Министерство образование Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнила:

студентка группы 20ВВ1:

Зацепилина Е.С.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза 2021 г.

**Цель работы**: освоить алгоритмы проведения унарных и бинарных операций над графами в их представлении в виде матриц смежности и списков смежности.

**Задание на лабораторную работу:**

**Задание 1**

**1.** Сгенерируйте (используя генератор случайных чисел) две матрицы M1, М2 смежности неориентированных помеченных графов G1, G2. 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 = G1 G2

б) пересечения G = G1 G2

в) кольцевой суммы G = G1 G2

Результат выполнения операции выведите на экран.

**Задание 4**

**1.** Для матричной формы представления графов выполните операцию декартова произведения графов G = G1 X G2.

Результат выполнения операции выведите на экран.

**Ход выполнения задания:**

Для выполнения задания на лабораторную работу были написаны программы 3\_matr и 3\_spisok на языке С++ в среде Visual Studio 2019. Полный текст программы смотрите в п. «Листинг».

**Задания для представления графов в виде матриц смежности:**

**1.1** Для осуществления генерации матриц смежности для неориентированных графов применялась динамическая структура данных стандартной библиотеки «vector» - vector. Vector использовался для удобного и безопасного добавления и удаления строк\столбцов в матрицу смежности графа. С использованием этой структуры в памяти резервируется место для «вектора векторов», который имеет такую же структуру и доступ к элементам, как двумерный массив:

m1.resize(n1);

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

m1.at(i).resize(n1);

}

Заполнение матриц происходит случайным образом. В матрицу записываются 0 или 1 по следующим законам:

1) m[i][j] = m[j][i] – матрица симметрична относительно главной диагонали;

2) m[i][j] = 0, если i = j – главная диагональ всегда заполнена нулями;

3) Шанс выпадения 0 или 1 определяется соотношением 30/70 соответственно.

Заполнение матриц смежности для 2 графов:

srand(time(NULL));

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

for (int j = 0; j < n1; j++) {

if (i == j) {

m1[i][j] = 0;

break;

}

else {

int z = rand() % 101;

if (z < 30) { m1[i][j] = 0; }

else { m1[i][j] = 1; }

m1[j][i] = m1[i][j];

}

}

}

Результат выполнения процесса инициализации матриц смежности можно увидеть в п. «Результаты».

**2.1** Унарные операции для графов.

**а)** Отождествление вершин:

Операция отождествления вершин графа для матриц смежности выполняется путём логического сложения соответствующих столбцов и строк первой и второй вершины. Результат записывается в вершину, индекс которой в матрице смежности меньше. Вторая вершина удаляется из матрицы:

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

if ( i == v1 - 1 ) { arr[i][v1 - 1] = 0; }

else if (log\_sum(arr[v1 - 1][i], arr[v2 - 1][i])) { arr[v1 - 1][i] = 1; }

else { arr[v1 - 1][i] = 0; }

}

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

if ( i == v1 - 1 ) { arr[i][v1 - 1] = 0; }

else if (log\_sum(arr[i][v1 - 1], arr[i][v2 - 1])) { arr[i][v1 - 1] = 1; }

else { arr[i][v1 - 1] = 0; }

}

**б)** Стягивание ребра:

Стягивание ребра – частный случай отождествления вершин, где вершины смежные. Поэтому для реализации операции стягивания ребра достаточно проверить условия смежности двух вершин и осуществить их отождествление (см. п. «Листинг»).

**в)** Расщепление вершины:

Расщепление вершины осуществляется путём добавления к матрице строки и столбца, в которую записываются те же самые значения, что и в расщепляемой вершине.

n++;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

arr[i][n - 1] = arr[i][v - 1];

arr[n - 1][i] = arr[v - 1][i];

}

udalenie(arr, n, v2);

Результаты выполнения операций можно увидеть в п. «Результаты».

**3.1** Бинарные операции для графов.

**а)** Объединение:

Объединение графов для матриц смежности осуществляется путём поэлементного логического сложения элементов двух матриц. Для этого сначала нужно рассмотреть случай, когда матрицы смежности графов не равны по размерности. Для этого определяется размерность большей матрицы. Вторая матрица увеличивается до этих размеров. Все новые элементы заполняются нулями:

while (n1 != n2) {

if (n1 > n2) {

n2++;

arr2.resize(n2);

for (int i = 0; i < n2; i++) {

arr2[i].resize(n2, 0);

}

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

arr2[n2 - 1][i] = 0;

}

}

else if (n1 < n2) {

n1++;

arr1.resize(n1);

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

arr1[i].resize(n1, 0);

}

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

arr1[n1 - 1][i] = 0;

}

}

}

Затем создаётся новая матрица размерности n = n1, элементы которой равны логической сумме соответствующих элементов двух первых матриц:

n = n1;

vector < vector <int> > arr;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

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

if (log\_sum(arr1[i][j], arr2[i][j])) { arr[i][j] = 1; }

else { arr[i][j] = 0; }

}

}

**б)** Пересечение:

Пересечение графов для матриц смежности осуществляется путём поэлементного логического умножения элементов двух матриц. Для этого сначала нужно рассмотреть случай, когда матрицы смежности графов не равны по размерности. Для этого определяется размерность меньшей матрицы. Вторая матрица уменьшается до этих размеров (результатом умножения добавленных к меньшей матрице строк всегда будет равен 0, поэтому оптимальным решением будет уменьшить большую матрицу):

while (n1 != n2) {

if (n1 > n2) {

udalenie(arr1, n1, n1);

n1--;

}

else if (n1 < n2) {

udalenie(arr2, n2, n2);

n2--;

}

}

Затем создаётся новая матрица размерности n = n1, элементы которой равны логическому произведению соответствующих элементов двух первых матриц:

vector < vector <int> > arr;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

