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»

Выполнили

студенты группы 22ВВВ1:

Ганин И.Р.

Курушин Я.С.

Приняли:

К.т.н, доцент Акифьев И. В.

К.т.н, доцент Юрова О. В.

Пенза 2023

**Цель работы**

Научится выполнять бинарные и унарные операции 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.

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.

**Ход работы**

**Теоретическая часть**

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, декартово произведение, кольцевая сумма

**Практическая часть**

Задание 1:

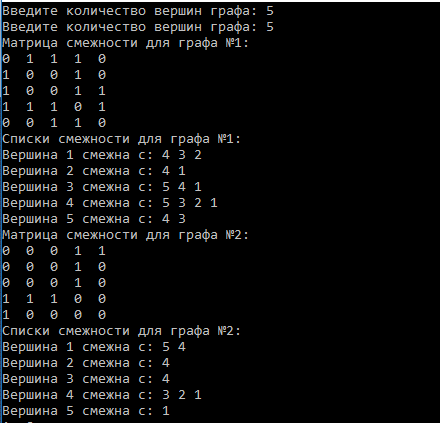
****

Рисунок 1- создали 2 матрицы смежности и списки смежности

Задание 2:

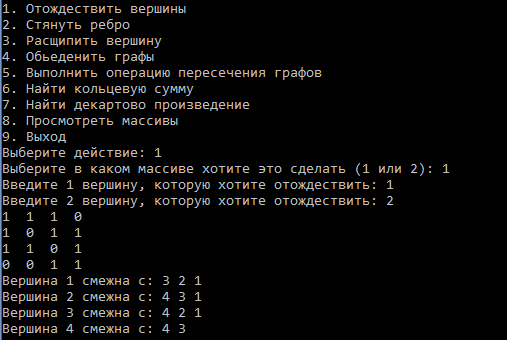


Рисунок 2- отождествили 1 и 2 вершины в первом графе

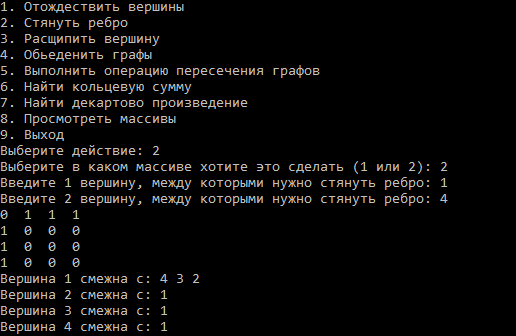


Рисунок 3- стянули ребро между 1 и 4 вершинами во втором графе

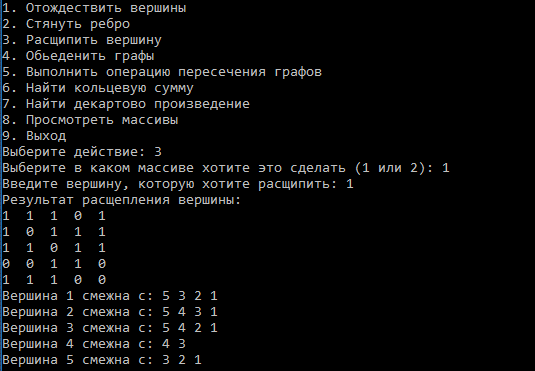


Рисунок 4- отождествили вершину 1 в первом графы

Задание 3:

