Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнили ст. группы 22ВВВ1:

Уткин М.М.

Саветкин Д.Д.

Соколовский Е.В

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Научиться пользоваться унарными и бинарными операциями над графами.

**Лабораторное 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.

2.\* 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*1https://lh7-us.googleusercontent.com/TYTzmrvURXtPql_7of7cMc3J0zMTq6Lf4Ud5m1DdR71gHWEORiul5kXQ2rRfmjO07qny1qHtzHdFiBbHtRkdPbyFfPhyxgEBqTWkHKplTOfoh-gCAQCJbpR4xb6jzfJw_hMmIBvKPrb9NemGHALKuw*G*2

б) пересечения  *G* = *G*1https://lh7-us.googleusercontent.com/d6LyOnvg2LzU7eqtjEZKlu54X0EhJUslvflmUWXhnl1YNuL8VPpMjbAJCvsUADaAxhf5OpjnMfO0HdMWmC5hbWq8z-3DO966JuIrvSdG4PbRX31uJgQmOo_DYZNOjU9Hw8ploisnGPrgNb0yjzzw3g*G*2

в) кольцевой суммы *G* = *G*1https://lh7-us.googleusercontent.com/r9tumN74Sz4jmDZp2UGJB9N0bJdYRPi7DVWaWf8fBbDC-tge2w2NMYDqk7KqCgWEnUE_7HIMxcujK0cd5ZHP_R_1f1Hyl0x3Y-4FddQ0mo59zi6sq9CODYW0lLuBA4jao4YDtpXtmiSZmWmdL08oSg*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.

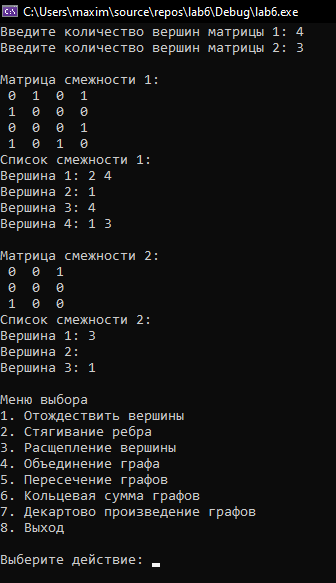
Результат выполнения операции выведите на экран.

.

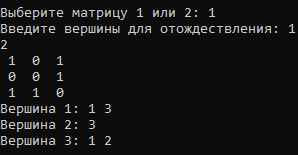
**Ход работы:**

В программе реализовали функции для различных операций над графами, таких как отождествление вершин, стягивание ребер, расщепление вершин, объединение графов, пересечение графов, кольцевая сумма графов, и декартово произведение графов. Реализовали создание и заполнение двух матриц и их списков сменностей для дальнейших операций над ними. Сделали интерфейс для выбора операции и просмотра результата.

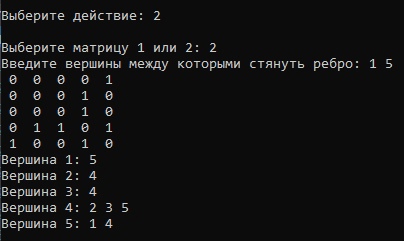
**Результаты работы программы**



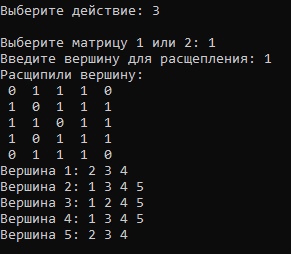
Отождествление вершин:



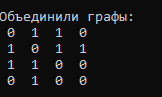
Стягивание ребра:



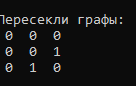
Расщепление вершины:



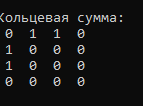
Объединение графов:



Пересечение графов:



Кольцевая сумма графов:



**Вывод:**

В ходе данной лабораторной работе, мы научились выполнять унарные и бинарные операции над графами.