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

if (log\_pr(arr1[i][j], arr2[i][j])) { arr[i][j] = 1; }

else { arr[i][j] = 0; }

}

}

**в)** Кольцевая сумма:

Кольцевая сумма графов для матриц смежности осуществляется путём поэлементного осуществления логической операции исключающего или элементов двух матриц. Для этого сначала нужно рассмотреть случай, когда матрицы смежности графов не равны по размерности. Для этого определяется размерность большей матрицы. Вторая матрица увеличивается до этих размеров. Все новые элементы заполняются нулями (аналогично в п. «Объединение»). Затем создаётся новая матрица размерности n = n1, элементы которой равны исключающему или соответствующих элементов двух первых матриц:

vector < vector <int> > arr;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

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

if (iskl\_ili(arr1[i][j], arr2[i][j])) { arr[i][j] = 1; }

else { arr[i][j] = 0; }

}

}

Результаты выполнения операций можно увидеть в п. «Результаты».

**4\*.** Декартово произведение графов для матриц смежности.

Алгебраическую теорию графов можно использовать для анализа декартова произведения графов. Если граф G1 имеет n1 вершин и n1 x n1 матрицу смежности A1, а граф G2 имеет n2 вершин и n2 x n2 матрицу смежности A2, то матрица смежности декартова произведения двух графов задаётся формулой



где  означает произведение Кронекера матриц, а En означает n x n единичную матрицу.

Для реализации данной формулы сначала создаются единичные матрицы:

int ne1 = n1;

int ne2 = n2;

e1.resize(ne1);

for (int i = 0; i < ne1; i++) {

e1.at(i).resize(ne1);

}

e2.resize(ne2);

for (int i = 0; i < ne2; i++) {

e2.at(i).resize(ne2);

}

for (int i = 0; i < ne1; i++) {

for (int j = 0; j < ne1; j++) {

if (i == j) e1[i][j] = 1;

else e1[i][j] = 0;

}

}

for (int i = 0; i < ne2; i++) {

for (int j = 0; j < ne2; j++) {

if (i == j) e2[i][j] = 1;

else e2[i][j] = 0;

}

}

Затем инициализируем две матрицы, в которые запишутся результаты произведения Кронекера:

vector < vector <int> > arr2\_e1, arr1\_e2;

n = n2 \* n1;

arr2\_e1.resize(n);

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

arr2\_e1.at(i).resize(n);

}

arr1\_e2.resize(n);

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

arr1\_e2.at(i).resize(n);

}

Если A — матрица размера m×n, B — матрица размера p×q, тогда произведение Кронекера есть блочная матрица размера mp×nq. Для реализации этого произведения матриц элементы индексируются следующим образом:

C[i][j] = A[целое от деления i на размерность матрицы множителя][ целое от деления j на размерность матрицы множителя] \* B[остаток от деления i на размерность матрицы множителя][ остаток от деления j на размерность матрицы множителя].

Где C – результат произведения Кронекера матриц A и B. В программе это реализовано таким образом:

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

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

arr1\_e2[i][j] = arr1[i / ne2][j / ne2] \* e2[i % ne2][j % ne2];

}

}

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

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

arr2\_e1[i][j] = e1[i / n2][j / n2] \* arr2[i % n2][j % n2];

}

}

Затем происходит поэлементное сложение двух произведений Кронекера:

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

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

arr1\_e2[i][j] = arr1\_e2[i][j] + arr2\_e1[i][j];

}

}

Результаты выполнения операций можно увидеть в п. «Результаты».

**Задания для представления графов в виде списков смежности:**

**1.2** Для осуществления генерации списков смежности для неориентированных графов применялась динамическая структура данных стандартной библиотеки «vector» - vector. Vector использовался для удобного и безопасного добавления и удаления вершин в списке смежности графа. Также использовалась пользовательский класс \_queue:

class \_queue {

private:

//Класс элемента очереди

class item {

public:

int val;

item\* next;

item(int a) {

val = a;

next = NULL;

}

};

//Указатель на первый элемент очереди

item\* it\_first;

public:

int name; //Служит только для обозначения имени вершины графа

//Конструктор

\_queue() {

it\_first = NULL;

name = NULL;

}

\_queue(int a) {

it\_first = NULL;

name = a;

}

//Удаление первого эл-та очереди

void pop() {

if (!empty()) {

item\* it = it\_first->next;

delete it\_first;

it\_first = it;

}

}

//Добавление в конец очереди

void push(int a) {

if (it\_first == NULL) {

it\_first = new item(a);

} else {

item\* it1 = it\_first;

while (it1->next != NULL) {

it1 = it1->next;

}

it1->next = new item(a);

}

}

//Пуста ли очередь

boolean empty() {

if (it\_first == NULL) return true;

else return false;

}

//Отчистка очереди

void clear() {

while (!empty()) {

pop();

}

}

//Возвращает val первого элемента очереди

int first() {

if (empty()) return NULL;

else return it\_first->val;

}

//Удаляет из очереди элемент с конкретным val

void del\_crnt(int a) {

if (!empty()) {

if (it\_first->val == a) pop();

else if (it\_first->next != NULL) {

item\* it2 = it\_first;

item\* it1 = it\_first->next;

while ((it1->next != NULL)&&(it1->val != a)) {

it2 = it1;

it1 = it1->next;

}

if (it1->val == a) {

it2->next = it1->next;

delete it1;

}

}

}

}

//Есть ли в очереди элемент с конкретным val

boolean find(int a) {

if (!empty()) {

if (it\_first->val == a) return true;

else if (it\_first->next != NULL) {

item\* it1 = it\_first->next;

while ((it1->next != NULL) && (it1->val != a)) {

it1 = it1->next;

}

if (it1->val == a) {

return true;

} else return false;

}

} else return false;

}

//Вывод очереди

void vivod() {

item\* it = it\_first;

while (it != NULL) {

printf(" %d", it->val);

it = it->next;

}

}

};

Список смежности формируется на основе матрицы смежности графа (см. п. 1.1). Затем в памяти резервируется место для «вектора класса \_queue»:

vector <\_queue> g;

g.resize(n);

Затем для каждого элемента вектора g[i] записывается имя вершины name и в очередь добавляется вершина j + 1, если в матрице смежности элемент m[i][j] = 1:

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

g[i].name = i + 1;

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

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

g[i].push(j + 1);

}

}

}

Результат выполнения процесса инициализации списка смежности можно увидеть в п. «Результаты».

