

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №6

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Унарные и бинарные операции над графами"

Выполнили студенты группы
24ВВВЗ:

Пяткин Р. С.

Гусаров Е. Е.

Принял:

к.т.н., доцент, Юрова О. В.

к.т.н., Деев М. В.

Пенза 2025

Цель

Изучение унарных и бинарных операций над графами.

Лабораторное задание

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) две матрицы M_1 , M_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

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.

* 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

Для матричной формы представления графов выполните операцию:

- а) объединения
- б) пересечения
- в) кольцевой суммы

Результат выполнения операции выведите на экран.

Задание 4 *

Для матричной формы представления графов выполните операцию декартова произведения графов $G = G_1 \times G_2$.

Результат выполнения операции выведите на экран.

Результаты программ

```
Введите кол-во вершин в матрице M1: 3
Введите кол-во вершин в матрице M2: 4
Матрица смежности M1:
    0  1  2
0| 0  1  1
1| 1  0  1
2| 1  1  1

Матрица смежности M2:
    0  1  2  3
0| 0  0  0  1
1| 0  1  0  0
2| 0  0  1  0
3| 1  0  0  1

Список смежности для M1:
Вершина 0: -> 2 -> 1
Вершина 1: -> 2 -> 0
Вершина 2: -> 2 -> 1 -> 0

Список смежности для M2:
Вершина 0: -> 3
Вершина 1: -> 1
Вершина 2: -> 2
Вершина 3: -> 3 -> 0
```

Рис. 1 — lab_6_1.cpp

```

Введите количество вершин в графе: 3
Исходная матрица смежности:
    0  1  2
0| 1  0  1
1| 0  0  1
2| 1  1  1

Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущую матрицу
5 - создать новый граф
6 - выход
Ваш выбор: 1
Введите номера вершин для отождествления: 1
2
Результат операции:
    0  1
0| 1  1
1| 1  1

Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущую матрицу
5 - создать новый граф
6 - выход
Ваш выбор: 3
Введите номер вершины для расщепления: 0
Результат операции:
    0  1  2
0| 1  1  1
1| 1  1  1
2| 1  1  0

```

Рис. 2.1 — lab_6_2_1.cpp

```
Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущую матрицу
5 - создать новый граф
6 - выход
Ваш выбор: 2
Введите номера вершин для стягивания ребра: 1
2
Результат операции:
      0   1
0 | 1   1
1 | 1   1

Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущую матрицу
5 - создать новый граф
6 - выход
Ваш выбор: 6
Программа завершена.
```

Рис. 2.2 — lab_6_2_1.cpp

```
Введите количество вершин в графе: 4
Исходный граф:
Вершина 0: -> 3 -> 2 -> 1
Вершина 1: -> 3 -> 0
Вершина 2: -> 0
Вершина 3: -> 1 -> 0

Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущий граф
5 - создать новый граф
6 - выход
Ваш выбор: 1
Введите номера вершин для отождествления: 1
2
Результат операции:
Вершина 0: -> 2 -> 1
Вершина 1: -> 2 -> 0
Вершина 2: -> 1 -> 0

Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущий граф
5 - создать новый граф
6 - выход
Ваш выбор: 3
Введите номер вершины для расщепления: 2
Результат операции:
Вершина 0: -> 1 -> 3
Вершина 1: -> 2 -> 0
Вершина 2: -> 3 -> 1
Вершина 3: -> 2 -> 0
```

Рис. 3.1 — lab_6_2_2.cpp

```
Вершина 0: -> 1 -> 3
Вершина 1: -> 2 -> 0
Вершина 2: -> 3 -> 1
Вершина 3: -> 2 -> 0

Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущий граф
5 - создать новый граф
6 - выход
Ваш выбор: 2
Введите номера вершин для стягивания ребра: 2
1
Результат операции:
Вершина 0: -> 1 -> 2
Вершина 1: -> 2
Вершина 2: -> 1 -> 0

Выберите операцию:
1 - отождествление вершин
2 - стягивание ребра
3 - расщепление вершины
4 - показать текущий граф
5 - создать новый граф
6 - выход
Ваш выбор: 
```