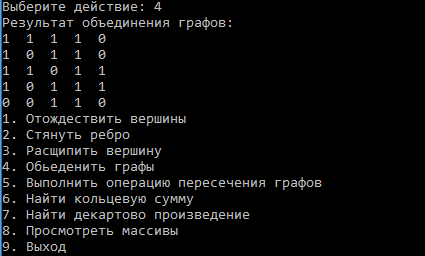


Рисунок 5- объединили графы

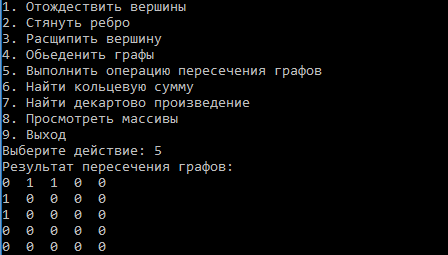


Рисунок 6- выполнили пересечение графов

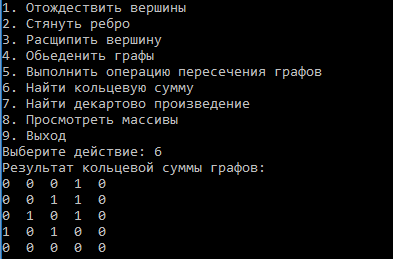


Рисунок 7- нашли кольцевую сумму

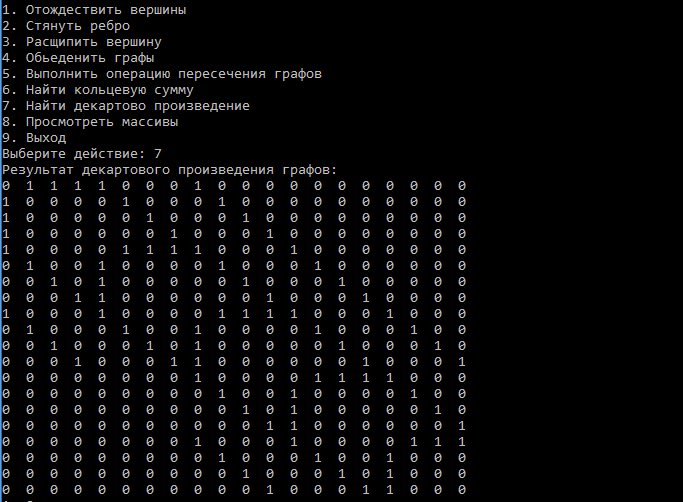


Рисунок 8- нашли декартово произведение

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

**Приложение А  
Листинг**

**Файл Source.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

typedef struct Node {

int inf;

struct Node\* next;

}Node;

void addSp(Node\*\* arr\_sp, int\*\* arr, int rows) {

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

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

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

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->inf = j + 1;

newNode->next = arr\_sp[i];

arr\_sp[i] = newNode;

}

}

}

}

void printSp(Node\*\* arr\_sp, int rows) {

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

Node\* current = arr\_sp[i];

printf("Вершина %d смежна с: ", i + 1);

while (current != NULL) {

printf("%d ", current->inf);

current = current->next;

}

printf("\n");

}

}

void freeList(Node\* head) {

Node\* current = head;

while (current != NULL) {

Node\* temp = current;

current = current->next;

free(temp);

}

}

void mergeRows(int\*\* arr, int\* rows, int rows1, int rows2, Node\*\* arr\_sp) {

// Проверка

if (rows1 >= \*rows || rows2 >= \*rows || rows1 < 0 || rows2 < 0) {

printf("Ошибка: некорректные номера вершин!\n");

return;

}

// Отождествляем вершины путем присвоения им одного и того же значения

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

arr[i][rows1] = arr[i][rows1] || arr[i][rows2];

}

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

arr[rows1][i] = arr[rows1][i] || arr[rows2][i];

}

// Сдвигаем вершины справа от rows2 на одну позицию влево

for (int i = 0; i < \*rows - 1; i++) {

for (int j = rows2; j < \*rows - 1; j++) {

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

}

}

for (int i = rows2; i < \*rows - 1; i++) {

for (int j = 0; j < \*rows - 1; j++) {

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

}

}

// Уменьшаем количество вершин на 1

(\*rows)--;

for (int i = 0; i < \*rows+1; i++) {

freeList(arr\_sp[i]);

arr\_sp[i] = NULL;

}

addSp(arr\_sp, arr, \*rows);

}

void contractEdge(int\*\* arr, int\* rows, int rows1, int rows2, Node\*\* arr\_sp) {

// Стягивание ребра

if (rows1 >= \*rows || rows2 >= \*rows || rows1 < 0 || rows2 < 0) {

printf("Ошибка: некорректные номера вершин!\n");

return;

}

if (arr[rows1][rows2] == 0) {

printf("Между вершинами должно быть ребро!\n");

return;

}

// Отождествляем вершины путем присвоения им одного и того же значения

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

arr[i][rows1] = arr[i][rows1] || arr[i][rows2];

}

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

arr[rows1][i] = arr[rows1][i] || arr[rows2][i];

}

arr[rows1][rows1] = 0;