**2.2** Унарные операции графов.

**а)** Отождествление вершин:

Операция отождествления вершин для списка смежности осуществляется путём добавления в список первой вершины всех вершин из списка второй вершины и для каждой такой вершины добавляется первая вершина в конец очереди:

while (!g[v2\_i].empty()) {

if (!g[poisk\_name(g, g[v2\_i].first())].find(v1)) g[poisk\_name(g, g[v2\_i].first())].push(v1);

if (!g[v1\_i].find(g[v2\_i].first())) {

g[v1\_i].push(g[v2\_i].first());

}

g[v2\_i].pop();

}

Далее вторая вершина удаляется из графа:

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

g[i].del\_crnt(v2);

}

**б)** Стягивание ребра – частный случай отождествления вершин, где вершины смежные. Поэтому для реализации операции стягивания ребра достаточно проверить условия смежности двух вершин и осуществить их отождествление (см. п. «Листинг»).

**в)** Расщепление вершины:

Расщепление вершины для списка смежности осуществляется путём добавления в вектор g новой вершины. Новая вершина добавляется по закону:

Если между i и i - 1 вершинами существует промежуток (имена вершин увеличиваются больше, чем на 1), то новая вершина добавляется между вершинами. В ином случае вершина добавляется последней:

int v2 = g.size() + 1;

auto iter = g.cend();

int v1\_i = poisk\_name(g, v1);

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

if (g[i].name != g[i - 1].name + 1) {

iter = g.cbegin() + i;

v2 = g[i - 1].name + 1;

}

}

g.insert(iter, \_queue(v2));

int v2\_i = poisk\_name(g, v2);

Затем в новую вершину добавляются все вершины, которые есть в списке расщепляемой вершины и они сохраняются в вектор:

vector <int> tmp;

while (!g[v1\_i].empty()) {

g[v2\_i].push(g[v1\_i].first());

tmp.push\_back(g[v1\_i].first());

g[v1\_i].pop();

}

Затем вершины из вектора возвращаются в расщепляемую вершину:

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

g[v1\_i].push(tmp[i]);

}

Далее для каждой вершины в векторе добавляется вторая вершина:

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

g[poisk\_name(g, tmp[i])].push(v2);

}

**Вывод:** в результате выполнения лабораторной работы были получены навыки составления матриц и списков смежности для графов, осуществления унарных и бинарных операция для матричного представления графов и осуществления унарных операция для списочного представления графов.

**Листинг:**

**3\_matr.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <locale.h>

#include <windows.h>

#include <vector>

using namespace std;

//Функция вывода одной матрицы

void vivod\_m(vector <vector<int>> arr, int n) {

cout << (" |");

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

printf("%-2d|", i + 1);

}

cout << ("\n");

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

printf("|");

printf("%-2d|", i + 1);

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

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

}

printf("\n");

}

}

//Функция вывода двух матриц

void vivod(vector <vector<int>> arr1, vector <vector<int>> arr2, int n1, int n2) {

cout << "Матрицы смежности для графов G1 и G2.\n";

cout << "\nG1:\n";

vivod\_m(arr1, n1);

cout << "\nG2:\n";

vivod\_m(arr2, n2);

}

//Удаление вершины из матрицы

void udalenie(vector <vector<int>>& arr, int& n, int v) {

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

auto iter = arr[i].cbegin();

arr[i].erase(iter + v - 1);

}

auto iter = arr.cbegin();

arr.erase(iter + v - 1);

n--;

}

//Расщепление вершины

void rasheplenie(vector <vector<int>>& arr, int& n, int v) {

n++;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

arr[i][n - 1] = arr[i][v - 1];

arr[n - 1][i] = arr[v - 1][i];

}

}

//Удаление всех вершин, у которых отсутвуют рёбра (отчистка матрицы)

void clear(vector <vector<int>>& arr, int& n) {

int val;

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

val = 0;

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

if (arr[i][j] != 0) { val++; }

}

if (val == 0) { udalenie(arr, n, i + 1); }

}

}

//Логическое ИЛИ для целых 0 и 1

boolean log\_sum(int a, int b) {

if (a + b != 0) { return true; }

else { return false; }

}

//Логическое И для целых 0 и 1

boolean log\_pr(int a, int b) {

if (a \* b != 0) { return true; }

else { return false; }

}

//Исключающее или для целых 0 и 1

boolean iskl\_ili(int a, int b) {

if (a != b) { return true; }

else { return false; }

}

//Отождествление вершин

void otozhdestvlenie(vector <vector<int>>& arr, int& n, int& v1, int& v2) {

int num;

if (v1 > v2) {

num = v1;

v1 = v2;

v2 = num;

}

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

if ( i == v1 - 1 ) { arr[i][v1 - 1] = 0; }

else if (log\_sum(arr[v1 - 1][i], arr[v2 - 1][i])) { arr[v1 - 1][i] = 1; }

else { arr[v1 - 1][i] = 0; }

}

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

if ( i == v1 - 1 ) { arr[i][v1 - 1] = 0; }

else if (log\_sum(arr[i][v1 - 1], arr[i][v2 - 1])) { arr[i][v1 - 1] = 1; }

else { arr[i][v1 - 1] = 0; }

}

udalenie(arr, n, v2);