Рис. 3.2 — lab_6_2_2.cpp

```

Внимание: Графы имеют разные размеры (3 и 4).
Хотите привести матрицы к одному размеру для выполнения операций? (y/n): y

Матрицы приведены к размеру 4x4:
Матрица смежности G1:
  0  1  2  3
0| 0  0  1  0
1| 0  0  0  0
2| 1  0  0  0
3| 0  0  0  0

Матрица смежности G2:
  0  1  2  3
0| 0  0  0  0
1| 0  0  1  0
2| 0  1  0  0
3| 0  0  0  0

Результаты операций над графами:
Объединение  $G1 \cup G2$ :
  0  1  2  3
0| 0  0  1  0
1| 0  0  1  0
2| 1  1  0  0
3| 0  0  0  0

Пересечение  $G1 \cap G2$ :
  0  1  2  3
0| 0  0  0  0
1| 0  0  0  0
2| 0  0  0  0
3| 0  0  0  0

Кольцевая сумма  $G1 \oplus G2$ :
  0  1  2  3
0| 0  0  1  0
1| 0  0  1  0
2| 1  1  0  0
3| 0  0  0  0

Программа завершена.

```

Рис. 4 — lab_3.cpp

```

ruslan@DexpAtlas:~/Документы/Algorithmization/LAB_6$ ./test
Введите количество вершин в графе G1: 2
Введите количество вершин в графе G2: 3
Матрица смежности G1 (2 вершин):
    0  1
0| 0  0
1| 0  1

Матрица смежности G2 (3 вершин):
    0  1  2
0| 0  0  1
1| 0  0  1
2| 1  1  1

Декартово произведение G1 X G2 (6 вершин):
    0  1  2  3  4  5
0| 0  0  1  0  0  0
1| 0  0  1  0  0  0
2| 1  1  1  0  0  0
3| 0  0  0  1  0  1
4| 0  0  0  0  1  1
5| 0  0  0  1  1  1

Соответствие вершин:
Вершина G1 X G2 = (вершина G1, вершина G2)
Вершина 0 = (0, 0)
Вершина 1 = (0, 1)
Вершина 2 = (0, 2)
Вершина 3 = (1, 0)
Вершина 4 = (1, 1)
Вершина 5 = (1, 2)

```

Рис. 5 — lab_4.cpp

Вывод:

В ходе выполнения лабораторной работы были разработаны программы для выполнения заданий Лабораторной работы №6. В процессе выполнения работы были использованы знания о унарных и бинарных операциях над графами.

Листинг

Файл lab_6_1.cpp

```
#include <fstream>

#include <iostream>

#include <ctime>

#include <random>

#include <list>


using namespace std;


// Список смежности

class Node{
public:
    int data;
    Node* next;

    Node(int value):data(value),next(nullptr){}
};

class LinkedList{
private:
    Node* head;

public:
    LinkedList():head(nullptr){}
    ~LinkedList(){ clear(); }

    void push_front(int value) {
        Node* newNode = new Node(value);
        newNode->next = head;
        head = newNode;
    }
};
```

```

}

Node* getHead() const{
return head;
}

void clear() {
Node* current = head;
while (current != nullptr) {
Node* next = current->next;
delete current;
current = next;
}
head = nullptr;
}

bool empty() const {
return head == nullptr;
}

};

class Graph{
int numVertices;
LinkedList* adjLists;
public:
Graph(int V);
void addEdge(int src, int dest);
void printGraph();
~Graph();
};

Graph::Graph(int vertices){
numVertices = vertices;
adjLists = new LinkedList[vertices];

```

```

}
Graph::~~Graph(){
delete[] adjLists;
}
void Graph::addEdge(int src, int dest){
adjLists[src].push_front(dest);
}
void Graph::printGraph()
{
for (int i = 0; i < numVertices; i++)
{
cout << "Вершина " << i << ":";
Node* current = adjLists[i].getHead();
while (current != nullptr)
{
cout << " -> " << current->data;
current = current->next;
}
cout << endl;
}
cout << endl;
}

```

```

// Матрица смежности
int printMatrix(int** m, int n){
cout << " ";
for(int i = 0; i < n; i++){
cout << i << " ";
}
cout << endl;