// Сдвигаем вершины справа от vertex2 на одну позицию влево

for (int i = 0; i < \*rows - 1; i++) {

for (int j = rows2; j < \*rows - 1; j++) {

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

}

}

for (int i = rows2; i < \*rows - 1; i++) {

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

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

}

}

// Уменьшаем количество вершин на 1

(\*rows)--;

for (int i = 0; i < \*rows + 1; i++) {

freeList(arr\_sp[i]);

arr\_sp[i] = NULL;

}

addSp(arr\_sp, arr, \*rows);

}

void deleteRows(int\*\* arr, int\* rows, int rows1, Node\*\* arr\_sp) {

// расщепление вершины

int new\_rows = \*rows + 1;

int\*\* arr\_split = (int\*\*)malloc(new\_rows \* sizeof(int\*));

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

arr\_split[i] = (int\*)malloc(new\_rows \* sizeof(int));

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

if (i < \*rows && j < \*rows) {

arr\_split[i][j] = arr[i][j]; // копируем связи исходной матрицы

}

else if (i == \*rows && j < \*rows) {

arr\_split[i][j] = arr[rows1][j]; // новая вершина соединена со смежными вершинами выбранной вершины

}

else if (i < \*rows && j == \*rows) {

arr\_split[i][j] = arr[i][rows1]; // смежные вершины соединены с новой вершиной

}

else {

arr\_split[i][j] = 0; // новая вершина не связана с остальными вершинами

}

}

}

(\*rows)++;

arr = (int\*\*)malloc(\*rows \* sizeof(int\*));

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

arr[i] = (int\*)malloc(\*rows \* sizeof(int));

}

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

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

arr[i][j] = arr\_split[i][j];

}

}

printf("Результат расщепления вершины:\n");

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

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

printf("%-2d ", arr[i][j]);

}

printf("\n");

}

for (int i = 0; i < \*rows - 1; i++) {

freeList(arr\_sp[i]);

arr\_sp[i] = NULL;

}

arr\_sp = (Node\*\*)malloc((\*rows) \* sizeof(Node\*));

if (arr\_sp == NULL) {

printf("Не удалось выделить память!\n");

return;

}

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

arr\_sp[i] = NULL;

}

addSp(arr\_sp, arr, \*rows);

printSp(arr\_sp, \*rows);

}

void combineArr(int\*\* arr, int\*\* arr\_2, int rows, int rows\_2) { // обьединение графов

int max\_rows = (rows > rows\_2) ? rows : rows\_2; // максимальный размер для объединенного массива

int\*\* arr\_union = (int\*\*)malloc(max\_rows \* sizeof(int\*));

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

arr\_union[i] = (int\*)malloc(max\_rows \* sizeof(int));

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

int val\_1 = (i < rows && j < rows) ? arr[i][j] : 0; // значение из arr, если индексы в пределах размеров массива

int val\_2 = (i < rows\_2 && j < rows\_2) ? arr\_2[i][j] : 0; // значение из arr\_2, если индексы в пределах размеров массива

arr\_union[i][j] = val\_1 || val\_2; // логическое ИЛИ значений

}

}

printf("Результат объединения графов:\n");

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

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

printf("%-2d ", arr\_union[i][j]);

}

printf("\n");

}

}

void crossArr(int\*\* arr, int\*\* arr\_2, int rows, int rows\_2) { // пересечение графов

int max\_rows = (rows > rows\_2) ? rows : rows\_2;

int\*\* arr\_intersect = (int\*\*)malloc(max\_rows \* sizeof(int\*));

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

arr\_intersect[i] = (int\*)malloc(max\_rows \* sizeof(int));

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

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

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

arr\_intersect[i][j] = val\_1 && val\_2;

}

}

printf("Результат пересечения графов:\n");

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

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

printf("%-2d ", arr\_intersect[i][j]);

}

printf("\n");

}

}

void sumArr(int\*\* arr, int\*\* arr\_2, int rows, int rows\_2) { // кольцевая сумма

int max\_rows = (rows > rows\_2) ? rows : rows\_2;

int\*\* arr\_ring\_sum = (int\*\*)malloc(max\_rows \* sizeof(int\*));

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

arr\_ring\_sum[i] = (int\*)malloc(max\_rows \* sizeof(int));

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

if (i < rows && i < rows\_2 && j < rows && j < rows\_2) {

if (i==j || (arr[i][j] == 1 && arr\_2[i][j] == 1)) {

arr\_ring\_sum[i][j] = 0;

}

else {

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

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

arr\_ring\_sum[i][j] = val\_1 || val\_2;