}

//Объединение двух матриц

vector <vector<int>> obiedinenie(vector <vector<int>> arr1, vector <vector<int>> arr2, int n1, int n2, int& n) {

while (n1 != n2) {

if (n1 > n2) {

n2++;

arr2.resize(n2);

for (int i = 0; i < n2; i++) {

arr2[i].resize(n2, 0);

}

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

arr2[n2 - 1][i] = 0;

}

}

else if (n1 < n2) {

n1++;

arr1.resize(n1);

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

arr1[i].resize(n1, 0);

}

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

arr1[n1 - 1][i] = 0;

}

}

}

n = n1;

vector < vector <int> > arr;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

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

if (log\_sum(arr1[i][j], arr2[i][j])) { arr[i][j] = 1; }

else { arr[i][j] = 0; }

}

}

return arr;

}

//Пересечение двух матриц

vector <vector<int>> peresechenie(vector <vector<int>> arr1, vector <vector<int>> arr2, int n1, int n2, int& n) {

while (n1 != n2) {

if (n1 > n2) {

udalenie(arr1, n1, n1);

n1--;

}

else if (n1 < n2) {

udalenie(arr2, n2, n2);

n2--;

}

}

n = n1;

vector < vector <int> > arr;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

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

if (log\_pr(arr1[i][j], arr2[i][j])) { arr[i][j] = 1; }

else { arr[i][j] = 0; }

}

}

return arr;

}

//Кольцевая сумма двух матриц

vector <vector<int>> summa(vector <vector<int>> arr1, vector <vector<int>> arr2, int n1, int n2, int& n) {

while (n1 != n2) {

if (n1 > n2) {

n2++;

arr2.resize(n2);

for (int i = 0; i < n2; i++) {

arr2[i].resize(n2, 0);

}

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

arr2[n2 - 1][i] = 0;

}

}

else if (n1 < n2) {

n1++;

arr1.resize(n1);

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

arr1[i].resize(n1, 0);

}

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

arr1[n1 - 1][i] = 0;

}

}

}

n = n1;

vector < vector <int> > arr;

arr.resize(n);

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

arr.at(i).resize(n);

}

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

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

if (iskl\_ili(arr1[i][j], arr2[i][j])) { arr[i][j] = 1; }

else { arr[i][j] = 0; }

}

}

return arr;

}

//Декартово произведение двух матриц

vector <vector<int>> proizv(vector <vector<int>> arr1, vector <vector<int>> arr2, int n1, int n2, int& n) {

vector < vector <int> > e1, e2;

int ne1 = n1;

int ne2 = n2;

e1.resize(ne1);

for (int i = 0; i < ne1; i++) {

e1.at(i).resize(ne1);

}

e2.resize(ne2);

for (int i = 0; i < ne2; i++) {

e2.at(i).resize(ne2);

}

for (int i = 0; i < ne1; i++) {

for (int j = 0; j < ne1; j++) {

if (i == j) e1[i][j] = 1;

else e1[i][j] = 0;

}

}

for (int i = 0; i < ne2; i++) {

for (int j = 0; j < ne2; j++) {

if (i == j) e2[i][j] = 1;

else e2[i][j] = 0;

}

}

vector < vector <int> > arr2\_e1, arr1\_e2;

n = n2 \* n1;

arr2\_e1.resize(n);

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

arr2\_e1.at(i).resize(n);

}

arr1\_e2.resize(n);

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

arr1\_e2.at(i).resize(n);

}

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

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

arr1\_e2[i][j] = arr1[i / ne2][j / ne2] \* e2[i % ne2][j % ne2];

}

}

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

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

arr2\_e1[i][j] = e1[i / n2][j / n2] \* arr2[i % n2][j % n2];

}

}

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

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

arr1\_e2[i][j] = arr1\_e2[i][j] + arr2\_e1[i][j];

}

}

return arr1\_e2;

}

int main(void)

{

char\* locale = setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

perez:

int n1 = 5;

int n2 = 5;

int n = 5;

int v1;

int v2;

int num;

vector < vector <int> > m1;

vector < vector <int> > m2;

vector < vector <int> > m;

new\_n:

m1.resize(n1);

m2.resize(n2);

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

m1.at(i).resize(n1);

}

for (int i = 0; i < n2; i++) {

m2.at(i).resize(n2);

}

srand(time(NULL));

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

for (int j = 0; j < n1; j++) {

if (i == j) {

m1[i][j] = 0;

break;

}

else {

int z = rand() % 101;

if (z < 30) { m1[i][j] = 0; }

else { m1[i][j] = 1; }

m1[j][i] = m1[i][j];

}

}

}

for (int i = 0; i < n2; i++) {

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

if (i == j) {

m2[i][j] = 0;

break;

}

else {

int z = rand() % 101;

if (z < 30) { m2[i][j] = 0; }

else { m2[i][j] = 1; }

m2[j][i] = m2[i][j];

}

}

}

clear(m1, n1);

clear(m2, n2);

begin:

vivod(m1, m2, n1, n2);

cout << "\n Выберите дальнейшее действие:\n";

cout << "1. Операции для одного графа.\n";

cout << "2. Операции для обоих графов.\n";

cout << "3. Обновить.\n";

cout << "4. Создать новые графы.\n";

cout << "5. Задать новое количество вершин для графов.\n";

cout << "6. Удалить вершины без рёбер.\n";

cout << "7. Закрыть программу.\n";

cout << "Ввод: ";

cin >> num;

switch (num) {

case 1:

system("cls");

vivod(m1, m2, n1, n2);

cout << ("\n Выберите дальнейшее действие:\n");

cout << ("1. Отождествление вершин.\n");

cout << ("2. Стягивание ребра.\n");

cout << ("3. Расщепление вершины.\n");

cout << ("4. Назад.\n");

cout << ("Ввод: ");

cin >> num;

switch (num) {

case 1:

system("cls");

vivod(m1, m2, n1, n2);

cout << "\n Выберите граф: ";

cin >> num;

cout << "\n Выберите первую вершину: ";

cin >> v1;

cout << "\n Выберите вторую вершину: ";

cin >> v2;

system("cls");

if (v1 == v2) { cout << "Невозможно.\n\n"; }

else {

if (num == 1) { otozhdestvlenie(m1, n1, v1, v2); clear(m1, n1);}

else { otozhdestvlenie(m2, n2, v1, v2); clear(m2, n2);}

cout << "Новая вершина записана в колонке(строке) " << v1 << ".\n\nРезультат:\n\n";