```

```

for(int i = 0; i < n; i++){
    cout << i << "| ";
    for(int j = 0; j < n; j++){
        cout << m[i][j] << " ";
    }
    cout << endl;
}
cout << endl;
return 0;
}

int** createMatrix(int n){
    int** m = new int*[n];
    for (int i = 0; i < n; i++) {
        m[i] = new int[n];
    }
    for(int i = 0; i < n; i++){
        for(int j = i; j < n; j++){
            m[j][i] = m[i][j] = rand() % 2;
        }
    }
    return m;
}

void deleteMatrix(int** m, int n) {
    for (int i = 0; i < n; ++i) {
        delete[] m[i];
    }
    delete[] m;
}

```

// Функция для преобразования матрицы смежности в список смежности

```

Graph matrixToList(int** matrix, int n) {

```

```

Graph graph(n);
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (matrix[i][j] == 1) {
            graph.addEdge(i, j);
        }
    }
}
return graph;
}

int main(){
    srand(time(0));
    int N1 = 0;
    int N2 = 0;
    cout << "Введите кол-во вершин в матрице M1: ";
    cin >> N1;
    cout << "Введите кол-во вершин в матрице M2: ";
    cin >> N2;
    int** M1 = createMatrix(N1);
    int** M2 = createMatrix(N2);
    cout << "Матрица смежности M1:" << endl;
    printMatrix(M1, N1);
    cout << "Матрица смежности M2:" << endl;
    printMatrix(M2, N2);

    Graph graph1 = matrixToList(M1, N1);
    Graph graph2 = matrixToList(M2, N2);
    cout << "Список смежности для M1:" << endl;
    graph1.printGraph();
    cout << "Список смежности для M2:" << endl;

```

```
graph2.printGraph();
```

```
deleteMatrix(M1, N1);
```

```
deleteMatrix(M2, N2);
```

```
return 0;
```

```
}
```

Файл lab_6_2_1.cpp

```
#include <fstream>
```

```
#include <iostream>
```

```
#include <ctime>
```

```
#include <random>
```

```
#include <list>
```

```
using namespace std;
```

```
void printMatrix(int** m, int n) {
```

```
cout << " ";
```

```
for (int i = 0; i < n; i++) {
```

```
cout << i << " ";
```

```
}
```

```
cout << endl;
```

```
for (int i = 0; i < n; i++) {
```

```
cout << i << "| ";
```

```
for (int j = 0; j < n; j++) {
```

```
cout << m[i][j] << " ";
```

```
}
```

```
cout << endl;
```

```
}
```

```
cout << endl;
```

```
}
```

```

int** createMatrix(int n) {
    int** m = new int*[n];
    for (int i = 0; i < n; i++) {
        m[i] = new int[n];
        for (int j = 0; j < n; j++) {
            m[i][j] = 0;
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            m[j][i] = m[i][j] = rand() % 2;
        }
    }
    return m;
}

```

```

int** copyMatrix(int** source, int n) {
    int** copy = new int*[n];
    for (int i = 0; i < n; i++) {
        copy[i] = new int[n];
        for (int j = 0; j < n; j++) {
            copy[i][j] = source[i][j];
        }
    }
    return copy;
}

```

```

void deleteMatrix(int** m, int n) {
    for (int i = 0; i < n; ++i) {
        delete[] m[i];
    }
}

```

```
delete[] m;
```

```
}
```

```
int** identifyVertices(int** matrix, int n, int v1, int v2) {
```

```
if (v1 == v2 || v1 >= n || v2 >= n || v1 < 0 || v2 < 0) {
```

```
cout << "Неверные вершины для отождествления!" << endl;
```

```
return nullptr;
```

```
}
```

```
int newSize = n - 1;
```

```
int** newMatrix = new int*[newSize];
```

```
for (int i = 0; i < newSize; i++) {
```

```
newMatrix[i] = new int[newSize]();
```

```
}
```

```
int oldToNew[n];
```

```
int newIndex = 0;
```

```
for (int i = 0; i < n; i++) {
```

```
if (i != v2) {
```

```
oldToNew[i] = newIndex++;
```

```
}
```

```
}
```

```
for (int i = 0; i < n; i++) {
```

```
if (i == v2) continue;
```

```
for (int j = 0; j < n; j++) {
```

```
if (j == v2) continue;
```

```
int newI = oldToNew[i];
```

```
int newJ = oldToNew[j];
```

```
if (i == v1 && j == v1) {
```

```
newMatrix[newI][newJ] = matrix[v1][v1] || matrix[v2][v2];
```

```
}
```

```
else if (i == v1) {
```

```
newMatrix[newI][newJ] = matrix[v1][j] || matrix[v2][j];
```

```

}
else if (j == v1) {
    newMatrix[newI][newJ] = matrix[i][v1] || matrix[i][v2];
}
else {
    newMatrix[newI][newJ] = matrix[i][j];
}
}
}
return newMatrix;
}

int** contractEdge(int** matrix, int n, int v1, int v2) {
    if (v1 == v2 || v1 >= n || v2 >= n || v1 < 0 || v2 < 0) {
        cout << "Неверные вершины для стягивания!" << endl;
        return nullptr;
    }
    if (matrix[v1][v2] == 0) {
        cout << "Ребро между вершинами " << v1 << " и " << v2 << " не существует!" << endl;
        return nullptr;
    }
    return identifyVertices(matrix, n, v1, v2);
}

int** splitVertex(int** matrix, int n, int v) {
    if (v >= n || v < 0) {
        cout << "Неверная вершина для расщепления!" << endl;
        return nullptr;
    }
    int newSize = n + 1;

