Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

## на тему «Унарные и бинарные операции над графами»

Выполнили:

студенты группы 20ВВ2

Киреев Б.П.

Верховский М.В.

Лукин В.Д.

Приняли:

д.т.н. Митрохин М.А.

к.т.н. доцент Юрова О.В.

Пенза 2021

Название: Унарные и бинарные операции над графами.

Цель работы: Научиться осуществлять унарные и бинарные операции над графами для их матричной формы представления.

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 = G1G2

б) пересечения G = G1G2

в) кольцевой суммы G = G1  G2

Результат выполнения операции выведите на экран.

\***Задание 4:**

Для матричной формы представления графов выполните операцию декартова произведения графов G = G1 x G2.

Результаты выполнения операции выведите на экран.

**Задание 1, 3, 4:**

Листинг:

Файл Lab3(1,3,4).cpp

#include <iostream>

#include <ctime>

#include <stdio.h>

#include <malloc.h>

#include <conio.h>

#include <math.h>

using namespace std;

int n, \*\* Matrix1 = NULL, \*\* Matrix2 = NULL;

int pogr = 0;

int choose = 0, \*\* Matrix3 = NULL, \*\* Matrix4 = NULL;

//Вершина / След.элемент; Номер

struct SmegnElem {

SmegnElem\* Nextelem;

int num;

};

//Список смежности

struct SmegnList {

SmegnElem\* First;

int Colo;

} \*\*SpecMatrix = NULL, \*\* SpecMatrix2 = NULL;

//Создание списка смежности

SmegnList\* CreateList() {

SmegnList\* groups = (SmegnList\*)malloc(sizeof(SmegnList));

groups->First = NULL;

groups->Colo = 0;

return groups;

}

//Добавление элемента в список смежности

void AddSmegElem(SmegnList\* group, int Chis) {

SmegnElem\* newItem = (SmegnElem\*)malloc(sizeof(SmegnElem));

newItem->Nextelem = NULL;

newItem->num = Chis;

if (group->Colo == 0) { //При отсутствии элементов записываем как первый

group->First = newItem;

group->Colo++;

return;

}

SmegnElem\* last = group->First; //При наличии других элементов начинаем перебор

while (last->Nextelem != NULL) {

last = last->Nextelem;

}

last->Nextelem = newItem;

group->Colo++;

}

//Отрисовка списка смежности

void PrintSmegElem(SmegnList\* groups) {

if (groups->Colo == 0) { //При размере = 0, у нас нет данных

cout << "Список пуст" << endl;

return;

}

SmegnElem\* current = groups->First;

while (current != NULL) { //Перебор структуры до последнего элемента(с выводом)

int value = current->num;

cout << value << " ";

current = current->Nextelem;

}

cout << endl;

}

void Obiedinenie() {

cout << "Объединение матриц 1 и 2:" << endl;

//Объединение

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

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

if (Matrix1[i][m] == 0 && Matrix2[i][m] == 0) {

Matrix3[i][m] = 0;

}

else {

Matrix3[i][m] = 1;

}

}

}

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

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

cout << Matrix3[i][m] << " ";

}

cout << endl;

}

}

void Peresechenie() {

cout << "Пересечение матриц 1 и 2:" << endl;

//Пересечение

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

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

if (Matrix1[i][m] == 1 && Matrix2[i][m] == 1) {

Matrix3[i][m] = 1;

}

else {

Matrix3[i][m] = 0;

}

}

}

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

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

cout << Matrix3[i][m] << " ";

}

cout << endl;

}

}

void ShapeSumm() {

cout << "Кольцевая сумма матриц 1 и 2:" << endl;

//Кольцевая сумма

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

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

if ((Matrix1[i][m] == 0 && Matrix2[i][m] == 0) || (Matrix1[i][m] == 1 && Matrix2[i][m] == 1)) {

Matrix3[i][m] = 0;

}

else {

Matrix3[i][m] = 1;

}

}

}

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

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

cout << Matrix3[i][m] << " ";