}

goto begin;

case 2:

system("cls");

vivod(m1, m2, n1, n2);

cout << ("\n Выберите граф: ");

cin >> num;

cout << ("\n Выберите первую вершину: ");

cin >> v1;

cout << ("\n Выберите вторую вершину: ");

cin >> v2;

system("cls");

if (v1 == v2) { cout << ("Невозможно.\n\n"); }

else if (m1[v1 - 1][v2 - 1] != m2[v1 - 1][v2 - 1]) { cout << ("Введённые вершины не смежные.\n\n"); }

else {

if (num == 1) { otozhdestvlenie(m1, n1, v1, v2); clear(m1, n1);}

else { otozhdestvlenie(m2, n2, v1, v2); clear(m2, n2);}

cout << "Новая вершина записана в колонке(строке) " << v1 << ".\n\nРезультат:\n\n";

}

goto begin;

case 3:

system("cls");

vivod(m1, m2, n1, n2);

cout << ("\n Выберите граф: ");

cin >> num;

cout << ("\n Выберите вершину: ");

cin >> v1;

if (num == 1) {

rasheplenie(m1, n1, v1);

clear(m1, n1);

} else {

rasheplenie(m2, n2, v1);

clear(m2, n2);

}

system("cls");

cout << "Вершина " << v1 << " расщеплена. Вторая вершина сохранена в последней строке(столбце) матрицы";

goto begin;

case 4:

system("cls");

goto begin;

}

case 2:

system("cls");

vivod(m1, m2, n1, n2);

cout << ("\n Выберите дальнейшее действие:\n");

cout << ("1. Объединение.\n");

cout << ("2. Пересечение.\n");

cout << ("3. Кольцевая сумма.\n");

cout << ("4. Декартово произведение.\n");

cout << ("5. Назад.\n");

cout << ("Ввод: ");

cin >> num;

switch (num) {

case 1:

system("cls");

m = obiedinenie(m1, m2, n1, n2, n);

clear(m, n);

cout << "Новая матрица.\n\n";

vivod\_m(m, n);

cout << "\n\n";

goto begin;

case 2:

system("cls");

m = peresechenie(m1, m2, n1, n2, n);

cout << "Новая матрица.\n\n";

vivod\_m(m, n);

cout << "\n\n";

goto begin;

case 3:

system("cls");

m = summa(m1, m2, n1, n2, n);

cout << "Новая матрица.\n\n";

vivod\_m(m, n);

cout << "\n\n";

goto begin;

goto begin;

case 4:

system("cls");

m = proizv(m1, m2, n1, n2, n);

cout << "Новая матрица.\n\n";

vivod\_m(m, n);

cout << "\n\n";

goto begin;

goto begin;

case 5:

system("cls");

goto begin;

}

case 3:

system("cls");

goto begin;

case 4:

system("cls");

goto perez;

case 5:

system("cls");

vivod(m1, m2, n1, n2);

cout << "\n Кол-во вершин для первого графа: ";

cin >> n1;

cout << "\n Кол-во вершин для второго графа: ";

cin >> n2;

system("cls");

goto new\_n;

case 6:

system("cls");

clear(m1, n1);

clear(m2, n2);

goto new\_n;

case 7:

return 0;

}

}

**3\_spisok.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <locale.h>

#include <windows.h>

#include <vector>

using namespace std;

class \_queue {

private:

//Класс элемента очереди

class item {

public:

int val;

item\* next;

item(int a) {

val = a;

next = NULL;

}

};

//Указатель на первый элемент очереди

item\* it\_first;

public:

int name; //Служит только для обозначения имени вершины графа

//Конструктор

\_queue() {

it\_first = NULL;

name = NULL;

}

\_queue(int a) {

it\_first = NULL;

name = a;

}

//Удаление первого эл-та очереди

void pop() {

if (!empty()) {

item\* it = it\_first->next;

delete it\_first;

it\_first = it;

}

}

//Добавление в конец очереди

void push(int a) {

if (it\_first == NULL) {

it\_first = new item(a);

} else {

item\* it1 = it\_first;

while (it1->next != NULL) {

it1 = it1->next;

}

it1->next = new item(a);

}

}

//Пуста ли очередь

boolean empty() {

if (it\_first == NULL) return true;

else return false;

}

//Отчистка очереди

void clear() {

while (!empty()) {

pop();

}

}

//Возвращает val первого элемента очереди

int first() {

if (empty()) return NULL;

else return it\_first->val;

}

//Удаляет из очереди элемент с конкретным val

void del\_crnt(int a) {

if (!empty()) {

if (it\_first->val == a) pop();

else if (it\_first->next != NULL) {

item\* it2 = it\_first;

item\* it1 = it\_first->next;

while ((it1->next != NULL)&&(it1->val != a)) {

it2 = it1;

it1 = it1->next;

}

if (it1->val == a) {

it2->next = it1->next;

delete it1;

}

}

}

}

//Есть ли в очереди элемент с конкретным val

boolean find(int a) {

if (!empty()) {

if (it\_first->val == a) return true;

else if (it\_first->next != NULL) {

item\* it1 = it\_first->next;

while ((it1->next != NULL) && (it1->val != a)) {

it1 = it1->next;