```

```

int** newMatrix = new int*[newSize];
for (int i = 0; i < newSize; i++) {
    newMatrix[i] = new int[newSize]();
}
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        newMatrix[i][j] = matrix[i][j];
    }
}
int newVertex = n;
newMatrix[v][newVertex] = 1;
newMatrix[newVertex][v] = 1;
for (int i = 0; i < n; i++) {
    if (i != v && matrix[v][i] == 1) {
        newMatrix[newVertex][i] = 1;
        newMatrix[i][newVertex] = 1;
    }
}
return newMatrix;
}

int main() {
    srand(time(0));
    int n;
    cout << "Введите количество вершин в графе: ";
    cin >> n;
    int** matrix = createMatrix(n);
    int currentSize = n;
    cout << "Исходная матрица смежности:" << endl;
    printMatrix(matrix, currentSize);
    int choice;

```

```

bool running = true;
while (running) {
    cout << "\nВыберите операцию:" << endl;
    cout << "1 - отождествление вершин" << endl;
    cout << "2 - стягивание ребра" << endl;
    cout << "3 - расщепление вершины" << endl;
    cout << "4 - показать текущую матрицу" << endl;
    cout << "5 - создать новый граф" << endl;
    cout << "6 - выход" << endl;
    cout << "Ваш выбор: ";
    cin >> choice;
    int** resultMatrix = nullptr;
    int newSize = currentSize;
    switch (choice) {
        case 1: {
            int v1, v2;
            cout << "Введите номера вершин для отождествления: ";
            cin >> v1 >> v2;
            resultMatrix = identifyVertices(matrix, currentSize, v1, v2);
            newSize = currentSize - 1;
            break;
        }
        case 2: {
            int v1, v2;
            cout << "Введите номера вершин для стягивания ребра: ";
            cin >> v1 >> v2;
            resultMatrix = contractEdge(matrix, currentSize, v1, v2);
            newSize = currentSize - 1;
            break;
        }
        case 3: {

```

```
int v;
cout << "Введите номер вершины для расщепления: ";
cin >> v;
resultMatrix = splitVertex(matrix, currentSize, v);
newSize = currentSize + 1;
break;
}
case 4: {
cout << "Текущая матрица смежности:" << endl;
printMatrix(matrix, currentSize);
break;
}
case 5: {
deleteMatrix(matrix, currentSize);
cout << "Введите количество вершин в новом графе: ";
cin >> n;
matrix = createMatrix(n);
currentSize = n;
cout << "Новая матрица смежности:" << endl;
printMatrix(matrix, currentSize);
break;
}
case 6: {
running = false;
break;
}
default:
cout << "Неверный выбор!" << endl;
}
if (resultMatrix != nullptr && choice >= 1 && choice <= 3) {
deleteMatrix(matrix, currentSize);
```

```

matrix = resultMatrix;
currentSize = newSize;
cout << "Результат операции:" << endl;
printMatrix(matrix, currentSize);
}
}
deleteMatrix(matrix, currentSize);
cout << "Программа завершена." << endl;
return 0;
}

```

Файл lab_6_2_2.cpp

```

#include <iostream>
#include <ctime>
#include <random>

using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