}

}

else {

arr\_ring\_sum[i][j] = 0;

}

}

}

printf("Результат кольцевой суммы графов:\n");

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

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

printf("%-2d ", arr\_ring\_sum[i][j]);

}

printf("\n");

}

}

void decartArr(int\*\* arr, int\*\* arr\_2, int rows, int rows\_2) { // декартово произведение

int decart\_rows = rows \* rows\_2;

int decart\_cols = rows \* rows\_2;

int\*\* arr\_decart = (int\*\*)malloc(decart\_rows \* sizeof(int\*)); // Результирующий граф G

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

arr\_decart[i] = (int\*)malloc(decart\_cols \* sizeof(int));

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

arr\_decart[i][j] = 0;

}

}

int u = 0, v = 0;

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

for (int j = 0; j < rows\_2; j++) {

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

for (int l = 0; l < rows\_2; l++) {

u = i \* rows\_2 + j;

v = k \* rows\_2 + l;

if (j == l) {

arr\_decart[u][v] = arr[i][k]; // Создаем ребро в G, если есть рёбра в G1 и G2

}

if (i == k) {

arr\_decart[u][v] = arr\_2[j][l];

}

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

arr\_decart[u][v] = 0;

}

}

}

}

}

// Вывод результата на экран

printf("Результат декартового произведения графов:\n");

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

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

printf("%-2d ", arr\_decart[i][j]);

}

printf("\n");

}

}

void main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int\*\* arr = 0; // матрица смежности

int\*\* arr\_2 = 0; // матрица смежности 2

int i, j, rows, rows\_2, k, unit = 0;

printf("Введите количество вершин графа: ");

scanf("%d", &rows);

printf("Введите количество вершин графа: ");

scanf("%d", &rows\_2);

Node\*\* arr\_sp = NULL;

Node\*\* arr\_sp\_2 = NULL;

// Выделение памяти для массива списков смежности

arr\_sp = (Node\*\*)malloc(rows \* sizeof(Node\*));

if (arr\_sp == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows; i++) {

arr\_sp[i] = NULL;

}

arr\_sp\_2 = (Node\*\*)malloc(rows\_2 \* sizeof(Node\*));

if (arr\_sp\_2 == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows\_2; i++) {

arr\_sp\_2[i] = NULL;

}

arr = (int\*\*)malloc(rows \* sizeof(int\*));

if (arr == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows; i++) {

arr[i] = (int\*)malloc(rows \* sizeof(int));

}

// генерировать случайные значения для матрицы смежности

for (i = 0; i < rows; i++) {

for (j = i; j < rows; j++) {

if (i == j) {

arr[i][j] = 0; // на главной диагонали нули

}

else {

arr[i][j] = rand() % 2; // случайные значения 0 или 1

arr[j][i] = arr[i][j]; // симметрично заполнять значения для неориентированного графа

}

}

}

// выводить матрицу смежности на экран

printf("Матрица смежности для графа №1:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

printf("%-2d ", arr[i][j]);

}

printf("\n");

}

printf("Списки смежности для графа №1:\n");

addSp(arr\_sp, arr, rows);

printSp(arr\_sp, rows);

arr\_2 = (int\*\*)malloc(rows\_2 \* sizeof(int\*));

if (arr\_2 == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows\_2; i++) {

arr\_2[i] = (int\*)malloc(rows\_2 \* sizeof(int));

}

// генерировать случайные значения для матрицы смежности 2

for (i = 0; i < rows\_2; i++) {

for (j = i; j < rows\_2; j++) {

if (i == j) {

arr\_2[i][j] = 0; // на главной диагонали нули

}

else {

arr\_2[i][j] = rand() % 2; // случайные значения 0 или 1

arr\_2[j][i] = arr\_2[i][j]; // симметрично заполнять значения для неориентированного графа

}

}

}

printf("Матрица смежности для графа №2:\n");

// выводить матрицу смежности на экран

for (i = 0; i < rows\_2; i++) {

for (j = 0; j < rows\_2; j++) {

printf("%-2d ", arr\_2[i][j]);

}

printf("\n");