}

if (it1->val == a) {

return true;

} else return false;

}

} else return false;

}

//Вывод очереди

void vivod() {

item\* it = it\_first;

while (it != NULL) {

printf(" %d", it->val);

it = it->next;

}

}

};

//Поиск индекса очереди по имени

int poisk\_name(vector <\_queue> g, int v) {

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

if (g[i].name == v) return i;

}

return -1;

}

//Вывод графа

void vivod(vector <\_queue> g) {

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

printf("%d:", g[i].name);

g[i].vivod();

printf("\n");

}

}

//Освобождение памяти от очередей

void delete\_g(vector <\_queue> g) {

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

g[i].clear();

}

}

//Отождествление

void otozhdestvlenie(vector <\_queue>& g, int v1, int v2) {

int v1\_i = poisk\_name(g, v1);

int v2\_i = poisk\_name(g, v2);

while (!g[v2\_i].empty()) {

if (!g[poisk\_name(g, g[v2\_i].first())].find(v1)) g[poisk\_name(g, g[v2\_i].first())].push(v1);

if (!g[v1\_i].find(g[v2\_i].first())) {

g[v1\_i].push(g[v2\_i].first());

}

g[v2\_i].pop();

}

//Удаление вершины v2 из графа

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

g[i].del\_crnt(v2);

}

auto it = g.cbegin();

g.erase(it + v2\_i);

}

//Расщепление

int rasheplenie(vector <\_queue>& g, int v1) {

//Если есть свободное место в графе, то вторая вершина записывается в неё. Иначе к графу добавляется новая вершина.

int v2 = g.size() + 1;

auto iter = g.cend();

int v1\_i = poisk\_name(g, v1);

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

if (g[i].name != g[i - 1].name + 1) {

iter = g.cbegin() + i;

v2 = g[i - 1].name + 1;

}

}

g.insert(iter, \_queue(v2));

int v2\_i = poisk\_name(g, v2);

//Из v1 вершины добавляются в v2 и сохраняются в вектор

vector <int> tmp;

while (!g[v1\_i].empty()) {

g[v2\_i].push(g[v1\_i].first());

tmp.push\_back(g[v1\_i].first());

g[v1\_i].pop();

}

//Вершины v1 возвращаются

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

g[v1\_i].push(tmp[i]);

}

//Добавление v2 по всем ребрам

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

g[poisk\_name(g, tmp[i])].push(v2);

}

return v2;

}

int main(void)

{

char\* locale = setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

perez:

int n = 5;

int v1;

int v2;

int num;

vector < vector <int> > m;

vector <\_queue> g;

new\_n:

m.resize(n);

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

m.at(i).resize(n);

}

srand(time(NULL));

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

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

if (i == j) {

m[i][j] = 0;

break;

}

else {

int z = rand() % 101;

if (z < 30) { m[i][j] = 0; }

else { m[i][j] = 1; }

m[j][i] = m[i][j];

}

}

}

g.resize(n);

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

g[i].name = i + 1;

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

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

g[i].push(j + 1);

}

}

}

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

if (g[i].empty()) {

auto it = g.cbegin();

g.erase(it + i);

}

}

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

m[i].clear();

}

m.clear();

begin:

cout << "Список смежности для графа:\n";

vivod(g);

cout << "\n Выберите дальнейшее действие:\n";

cout << "1. Операции для одного графа.\n";

cout << "2. Обновить.\n";

cout << "3. Создать новый граф.\n";

cout << "4. Задать новое количество вершин для графа.\n";

cout << "5. Закрыть программу.\n";

cout << "Ввод: ";

cin >> num;

switch (num) {

case 1:

system("cls");

cout << "Список смежности для графа:\n";

vivod(g);

cout << ("\n Выберите дальнейшее действие:\n");

cout << ("1. Отождествление вершин.\n");

cout << ("2. Стягивание ребра.\n");

cout << ("3. Расщепление вершины.\n");

cout << ("4. Назад.\n");

cout << ("Ввод: ");

cin >> num;

switch (num) {

case 1:

system("cls");

cout << "Список смежности для графа:\n";

vivod(g);

cout << "\n Выберите первую вершину(в эту вершину запишется новая вершина): ";

cin >> v1;

cout << "\n Выберите вторую вершину: ";

cin >> v2;

system("cls");

if (v1 == v2) { cout << "Невозможно.\n\n"; }

else {

cout << "Граф до отождествления:\n\n";

vivod(g);

otozhdestvlenie(g, v1, v2);

cout << "\nНовая вершина записана в вершину " << v1 << ".\n\nРезультат:\n\n";

}

goto begin;

case 2:

system("cls");

cout << "Список смежности для графа:\n";

vivod(g);

cout << ("\n Выберите первую вершину: ");

cin >> v1;

cout << ("\n Выберите вторую вершину: ");

cin >> v2;

system("cls");

if (v1 == v2) { cout << ("Невозможно.\n\n"); }

else if (!g[poisk\_name(g, v1)].find(v2)) { cout << ("Введённые вершины не смежные.\n\n"); }

else {

cout << "Граф до стягивания:\n\n";

vivod(g);

otozhdestvlenie(g, v1, v2);

cout << "\nНовая вершина записана в вершину " << v1 << ".\n\nРезультат:\n\n";

}

goto begin;

case 3:

system("cls");

cout << "Список смежности для графа:\n";

vivod(g);

cout << ("\n Выберите вершину: ");

cin >> v1;

system("cls");

cout << "Граф до расщепления:\n\n";

vivod(g);

v2 = rasheplenie(g, v1);

cout << "\nВершина расщеплена на вершины " << v1 << " и " << v2 << ".\n\n";

goto begin;

case 4:

system("cls");

goto begin;