**Листинг:**

#include <iostream>

#include <cstdlib>

#include <stack>

#include <vector>

#include <algorithm>

#include <queue>

#include <cstring>

#include <iomanip> // Для использования std::setw

using namespace std;

//Работа со списком смежности

struct Node {

int first;

Node\* next;

};

void List(Node\*\*& Matr1, int\*\* mtrx, int n) {

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

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

if (mtrx[i][j] == 1) {

Node\* newNode = new Node;

newNode->first = j + 1;

newNode->next = Matr1[i];

Matr1[i] = newNode;

}

}

}

}

void prList(Node\*\* Matr1, int n) {

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

Node\* actual = Matr1[i];

std::vector<int> vertices;

while (actual != nullptr) {

vertices.push\_back(actual->first);

actual = actual->next;

}

// Вывод вершин в обратном порядке

std::cout << "Вершина " << i + 1 << ": ";

for (auto it = vertices.rbegin(); it != vertices.rend(); ++it) {

std::cout << \*it << ' ';

}

std::cout << '\n';

}

}

void frList(Node\* head) {

Node\* actual = head;

while (actual != nullptr) {

Node\* temp = actual;

actual = actual->next;

delete temp;

}

}

// Очистка списка смежности

void frrList(Node\*& head) {

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

//Функции для работы с графами

//Функция для отождествления вершин

void likeningRows(int\*\* mtrx, int& n, int bar1, int bar2, Node\*\*& Matr1) {

// Проверка на корректность номеров вершин

if (bar1 >= n || bar2 >= n || bar1 < 0 || bar2 < 0) {

std::cout << "Такого выбора нет (\n";

return;

}

// Проверка наличия петли у вершины bar1

bool hasLoop1 = mtrx[bar1][bar1] != 0;

// Объединение рёбер в массиве смежности

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

mtrx[i][bar1] = mtrx[i][bar1] || mtrx[i][bar2];

}

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

mtrx[bar1][i] = mtrx[bar1][i] || mtrx[bar2][i];

}

// Сдвиг элементов после удаляемой строки и столбца

for (int i = bar2; i < n - 1; i++) {

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

mtrx[i][j] = mtrx[i + 1][j];

}

}

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

for (int j = bar2; j < n - 1; j++) {

mtrx[i][j] = mtrx[i][j + 1];

}

}

// Уменьшение количества вершин

n--;

// Проверка наличия петли у других вершин

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

if (mtrx[i][i] != 0) {

hasLoop1 = true;

break;

}

}

// Вывод петли, если она есть у вершины bar1

if (hasLoop1) {

mtrx[bar1][bar1] = 1; // Вывод петли в матрице

}

else {

mtrx[bar1][bar1] = 0; // Убираем петлю, если её не осталось

}

// Очистка списков смежности и создание новых

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

frList(Matr1[i]);

Matr1[i] = nullptr;

}

List(Matr1, mtrx, n);

}

//Функция для стягивания ребра

void ScreedRidge(int\*\* mtrx, int& n, int bar1, int bar2, Node\*\*& Matr1) {

// Проверка на корректность номеров вершин

if (bar1 >= n || bar2 >= n || bar1 < 0 || bar2 < 0) {

std::cout << "Такого выбора нет (\n";

return;

}

if (mtrx[bar1][bar2] == 0) {

std::cout << "Такого выбора нет (\n";

return;

}

// Сохранение рёбер перед удалением связанных вершин

std::vector<std::vector<int>> savedEdges(n, std::vector<int>(n, 0));

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

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

savedEdges[i][j] = mtrx[i][j];

}

}

// Объединение рёбер и удаление связанных вершин

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

mtrx[i][bar1] = mtrx[i][bar1] || mtrx[i][bar2];

}

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

mtrx[bar1][i] = mtrx[bar1][i] || mtrx[bar2][i];

}

mtrx[bar1][bar1] = 0;

// Сдвиг связей после удаленной вершины

for (int i = bar2; i < n - 1; i++) {

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

mtrx[i][j] = mtrx[i + 1][j];

}

}

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

for (int j = bar2; j < n - 1; j++) {

mtrx[i][j] = mtrx[i][j + 1];

}

}

// Уменьшение размера матрицы

n--;

// Восстановление рёбер после удаления связанных вершин

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

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

mtrx[i][j] = savedEdges[i][j];

}

}

// Очистка списков смежности

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

frList(Matr1[i]);

Matr1[i] = nullptr;

}

// Обновление списков смежности

List(Matr1, mtrx, n);