}

cout << endl;

}

}

void DekartProizv() {

int pogro = 0;

int x = 1, y = 1, x1 = 0, x2;

int koef = 1, speca = 1, yepo = 0, depo = 1;

int contain = 0;

cout << "Декартово произведение матриц 1 и 2:" << endl;

//Декартово произведение

Matrix4 = (int\*\*)calloc(n \* n, 3);

for (int i = 0; i < n \* n; i++) { Matrix4[i] = (int\*)calloc(n \* n, 3); }

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

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

Matrix4[i][j] = x \* 10 + y;

x++;

}

y++;

pogro++;

x = pogro + 1;

}

pogro = 0;

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

koef = i / n + 1;

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

x = Matrix4[i][j] / 10;

y = Matrix4[i][j] % 10;

x2 = i + 1;

if (x + y <= (n \* 2) \* koef && x <= n \* koef && y <= n \* koef) { Matrix4[i][j] = Matrix2[x - (n \* (koef - 1)) - 1][y - (n \* (koef - 1)) - 1]; }

else if (x2 + n \* speca == x) {

x1++;

Matrix4[i][j] = Matrix1[yepo][depo];

speca++;

if (x1 == 3) { x1 = 0; depo++; }

}

else { Matrix4[i][j] = 0; }

Matrix4[j][i] = Matrix4[i][j];

}

Matrix4[i][i] = 0;

speca = 1;

pogro++;

if (depo <= 2) {}

else { yepo = 1; depo = 2; }

}

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

Matrix4[i][i] = 0;

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

cout.width(4);

cout << Matrix4[i][j];

}

cout << endl;

}

}

void DekartProizv2() {

int x = -1, y = -1;

Matrix4 = (int\*\*)calloc(n \* n, 3);

for (int i = 0; i < n \* n; i++) { Matrix4[i] = (int\*)calloc(n \* n, 3); }

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

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

y = -1;

x++;

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

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

y++;

if (i == j) { Matrix4[x][y] = Matrix2[k][l]; }

else { Matrix4[x][y] = 0; }

if (k == l) { Matrix4[x][y] = Matrix1[i][j]; }

}

}

}

}

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

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

cout.width(4);

cout << Matrix4[i][j];

}

cout << endl;

}

}

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(0));

//Задание 1//

cout << "Введите размерность матриц: ";

cin >> n;

if (n <= 0) { cout << "\nНекорректный ввод. Повторите попытку" << endl; return 0; }

Matrix1 = (int\*\*)calloc(n, 3);

Matrix2 = (int\*\*)calloc(n, 3);

SpecMatrix = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SpecMatrix2 = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

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

Matrix1[j] = (int\*)calloc(n, 3);

Matrix2[j] = (int\*)calloc(n, 3);

SpecMatrix[j] = CreateList(); //Создаём списки смежности

SpecMatrix2[j] = CreateList();

AddSmegElem(SpecMatrix[j], j + 1); //Вставляем в начало каждого списка смежности номер отвечающей вершины

AddSmegElem(SpecMatrix2[j], j + 1);

}

cout.precision(3 \* n);

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

for (int m = 0 + pogr; m < n; m++) {

Matrix1[i][m] = 0 + rand() % 2;

Matrix1[m][i] = Matrix1[i][m];

Matrix1[i][i] = 0;

Matrix2[i][m] = 0 + rand() % 2;

Matrix2[m][i] = Matrix2[i][m];

Matrix2[i][i] = 0;

}

pogr++;

}

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

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

if (Matrix1[i][m] == 1) { AddSmegElem(SpecMatrix[i], m + 1); }

if (Matrix2[i][m] == 1) { AddSmegElem(SpecMatrix2[i], m + 1); }

}

}

//Задание 3//

while (choose != 5) {

system("cls");

cout << "\nМатрица №1:" << endl;

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

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

cout << Matrix1[i][m] << " ";

}

cout << endl;