}

case 2:

system("cls");

goto begin;

case 3:

system("cls");

delete\_g(g);

goto perez;

case 4:

system("cls");

cout << "\n Кол-во вершин для графа: ";

cin >> n;

system("cls");

delete\_g(g);

goto new\_n;

case 5:

delete\_g(g);

return 0;

}

}

**Результаты:**

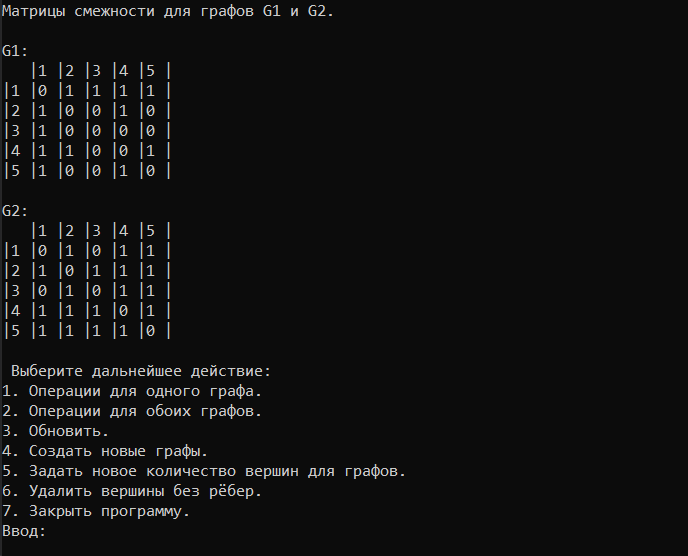


Рисунок 1 – Матрицы смежности для графов

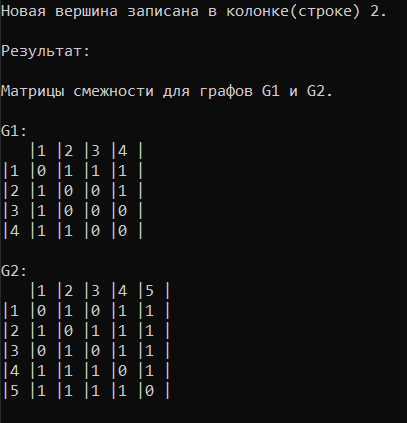
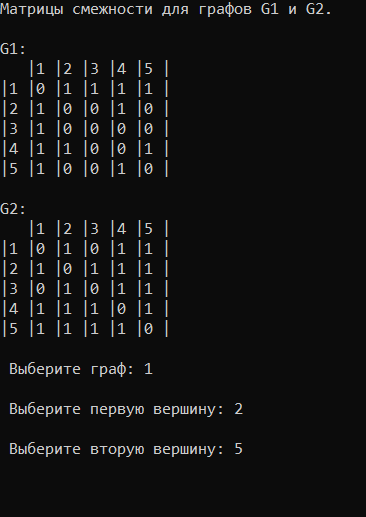


Рисунок 2 – Отождествление вершин

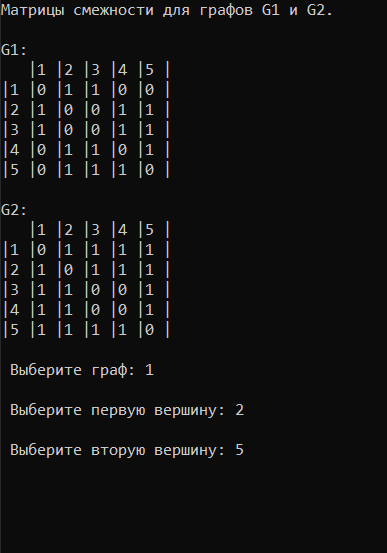
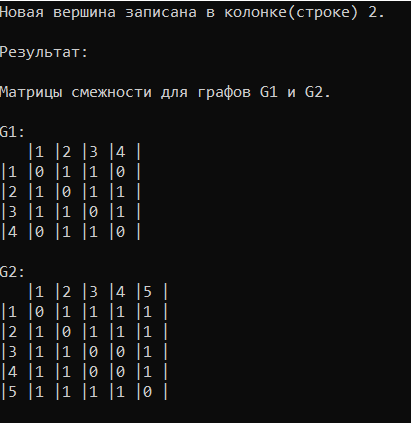
 

Рисунок 3 – Стягивание ребра

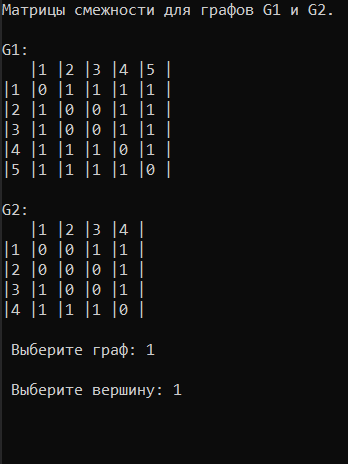
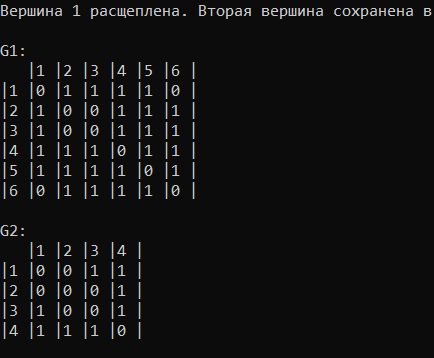
 