```

```

~LinkedList() { clear(); }

void push_front(int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

Node* getHead() const {
    return head;
}

void clear() {
    Node* current = head;
    while (current != nullptr) {
        Node* next = current->next;
        delete current;
        current = next;
    }
    head = nullptr;
}

bool empty() const {
    return head == nullptr;
}

bool contains(int value) const {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == value) {
            return true;
        }
        current = current->next;
    }
    return false;
}

```

```

}

void remove(int value) {
    Node* current = head;
    Node* prev = nullptr;
    while (current != nullptr && current->data != value) {
        prev = current;
        current = current->next;
    }
    if (current == nullptr) return;
    if (prev == nullptr) {
        head = current->next;
    } else {
        prev->next = current->next;
    }
    delete current;
}

LinkedList& operator=(const LinkedList& other) {
    if (this != &other) {
        clear();
        Node* current = other.head;
        while (current != nullptr) {
            push_front(current->data);
            current = current->next;
        }
    }
    return *this;
}

LinkedList(const LinkedList& other) : head(nullptr) {
    Node* current = other.head;
    while (current != nullptr) {
        push_front(current->data);
    }
}

```

```

current = current->next;
}
}
};

```

```

class Graph {
int numVertices;
LinkedList* adjLists;
public:
Graph(int V);
Graph(const Graph& other);
Graph& operator=(const Graph& other);
void addEdge(int src, int dest);
void removeEdge(int src, int dest);
bool hasEdge(int src, int dest) const;
void printGraph() const;
int getNumVertices() const { return numVertices; }
const LinkedList& getAdjList(int vertex) const { return adjLists[vertex]; }
~Graph();
};

Graph::Graph(int vertices) {
numVertices = vertices;
adjLists = new LinkedList[vertices];
}

Graph::Graph(const Graph& other) {
numVertices = other.numVertices;
adjLists = new LinkedList[numVertices];
for (int i = 0; i < numVertices; i++) {
Node* current = other.adjLists[i].getHead();
while (current != nullptr) {

```

```

adjLists[i].push_front(current->data);
current = current->next;
}
}
}

Graph& Graph::operator=(const Graph& other) {
    if (this != &other) {
        delete[] adjLists;
        numVertices = other.numVertices;
        adjLists = new LinkedList[numVertices];
        for (int i = 0; i < numVertices; i++) {
            Node* current = other.adjLists[i].getHead();
            while (current != nullptr) {
                adjLists[i].push_front(current->data);
                current = current->next;
            }
        }
    }
    return *this;
}

Graph::~~Graph() {
    delete[] adjLists;
}

void Graph::addEdge(int src, int dest) {
    if (src >= 0 && src < numVertices && dest >= 0 && dest < numVertices) {
        if (!adjLists[src].contains(dest)) {
            adjLists[src].push_front(dest);
        }
    }
}

```

```

void Graph::removeEdge(int src, int dest) {
    if (src >= 0 && src < numVertices && dest >= 0 && dest < numVertices) {
        adjLists[src].remove(dest);
    }
}

bool Graph::hasEdge(int src, int dest) const {
    if (src >= 0 && src < numVertices && dest >= 0 && dest < numVertices) {
        return adjLists[src].contains(dest);
    }
    return false;
}

void Graph::printGraph() const {
    for (int i = 0; i < numVertices; i++) {
        cout << "Вершина " << i << ":";
        Node* current = adjLists[i].getHead();
        while (current != nullptr) {
            cout << " -> " << current->data;
            current = current->next;
        }
        cout << endl;
    }
    cout << endl;
}

Graph createGraph(int n) {
    Graph graph(n);
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (rand() % 2 == 1) {
                graph.addEdge(i, j);
                graph.addEdge(j, i);
            }
        }
    }
}

```

```

}
}
}
return graph;
}

// Отождествление вершин
Graph identifyVertices(const Graph& graph, int v1, int v2) {
    if (v1 == v2 || v1 >= graph.getNumVertices() || v2 >=
graph.getNumVertices() || v1 < 0 || v2 < 0) {
        cout << "Неверные вершины для отождествления!" << endl;
        return graph;
    }
    int newSize = graph.getNumVertices() - 1;
    Graph newGraph(newSize);
    int* mapping = new int[graph.getNumVertices()];
    int newIndex = 0;
    for (int i = 0; i < graph.getNumVertices(); i++) {
        if (i == v2) {
            mapping[i] = (v1 < v2) ? v1 : v1 - 1;
        } else {
            mapping[i] = newIndex++;
        }
    }
    for (int i = 0; i < graph.getNumVertices(); i++) {
        if (i == v2) continue;
        Node* current = graph.getAdjList(i).getHead();
        while (current != nullptr) {
            int j = current->data;
            int newI = mapping[i];
            int newJ = mapping[j];