}

cout << "\nСписок смежности для Мартицы №1:" << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix[i]); }

cout << "\nМатрица №2:" << endl;

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

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

cout << Matrix2[i][m] << " ";

}

cout << endl;

}

cout << "\nСписок смежности для Мартицы №2:" << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix2[i]); }

cout << "\n1.Объединение" << endl;

cout << "2.Пересечение" << endl;

cout << "3.Кольцевая сумма" << endl;

cout << "4.Декартово произведение" << endl;

cout << "5.Выход\n" << endl;

cout << "Выберите операцию для выполнения: ";

cin >> choose;

cout << endl;

Matrix3 = (int\*\*)calloc(n, 3);

for (int j = 0; j < n; j++) { Matrix3[j] = (int\*)calloc(n, 3); }

switch (choose) {

case 1://Объединение

Obiedinenie();

\_getch();

break;

case 2://Пересечение

Peresechenie();

\_getch();

break;

case 3://Кольцевая сумма

ShapeSumm();

\_getch();

break;

case 4://Декартово произведение

DekartProizv2();

cout << endl;

DekartProizv();

\_getch();

break;

case 5:

cout << "\nВыход из программы" << endl;

break;

default:

cout << "\nНекорректный ввод. Повторите попытку." << endl;

break;

}

}

return 0;

}

Пояснительный текст к программе:

Данная программа осуществляет: генерацию двух матриц смежности неориентированных графов и списки смежности к ним(Рис.№1); для матричной формы графов выполняет операцию объединения (Рис.№2), пересечения (Рис.№3), кольцевой суммы (Рис.№4), декартова объединения (Рис.№5)

Результаты работы программы:

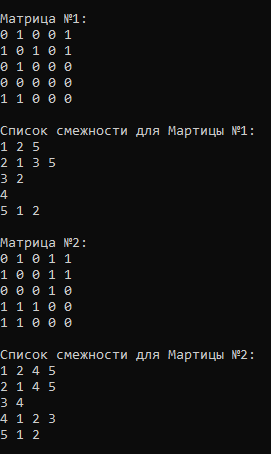


Рисунок №1 – Работа программы из задания №1.

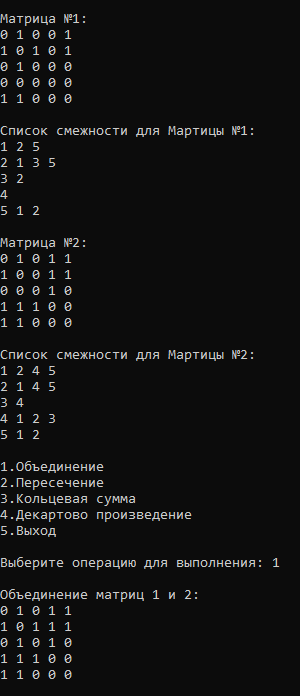


Рисунок №2 – Работа программы из задания №3 (объединение).

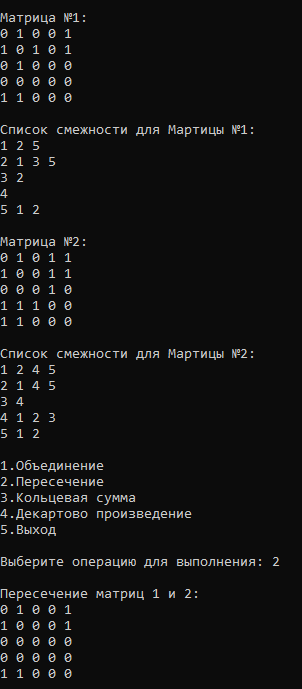


Рисунок №3 – Работа программы из задания №3 (пересечение).