Рисунок 4 – Расщепление вершины

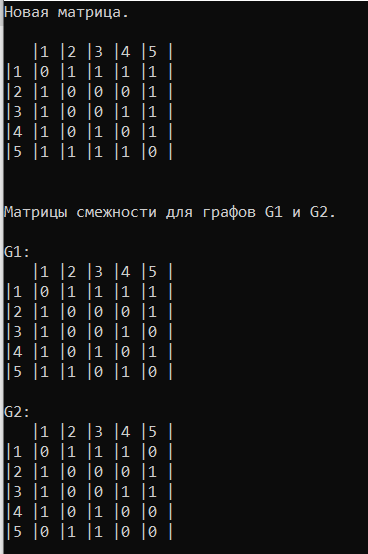


Рисунок 5 – Объединение

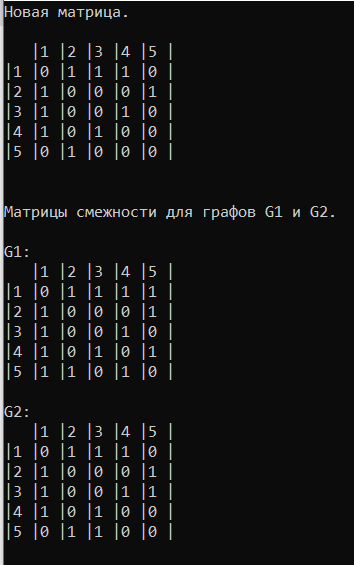


Рисунок 6 – Пересечение

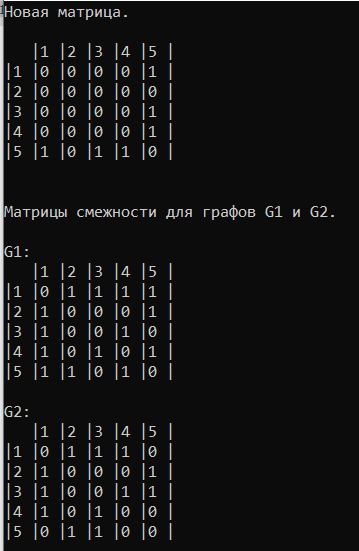


Рисунок 7 – Кольцевая сумма

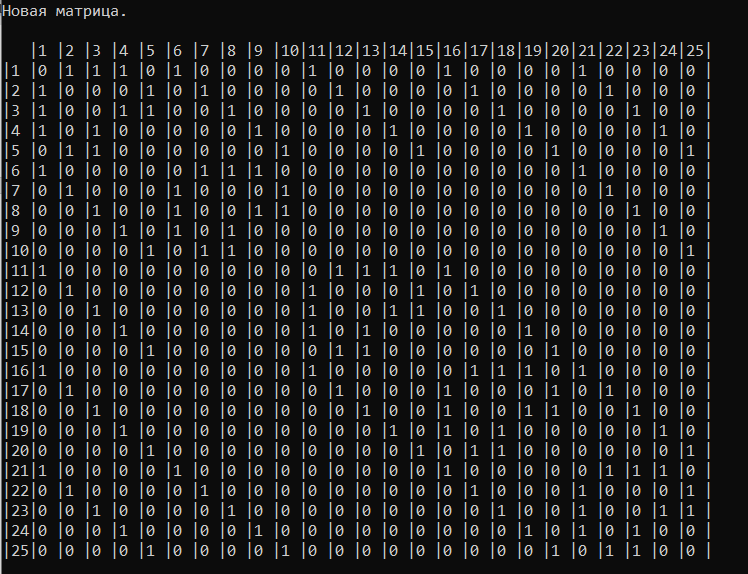


Рисунок 8 – Декартово произведение

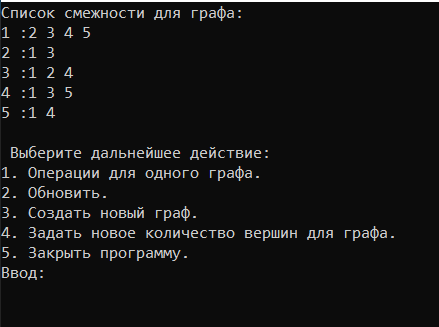
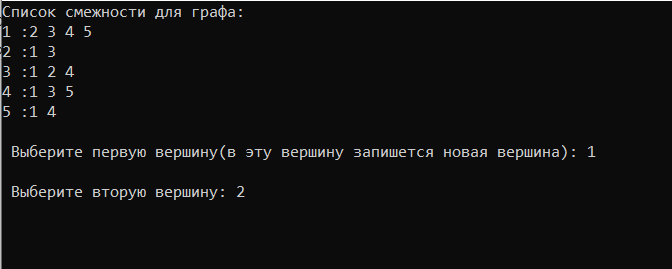


Рисунок 9 – Список смежности



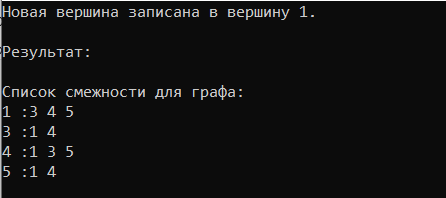


Рисунок 10 – Отождествление вершин

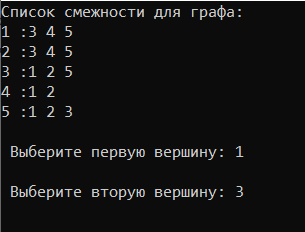
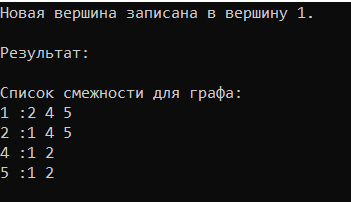
 

Рисунок 11 – Стягивание ребра

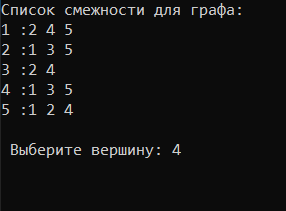
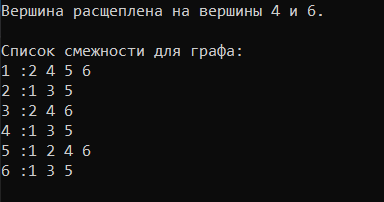
 

Рисунок 12 – Расщепление вершины