2);

size++;

cout << "Расщепление выполнено!" << endl;

}

else {

cout << "Некорректные номера вершин!" << endl;

}

break;

}

case 6: {

mergeGraphs(adjacencyMatrix, adjacencyMatrix1);

break;

}

case 7: {

intersectionGraphs(adjacencyMatrix, adjacencyMatrix1);

break;

}

case 8: {

XORGraphs(adjacencyMatrix, adjacencyMatrix1);

break;

}

case 0: {

cout << "Выход из программы." << endl;

break;

}

default: {

cout << "Некорректный выбор операции!" << endl;

break;

}

}

} while (choice != 0);

return 0;

}