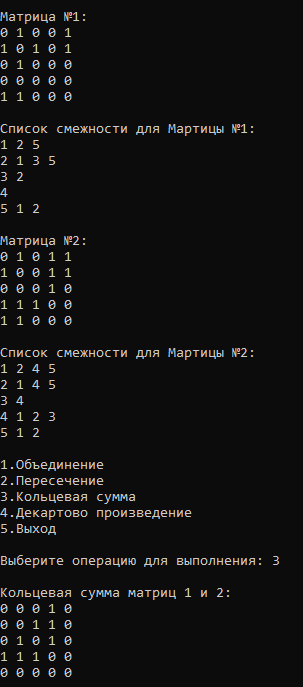


Рисунок №4 – Работа программы из задания №3 (кольцевая сумма).

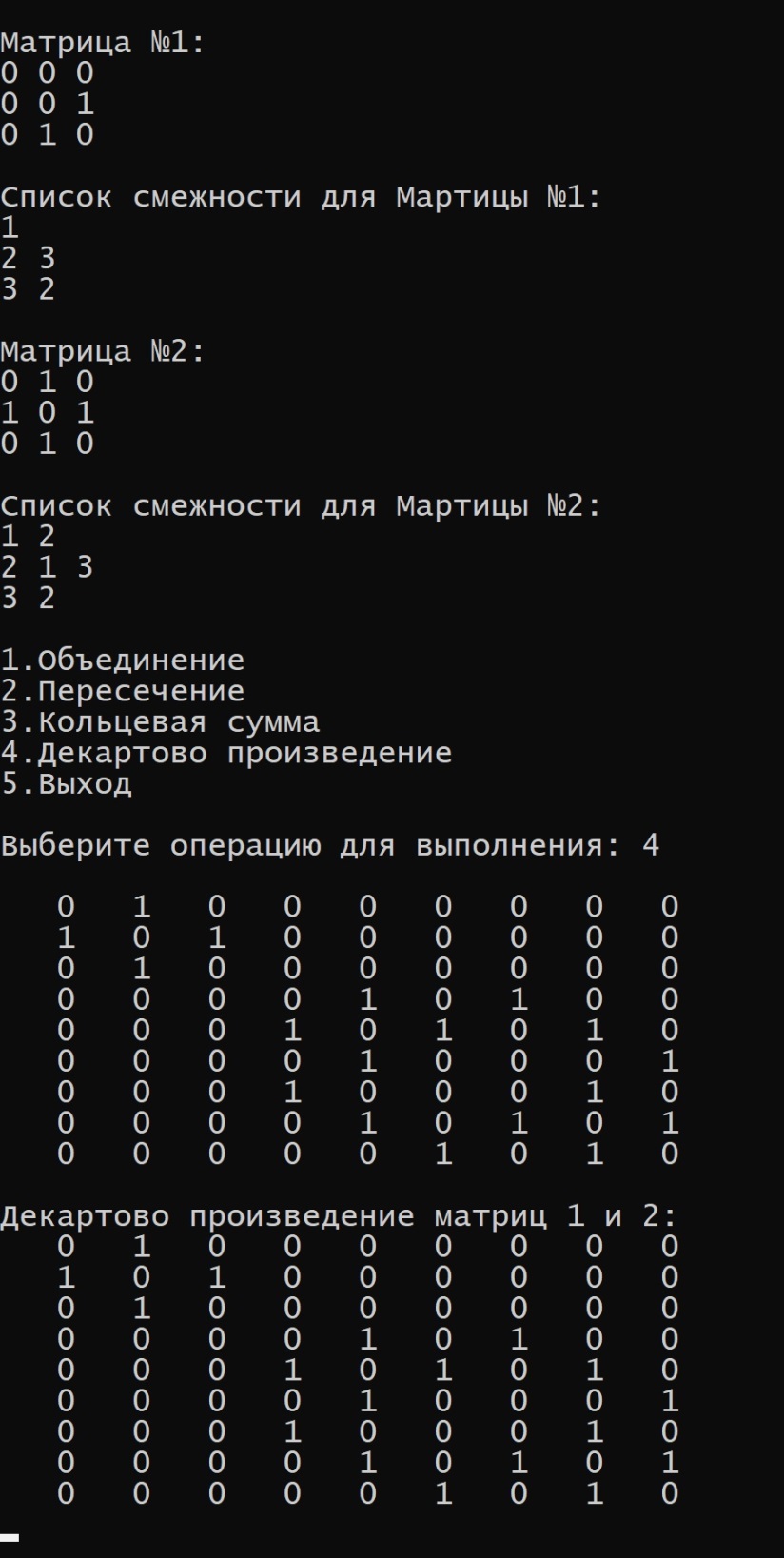


Рисунок №5 – Работа программы из задания №4 (декартево произведение).