```

```

if (newI != newJ) {
    newGraph.addEdge(newI, newJ);
}
current = current->next;
}
}
delete[] mapping;
return newGraph;
}

```

// Стягивание ребра

```

Graph contractEdge(const Graph& graph, int v1, int v2) {
    if (v1 == v2 || v1 >= graph.getNumVertices() || v2 >=
        graph.getNumVertices() || v1 < 0 || v2 < 0) {
        cout << "Неверные вершины для стягивания!" << endl;
        return graph;
    }
    if (!graph.hasEdge(v1, v2)) {
        cout << "Ребро между вершинами " << v1 << " и " << v2 << " не
            существует!" << endl;
        return graph;
    }
    return identifyVertices(graph, v1, v2);
}

```

```

Graph splitVertex(const Graph& graph, int v) {
    if (v >= graph.getNumVertices() || v < 0) {
        cout << "Неверная вершина для расщепления!" << endl;
        return graph;
    }
    int newSize = graph.getNumVertices() + 1;
    Graph newGraph(newSize);

```

```

int newVertex = graph.getNumVertices();
newGraph.addEdge(v, newVertex);
newGraph.addEdge(newVertex, v);
for (int i = 0; i < graph.getNumVertices(); i++) {
Node* current = graph.getAdjList(i).getHead();
while (current != nullptr) {
int j = current->data;
newGraph.addEdge(i, j);
current = current->next;
}
}
Node* current = graph.getAdjList(v).getHead();
while (current != nullptr) {
int neighbor = current->data;
if (neighbor != v && rand() % 2 == 0) {
newGraph.removeEdge(v, neighbor);
newGraph.removeEdge(neighbor, v);
newGraph.addEdge(newVertex, neighbor);
newGraph.addEdge(neighbor, newVertex);
}
current = current->next;
}
return newGraph;
}

```

```

int main() {
srand(time(0));

int n;

cout << "Введите количество вершин в графе: ";
cin >> n;

Graph graph = createGraph(n);

```

```

int currentSize = n;
cout << "Исходный граф:" << endl;
graph.printGraph();
int choice;
bool running = true;
while (running) {
    cout << "\nВыберите операцию:" << endl;
    cout << "1 - отождествление вершин" << endl;
    cout << "2 - стягивание ребра" << endl;
    cout << "3 - расщепление вершины" << endl;
    cout << "4 - показать текущий граф" << endl;
    cout << "5 - создать новый граф" << endl;
    cout << "6 - выход" << endl;
    cout << "Ваш выбор: ";
    cin >> choice;
    Graph resultGraph = graph;
    int newSize = currentSize;
    switch (choice) {
    case 1: {
        int v1, v2;
        cout << "Введите номера вершин для отождествления: ";
        cin >> v1 >> v2;
        resultGraph = identifyVertices(graph, v1, v2);
        newSize = currentSize - 1;
        break;
    }
    case 2: {
        int v1, v2;
        cout << "Введите номера вершин для стягивания ребра: ";
        cin >> v1 >> v2;
        resultGraph = contractEdge(graph, v1, v2);
    }
    }
}

```

```
newSize = currentSize - 1;
break;
}
case 3: {
int v;
cout << "Введите номер вершины для расщепления: ";
cin >> v;
resultGraph = splitVertex(graph, v);
newSize = currentSize + 1;
break;
}
case 4: {
cout << "Текущий граф:" << endl;
graph.printGraph();
break;
}
case 5: {
cout << "Введите количество вершин в новом графе: ";
cin >> n;
graph = createGraph(n);
currentSize = n;
cout << "Новый граф:" << endl;
graph.printGraph();
break;
}
case 6: {
running = false;
break;
}
default:
cout << "Неверный выбор!" << endl;
```

```

}

if (choice >= 1 && choice <= 3) {
    graph = resultGraph;
    currentSize = newSize;
    cout << "Результат операции:" << endl;
    graph.printGraph();
}
}

cout << "Программа завершена." << endl;
return 0;
}