}

//Функция для расщепления вершин

void SplittingApex(int\*\*& mtrx, int& n, int bar1, Node\*\*& Matr1) {

// Создание нового массива для разделения вершины

int recent\_bar = n + 1;

int\*\* bar\_nw = new int\* [recent\_bar];

for (int i = 0; i < recent\_bar; i++) {

bar\_nw[i] = new int[recent\_bar];

for (int j = 0; j < recent\_bar; j++) {

if (i < n && j < n) {

bar\_nw[i][j] = mtrx[i][j]; // Копирование существующих данных

}

else if (i == n && j < n) {

bar\_nw[i][j] = mtrx[bar1][j]; // Копирование связей из удаляемой строки

}

else if (i < n && j == n) {

bar\_nw[i][j] = mtrx[i][bar1]; // Копирование связей из удаляемого столбца

}

else {

bar\_nw[i][j] = 1; // Заполнение новых элементов нулями

}

}

}

// Увеличение размера массива

n++;

int\*\* NwBr = new int\* [n];

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

NwBr[i] = new int[n];

}

// Копирование данных в новый массив

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

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

NwBr[i][j] = bar\_nw[i][j];

}

}

// Освобождение старого массива

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

delete[] mtrx[i];

}

delete[] mtrx;

// Переключение указателя на новый массив

mtrx = NwBr;

// Вывод результата

std::cout << "Расщипили вершину:\n";

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

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

std::cout << std::setw(2) << mtrx[i][j] << " ";

}

std::cout << "\n";

}

// Очистка списков смежности

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

frList(Matr1[i]);

Matr1[i] = nullptr;

}

// Выделение нового массива списков смежности

delete[] Matr1;

Matr1 = new Node \* [n];

if (Matr1 == nullptr) {

std::cout << "Памяти не хватает (\n";

return;

}

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

Matr1[i] = nullptr;

}

// Заполнение новых списков смежности

List(Matr1, mtrx, n);

prList(Matr1, n);

}

//Функция для объединения графов

void UnificationGrafs(int\*\* mtrx, int\*\* mtrx2, int n, int m) {

int max\_rows = (n > m) ? n : m;

// Выделение памяти для объединенного массива

int\*\* BarUn = new int\* [max\_rows];

for (int i = 0; i < max\_rows; i++) {

BarUn[i] = new int[max\_rows];

for (int j = 0; j < max\_rows; j++) {

int val\_1 = (i < n&& j < n) ? mtrx[i][j] : 0;

int val\_2 = (i < m&& j < m) ? mtrx2[i][j] : 0;

BarUn[i][j] = val\_1 || val\_2;

}

}

// Вывод результата

std::cout << "Объединили графы:\n";

for (int i = 0; i < max\_rows; i++) {

for (int j = 0; j < max\_rows; j++) {

std::cout << std::setw(2) << BarUn[i][j] << " ";

}

std::cout << "\n";

}

// Освобождение памяти

for (int i = 0; i < max\_rows; i++) {

delete[] BarUn[i];

}

delete[] BarUn;

}

//Функция для пересечения графов

void IntersectionGrafs(int\*\* mtrx, int\*\* mtrx2, int n, int m) {

int max\_rows = (n > m) ? n : m;

// Выделение памяти для массива пересечения

int\*\* BarIntrs = new int\* [max\_rows];

for (int i = 0; i < max\_rows; i++) {

BarIntrs[i] = new int[max\_rows];

for (int j = 0; j < max\_rows; j++) {

int val\_1 = (i < n&& j < n) ? mtrx[i][j] : 0;

int val\_2 = (i < m&& j < m) ? mtrx2[i][j] : 0;

BarIntrs[i][j] = val\_1 && val\_2;

}

}

// Вывод результата

std::cout << "Пересекли графы:\n";

for (int i = 0; i < max\_rows - 1; i++) {

for (int j = 0; j < max\_rows - 1; j++) {

std::cout << std::setw(2) << BarIntrs[i][j] << " ";

}

std::cout << "\n";

}

// Освобождение памяти

for (int i = 0; i < max\_rows; i++) {

delete[] BarIntrs[i];

}

delete[] BarIntrs;

}

//Функция для кольцевой суммы графов

void RingSumGrafs(int\*\* mtrx, int\*\* mtrx2, int n, int m) {

int max\_rows = (n > m) ? n : m;

// Выделение памяти для массива кольцевой суммы

int\*\* BarRing = new int\* [max\_rows];

for (int i = 0; i < max\_rows; i++) {

BarRing[i] = new int[max\_rows];

for (int j = 0; j < max\_rows; j++) {

if (i < n && i < m && j < n && j < m) {

if (i == j || (mtrx[i][j] == 1 && mtrx2[i][j] == 1)) {

BarRing[i][j] = 0;