**Задание 2:**

Листинг:

Файл Lab3(2).cpp

#include <iostream>

#include <ctime>

#include <stdio.h>

#include <malloc.h>

#include <conio.h>

using namespace std;

int n, \*\* Matrix1 = NULL;

int pogr = 0, usrentr, usrentr2;

int choose = 0, \*\* Matrix3 = NULL;

//Вершина / След.элемент; Номер

struct SmegnElem {

SmegnElem\* Nextelem;

int num;

};

//Список смежности

struct SmegnList {

SmegnElem\* First;

int Colo;

} \*\*SpecMatrix = NULL, \*\* SpecMatrixRezerve = NULL;

//Создание списка смежности

SmegnList\* CreateList() {

SmegnList\* groups = (SmegnList\*)malloc(sizeof(SmegnList));

groups->First = NULL;

groups->Colo = 0;

return groups;

}

//Добавление элемента в список смежности

void AddSmegElem(SmegnList\* group, int Chis) {

SmegnElem\* newItem = (SmegnElem\*)malloc(sizeof(SmegnElem));

newItem->Nextelem = NULL;

newItem->num = Chis;

if (group->Colo == 0) { //При отсутствии элементов записываем как первый

group->First = newItem;

group->Colo++;

return;

}

SmegnElem\* last = group->First; //При наличии других элементов начинаем перебор

while (last->Nextelem != NULL) {

last = last->Nextelem;

}

last->Nextelem = newItem;

group->Colo++;

}

//Отрисовка списка смежности

void PrintSmegElem(SmegnList\* groups) {

if (groups->Colo == 0) { //При размере = 0, у нас нет данных

cout << "Список пуст" << endl;

return;

}

SmegnElem\* current = groups->First;

while (current != NULL) { //Перебор структуры до последнего элемента(с выводом)

int value = current->num;

cout << value << " ";

current = current->Nextelem;

}

cout << endl;

}

//Функции для матричного представления графа

void Otogdestvlenie() {

if (n == 0) { return; } //Нельзя отождествить матрицу 1 на 1

Matrix3 = (int\*\*)calloc(n, 3); //Выделяем динамический массив

pogr = 0;

for (int j = 0; j < n; j++) { Matrix3[j] = (int\*)calloc(n, 3); } //Заполняем динамический массив динамическими массивами

for (int i = 0; i < n; i++) { for (int m = 0; m < n; m++) { Matrix3[i][m] = 2; Matrix3[m][m] = 0; } } // Заполняем матрицу 0 и 2

cout << "Введите номера отождествляемых вершин: \n";

cout << "Первая: ";

cin >> usrentr;

cout << "Вторая: ";

cin >> usrentr2;

usrentr--;

usrentr2--;

n++; //Возвращаем n до нормального(большего на 1) значения матрицы

for (int j = 0; j < n; j++) { Matrix1[usrentr][j] = Matrix1[usrentr2][j]; Matrix1[j][usrentr] = Matrix1[usrentr2][j]; } //Цикл для переноски значений в нужную вершину

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

for (int m = 0 + pogr; m < n; m++) {

if (i == m || i == usrentr2 || m == usrentr2) {}

else if (m > usrentr2 && i < usrentr2) { Matrix3[i][m - 1] = Matrix1[i][m]; Matrix3[m - 1][i] = Matrix1[m][i]; }

else if (m > usrentr2 && i > usrentr2) { Matrix3[i - 1][m - 1] = Matrix1[i][m]; Matrix3[m - 1][i - 1] = Matrix1[i][m]; }

else { Matrix3[i][m] = Matrix1[i][m]; Matrix3[m][i] = Matrix1[m][i]; }

}

pogr++;