}

printf("Списки смежности для графа №2:\n");

addSp(arr\_sp\_2, arr\_2, rows\_2);

printSp(arr\_sp\_2, rows\_2);

int choice, rows1, rows2, what;

do {

printf("1. Отождествить вершины\n");

printf("2. Стянуть ребро\n");

printf("3. Расщипить вершину\n");

printf("4. Обьеденить графы\n");

printf("5. Выполнить операцию пересечения графов\n");

printf("6. Найти кольцевую сумму\n");

printf("7. Найти декартово произведение\n");

printf("8. Просмотреть массивы\n");

printf("9. Выход\n");

printf("Выберите действие: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Выберите в каком массиве хотите это сделать (1 или 2): ");

scanf("%d", &what);

if (what == 1) {

printf("Введите 1 вершину, которую хотите отождествить: ");

scanf("%d", &rows1);

printf("Введите 2 вершину, которую хотите отождествить: ");

scanf("%d", &rows2);

mergeRows(arr, &rows, rows1 - 1, rows2 - 1, arr\_sp);

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

printf("%-2d ", arr[i][j]);

}

printf("\n");

}

printSp(arr\_sp, rows);

}

else if (what == 2) {

printf("Введите 1 вершину, которую хотите отождествить: ");

scanf("%d", &rows1);

printf("Введите 2 вершину, которую хотите отождествить: ");

scanf("%d", &rows2);

mergeRows(arr\_2, &rows\_2, rows1 - 1, rows2 - 1, arr\_sp\_2);

for (i = 0; i < rows\_2; i++) {

for (j = 0; j < rows\_2; j++) {

printf("%-2d ", arr\_2[i][j]);

}

printf("\n");

}

printSp(arr\_sp\_2, rows\_2);

}

else {

printf("Некорректный выбор!\n");

break;

}

break;

case 2:

printf("Выберите в каком массиве хотите это сделать (1 или 2): ");

scanf("%d", &what);

if (what == 1) {

printf("Введите 1 вершину, между которыми нужно стянуть ребро: ");

scanf("%d", &rows1);

printf("Введите 2 вершину, между которыми нужно стянуть ребро: ");

scanf("%d", &rows2);

contractEdge(arr, &rows, rows1 - 1, rows2 - 1, arr\_sp);

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

printf("%-2d ", arr[i][j]);

}

printf("\n");

}

printSp(arr\_sp, rows);

}

else if (what == 2) {

printf("Введите 1 вершину, между которыми нужно стянуть ребро: ");

scanf("%d", &rows1);

printf("Введите 2 вершину, между которыми нужно стянуть ребро: ");

scanf("%d", &rows2);

contractEdge(arr\_2, &rows\_2, rows1 - 1, rows2 - 1, arr\_sp\_2);

for (i = 0; i < rows\_2; i++) {

for (j = 0; j < rows\_2; j++) {

printf("%-2d ", arr\_2[i][j]);

}

printf("\n");

}

printSp(arr\_sp\_2, rows\_2);

}

else {

printf("Некорректный выбор!\n");

break;

}

break;

case 3:

printf("Выберите в каком массиве хотите это сделать (1 или 2): ");

scanf("%d", &what);

if (what == 1) {

printf("Введите вершину, которую хотите расщипить: ");

scanf("%d", &rows1);

deleteRows(arr, &rows, rows1 - 1, arr\_sp);

}

else if (what == 2) {

printf("Введите вершину, которую хотите расщипить: ");

scanf("%d", &rows1);

deleteRows(arr\_2, &rows\_2, rows1 - 1, arr\_sp\_2);

}

else {

printf("Некорректный выбор!\n");

break;

}

break;

case 4:

combineArr(arr, arr\_2, rows, rows\_2);

break;

case 5:

crossArr(arr, arr\_2, rows, rows\_2);

break;

case 6:

sumArr(arr, arr\_2, rows, rows\_2);

break;

case 7:

decartArr(arr, arr\_2, rows, rows\_2);

break;

case 8:

printf("Матрица смежности для графа №1:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

printf("%-2d ", arr[i][j]);

}

printf("\n");

}

printSp(arr\_sp, rows);

printf("Матрица смежности для графа №2:\n");

// выводить матрицу смежности на экран

for (i = 0; i < rows\_2; i++) {

for (j = 0; j < rows\_2; j++) {

printf("%-2d ", arr\_2[i][j]);

}

printf("\n");

}

printSp(arr\_sp\_2, rows\_2);

break;

case 9:

system("cls");

printf("До свидания!\n");

break;

default:

printf("Некорректный выбор!\n");

break;

}

} while (choice != 9);

}