```

Файл lab_6_3.cpp

```

#include <iostream>
#include <ctime>
#include <random>
#include <algorithm>

using namespace std;

int printMatrix(int** m, int n) {
    cout << " ";
    for (int i = 0; i < n; i++) {
        cout << i << " ";
    }
    cout << endl;
    for (int i = 0; i < n; i++) {
        cout << i << "| ";
        for (int j = 0; j < n; j++) {
            cout << m[i][j] << " ";
        }
        cout << endl;
    }
}

```

```
}  
cout << endl;  
return 0;  
}
```

```
int** createMatrix(int n) {  
    int** m = new int*[n];  
    for (int i = 0; i < n; i++) {  
        m[i] = new int[n];  
        for (int j = 0; j < n; j++) {  
            m[i][j] = 0;  
        }  
    }  
    for (int i = 0; i < n; i++) {  
        for (int j = i; j < n; j++) {  
            if (i == j) {  
                m[i][j] = 0;  
            } else {  
                m[i][j] = m[j][i] = rand() % 2;  
            }  
        }  
    }  
    return m;  
}
```

```
int** resizeMatrix(int** original, int oldSize, int newSize) {  
    int** newMatrix = new int*[newSize];  
    for (int i = 0; i < newSize; i++) {  
        newMatrix[i] = new int[newSize];  
        for (int j = 0; j < newSize; j++) {  
            if (i < oldSize && j < oldSize) {
```

```

newMatrix[i][j] = original[i][j];
} else {
newMatrix[i][j] = 0;
}
}
}
return newMatrix;
}

```

```

void deleteMatrix(int** m, int n) {
for (int i = 0; i < n; ++i) {
delete[] m[i];
}
delete[] m;
}

```

```

int** unionGraphs(int** G1, int** G2, int n) {
int** result = new int*[n];
for (int i = 0; i < n; i++) {
result[i] = new int[n];
for (int j = 0; j < n; j++) {
result[i][j] = (G1[i][j] == 1 || G2[i][j] == 1) ? 1 : 0;
}
}
return result;
}

```

```

int** intersectionGraphs(int** G1, int** G2, int n) {
int** result = new int*[n];
for (int i = 0; i < n; i++) {
result[i] = new int[n];

```

```

for (int j = 0; j < n; j++) {
    result[i][j] = (G1[i][j] == 1 && G2[i][j] == 1) ? 1 : 0;
}
}
return result;
}

```

```

int** ringSumGraphs(int** G1, int** G2, int n) {
    int** result = new int*[n];
    for (int i = 0; i < n; i++) {
        result[i] = new int[n];
        for (int j = 0; j < n; j++) {
            result[i][j] = (G1[i][j] != G2[i][j]) ? 1 : 0;
        }
    }
    return result;
}

```

```

int main() {
    srand(time(0));

    int n1, n2;

    cout << "Введите количество вершин в графе G1: ";
    cin >> n1;

    cout << "Введите количество вершин в графе G2: ";
    cin >> n2;

    int** G1 = createMatrix(n1);
    int** G2 = createMatrix(n2);

    cout << "\nМатрица смежности G1 (" << n1 << "x" << n1 << "):" << endl;
    printMatrix(G1, n1);

    cout << "Матрица смежности G2 (" << n2 << "x" << n2 << "):" << endl;
    printMatrix(G2, n2);
}

```

```

if (n1 != n2) {
cout << "Внимание: Графы имеют разные размеры (" << n1 << " и " << n2 <<
")." << endl;

cout << "Хотите привести матрицы к одному размеру для выполнения
операций? (y/n): ";

char choice;

cin >> choice;

if (choice == 'y' || choice == 'Y') {
int newSize = max(n1, n2);

if (n1 < newSize) {
int** temp = resizeMatrix(G1, n1, newSize);
deleteMatrix(G1, n1);
G1 = temp;
n1 = newSize;
}

if (n2 < newSize) {
int** temp = resizeMatrix(G2, n2, newSize);
deleteMatrix(G2, n2);
G2 = temp;
n2 = newSize;
}

cout << "\nМатрицы приведены к размеру " << newSize << "x" << newSize <<
":" << endl;

cout << "Матрица смежности G1:" << endl;
printMatrix(G1, newSize);

cout << "Матрица смежности G2:" << endl;
printMatrix(G2, newSize);
} else {
cout << "\nОперации над графами не выполнялись." << endl;

cout << "Операции (объединение, пересечение, кольцевая сумма)" << endl;

cout << "могут быть выполнены только для графов одинакового размера." <<
endl;
}
}