}

else {

int val\_1 = (i < n&& j < n) ? mtrx[i][j] : 0;

int val\_2 = (i < m&& j < m) ? mtrx2[i][j] : 0;

BarRing[i][j] = val\_1 || val\_2;

}

}

else {

BarRing[i][j] = 0;

}

}

}

// Вывод результата

std::cout << "Кольцевая сумма:\n";

for (int i = 0; i < max\_rows; i++) {

for (int j = 0; j < max\_rows; j++) {

std::cout << std::setw(2) << BarRing[i][j] << " ";

}

std::cout << "\n";

}

// Освобождение памяти

for (int i = 0; i < max\_rows; i++) {

delete[] BarRing[i];

}

delete[] BarRing;

}

//Функция для декартого произведения графов

void DecGrafs(int\*\* mtrx, int\*\* mtrx2, int n, int m) {

int decart\_rows = n \* m;

int decart\_cols = n \* m;

// Выделение памяти для массива декартова произведения

int\*\* decart\_bar = new int\* [decart\_rows];

for (int i = 0; i < decart\_rows; i++) {

decart\_bar[i] = new int[decart\_cols];

for (int j = 0; j < decart\_cols; j++) {

decart\_bar[i][j] = 0;

}

}

int u = 0, v = 0;

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

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

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

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

u = i \* m + j;

v = k \* m + l;

if (j == l) {

decart\_bar[u][v] = mtrx[i][k];

}

if (i == k) {

decart\_bar[u][v] = mtrx2[j][l];

}

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

decart\_bar[u][v] = 0;

}

}

}

}

}

// Вывод результата

std::cout << "Декартовое произведение:\n";

for (int i = 0; i < decart\_rows; i++) {

for (int j = 0; j < decart\_cols; j++) {

std::cout << std::setw(2) << decart\_bar[i][j] << " ";

}

std::cout << "\n";

}

// Освобождение памяти

for (int i = 0; i < decart\_rows; i++) {

delete[] decart\_bar[i];

}

delete[] decart\_bar;

}

int main() {

setlocale(LC\_ALL, "RUS");

srand(time(nullptr));

int n, m;

std::cout << "Введите количество вершин матрицы 1: ";

std::cin >> n;

std::cout << "Введите количество вершин матрицы 2: ";

std::cin >> m;

Node\*\* Matr1 = new Node \* [n];

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

Matr1[i] = nullptr;

}

Node\*\* Matr2 = new Node \* [m];

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

Matr2[i] = nullptr;

}

int\*\* mtrx = new int\* [n];

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

mtrx[i] = new int[n];

}

// Инициализация массива mtrx случайными значениями для графа №1

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

for (int j = i; j < n; j++) {

if (i == j) {

mtrx[i][j] = 0; // На главной диагонали нули

}

else {

mtrx[i][j] = rand() % 2; // Случайные 0 или 1

mtrx[j][i] = mtrx[i][j]; // Симметричное заполнение

}

}

}

std::cout << "\n";

// Вывод матрицы смежности для графа №1

std::cout << "Матрица смежности 1:\n";

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

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

std::cout << std::setw(2) << mtrx[i][j] << " ";

}

std::cout << "\n";

}

// Вывод списков смежности для графа №1

std::cout << "Список смежности 1:\n";

List(Matr1, mtrx, n);

prList(Matr1, n);

// Выделение памяти для графа №2

int\*\* mtrx2 = new int\* [m];

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

mtrx2[i] = new int[m];

}

// Инициализация массива mtrx2 случайными значениями для графа №2

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

for (int j = i; j < m; j++) {

if (i == j) {

mtrx2[i][j] = 0; // На главной диагонали нули

}

else {

mtrx2[i][j] = rand() % 2; // Случайные 0 или 1

mtrx2[j][i] = mtrx2[i][j]; // Симметричное заполнение

}

}

}

std::cout << "\n";

// Вывод матрицы смежности для графа №2

std::cout << "Матрица смежности 2:\n";

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

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

std::cout << std::setw(2) << mtrx2[i][j] << " ";

}

std::cout << "\n";

}

// Вывод списков смежности для графа №2

std::cout << "Список смежности 2:\n";