```

```

deleteMatrix(G1, n1);
deleteMatrix(G2, n2);
return 0;
}
}

int n = n1;
cout << "\nРезультаты операций над графами:" << endl;
int** unionResult = unionGraphs(G1, G2, n);
cout << "Объединение  $G_1 \cup G_2$ :" << endl;
printMatrix(unionResult, n);
int** intersectionResult = intersectionGraphs(G1, G2, n);
cout << "Пересечение  $G_1 \cap G_2$ :" << endl;
printMatrix(intersectionResult, n);
int** ringSumResult = ringSumGraphs(G1, G2, n);
cout << "Кольцевая сумма  $G_1 \oplus G_2$ :" << endl;
printMatrix(ringSumResult, n);
deleteMatrix(G1, n);
deleteMatrix(G2, n);
deleteMatrix(unionResult, n);
deleteMatrix(intersectionResult, n);
deleteMatrix(ringSumResult, n);
cout << "Программа завершена." << endl;
return 0;
}

```

Файл lab_6_4.cpp

```

#include <iostream>
#include <ctime>
#include <random>

using namespace std;

```

```

int printMatrix(int** m, int n){
    cout << " ";
    for(int i = 0; i < n; i++){
        cout << i << " ";
    }
    cout << endl;
    for(int i = 0; i < n; i++){
        cout << i << "| ";
        for(int j = 0; j < n; j++){
            cout << m[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
    return 0;
}

int** createMatrix(int n){
    int** m = new int*[n];
    for (int i = 0; i < n; i++) {
        m[i] = new int[n];
    }
    for(int i = 0; i < n; i++){
        for(int j = i; j < n; j++){
            m[j][i] = m[i][j] = rand() % 2;
        }
    }
    return m;
}

void deleteMatrix(int** m, int n) {
    for (int i = 0; i < n; ++i) {
        delete[] m[i];
    }
}

```

```
}
```

```
delete[] m;
```

```
}
```

```
int** cartesianProduct(int** G1, int n1, int** G2, int n2) {
```

```
int n = n1 * n2;
```

```
int** result = new int*[n];
```

```
for (int i = 0; i < n; i++) {
```

```
result[i] = new int[n];
```

```
for (int j = 0; j < n; j++) {
```

```
result[i][j] = 0;
```

```
}
```

```
}
```

```
for (int i = 0; i < n1; i++) {
```

```
for (int j = 0; j < n2; j++) {
```

```
for (int k = 0; k < n1; k++) {
```

```
for (int l = 0; l < n2; l++) {
```

```
int vertex1 = i * n2 + j;
```

```
int vertex2 = k * n2 + l;
```

```
if ((i == k && G2[j][l] == 1) || (j == l && G1[i][k] == 1)) {
```

```
result[vertex1][vertex2] = 1;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
return result;
```

```
}
```

```
int main(){
```

```
srand(time(0));
```

```

int n1, n2;

cout << "Введите количество вершин в графе G1: ";
cin >> n1;

cout << "Введите количество вершин в графе G2: ";
cin >> n2;

int** G1 = createMatrix(n1);
int** G2 = createMatrix(n2);

cout << "Матрица смежности G1 (" << n1 << " вершин):" << endl;
printMatrix(G1, n1);

cout << "Матрица смежности G2 (" << n2 << " вершин):" << endl;
printMatrix(G2, n2);

int** product = cartesianProduct(G1, n1, G2, n2);
int productSize = n1 * n2;

cout << "Декартово произведение G1 X G2 (" << productSize << " вершин):"
<< endl;

printMatrix(product, productSize);

cout << "Соответствие вершин:" << endl;

cout << "Вершина G1 X G2 = (вершина G1, вершина G2)" << endl;

for (int i = 0; i < n1; i++) {
    for (int j = 0; j < n2; j++) {
        int vertex = i * n2 + j;

        cout << "Вершина " << vertex << " = (" << i << ", " << j << ")" << endl;
    }
}

cout << endl;

deleteMatrix(G1, n1);
deleteMatrix(G2, n2);
deleteMatrix(product, productSize);

return 0;
}

```