List(Matr2, mtrx2, m);

prList(Matr2, m);

int option, bar1, bar2, provided;

do {

std::cout << "\n";

std::cout << "Меню выбора\n";

std::cout << "1. Отождествить вершины\n";

std::cout << "2. Стягивание ребра\n";

std::cout << "3. Расщепление вершины\n";

std::cout << "4. Объединение графа\n";

std::cout << "5. Пересечение графов\n";

std::cout << "6. Кольцевая сумма графов\n";

std::cout << "7. Декартово произведение графов\n";

std::cout << "8. Выход\n";

std::cout << "\n";

std::cout << "Выберите действие: ";

std::cin >> option;

switch (option) {

case 1:

std::cout << "\n";

std::cout << "Выберите матрицу 1 или 2: ";

std::cin >> provided;

if (provided == 1) {

std::cout << "Введите вершины для отождествления: ";

std::cin >> bar1 >> bar2;

likeningRows(mtrx, n, bar1 - 1, bar2 - 1, Matr1);

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

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

cout << std::setw(2) << mtrx[i][j] << ' ';

}

cout << endl;

}

prList(Matr1, n);

std::cout << "\n";

}

else if (provided == 2) {

std::cout << "Введите вершины для отождествления: ";

std::cin >> bar1 >> bar2;

likeningRows(mtrx2, m, bar1 - 1, bar2 - 1, Matr2);

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

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

cout << std::setw(2) << mtrx2[i][j] << ' ';

}

cout << endl;

}

prList(Matr2, m);

std::cout << "\n";

}

else {

std::cout << "Такого нет(\n";

std::cout << "\n";

}

break;

case 2:

std::cout << "\n";

std::cout << "Выберите матрицу 1 или 2: ";

std::cin >> provided;

if (provided == 1) {

std::cout << "Введите вершины между которыми стянуть ребро: ";

std::cin >> bar1 >> bar2;

ScreedRidge(mtrx, n, bar1 - 1, bar2 - 1, Matr1);

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

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

cout << std::setw(2) << mtrx[i][j] << ' ';

}

cout << endl;

}

prList(Matr1, n);

std::cout << "\n";

}

else if (provided == 2) {

std::cout << "Введите вершины между которыми стянуть ребро: ";

std::cin >> bar1 >> bar2;

ScreedRidge(mtrx, n, bar1 - 1, bar2 - 1, Matr1);

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

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

cout << std::setw(2) << mtrx2[i][j] << ' ';

}

cout << endl;

}

prList(Matr2, m);

std::cout << "\n";

}

else {

std::cout << "Такого нет(\n";

std::cout << "\n";

}

break;

case 3:

std::cout << "\n";

std::cout << "Выберите матрицу 1 или 2: ";

std::cin >> provided;

if (provided == 1) {

std::cout << "Введите вершину для расщепления: ";

std::cin >> bar1;

SplittingApex(mtrx, n, bar1 - 1, Matr1);

std::cout << "\n";

}

else if (provided == 2) {

std::cout << "Введите вершину для расщепления: ";

std::cin >> bar1;

SplittingApex(mtrx2, m, bar1 - 1, Matr2);

std::cout << "\n";

}

else {

std::cout << "Такого нет(\n";

std::cout << "\n";

}

break;

case 4:

std::cout << "\n";

UnificationGrafs(mtrx, mtrx2, n, m);

std::cout << "\n";

break;

case 5:

std::cout << "\n";

IntersectionGrafs(mtrx, mtrx2, n, m);

std::cout << "\n";

break;

case 6:

std::cout << "\n";

RingSumGrafs(mtrx, mtrx2, n, m);

std::cout << "\n";

break;

case 7:

std::cout << "\n";

DecGrafs(mtrx, mtrx2, n, m);

std::cout << "\n";

break;

case 8:

std::cout << "\n";

std::cout << "Спасибо за то, что посмотрели наш код, надеюсь вам всё понравилось!!!!\n";

std::cout << "\n";

break;

default:

std::cout << "\n";

std::cout << "Такого нет(\n";

std::cout << "\n";

break;

}

} while (option != 8);

// Освобождение памяти

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

delete[] mtrx[i];

frList(Matr1[i]);

}

delete[] mtrx;

delete[] Matr1;

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

delete[] mtrx2[i];

frList(Matr2[i]);

}

delete[] mtrx2;

delete[] Matr2;

return 0;

}