}

cout << endl;

n--; //Уменьшаем n для взаимодействия со старой матрицей(на n-1 и n-1)

Matrix1 = (int\*\*)calloc(n, 3);

for (int j = 0; j < n; j++) { Matrix1[j] = (int\*)calloc(n, 3); }

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

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

Matrix1[i][m] = Matrix3[i][m];

Matrix1[m][m] = 0;

cout << Matrix1[i][m] << " ";

}

cout << endl;

}

}

void Stuagivanie() {

if (n == 0) { return; } //Нельзя отождествить матрицу 1 на 1

Matrix3 = (int\*\*)calloc(n, 3);

pogr = 0;

for (int j = 0; j < n; j++) { Matrix3[j] = (int\*)calloc(n, 3); }

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

for (int m = 0; m < n; m++) { Matrix3[i][m] = 2; Matrix3[m][m] = 0; }

cout << endl;

}

cout << "Введите номера стягиваемых вершин: \n";

cout << "Первая: ";

cin >> usrentr;

cout << "Вторая: ";

cin >> usrentr2;

usrentr--;

usrentr2--;

n++; //Возвращаем n до нормального(большего на 1) значения матрицы

for (int j = 0; j < n; j++) { Matrix1[usrentr][j] = Matrix1[usrentr2][j]; Matrix1[j][usrentr] = Matrix1[usrentr2][j]; } //Цикл для переноски значений в нужную вершину

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

for (int m = 0 + pogr; m < n; m++) {

if (i == m || i == usrentr2 || m == usrentr2) {}

else if (m > usrentr2 && i < usrentr2) { Matrix3[i][m - 1] = Matrix1[i][m]; Matrix3[m - 1][i] = Matrix1[m][i]; }

else if (m > usrentr2 && i > usrentr2) { Matrix3[i - 1][m - 1] = Matrix1[i][m]; Matrix3[m - 1][i - 1] = Matrix1[i][m]; }

else { Matrix3[i][m] = Matrix1[i][m]; Matrix3[m][i] = Matrix1[m][i]; }

}

pogr++;

}

cout << endl;

n--; //Уменьшаем n для взаимодействия со старой матрицей(на n-1 и n-1)

Matrix1 = (int\*\*)calloc(n, 3);

for (int j = 0; j < n; j++) { Matrix1[j] = (int\*)calloc(n, 3); }

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

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

Matrix1[i][m] = Matrix3[i][m];

Matrix1[m][m] = 0;

cout << Matrix1[i][m] << " ";

}

cout << endl;

}

}

void Raschep() {

Matrix3 = (int\*\*)calloc(n, 3);

for (int j = 0; j < n; j++) { Matrix3[j] = (int\*)calloc(n, 3); }

//Расширяем матрицу на одну строку и столбец

cout << "Введите номер вершины, которую хотите расщепить: ";

cin >> usrentr;

usrentr--;

n--;//Уменьшаем n для взаимодействия со старой матрицей(на n-1 и n-1)

if (usrentr > n || usrentr < 0) { cout << "\nНекорректный ввод. Повторите попытку." << endl; return; }

//Ограничиваем ввод пользователя

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

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

if (i == usrentr) { Matrix3[n][m] = Matrix1[i][m]; Matrix3[m][n] = Matrix3[n][m]; }

else { Matrix3[i][m] = Matrix1[i][m]; Matrix3[m][i] = Matrix1[m][i]; }

}

}

Matrix3[usrentr][n] = 1;

Matrix3[n][usrentr] = 1;

//У расщепляемых вершин должно быть соединяющее ребро

n++; //Возвращаем n до нормального(большего на 1) значения матрицы

Matrix1 = (int\*\*)calloc(n, 3);

for (int j = 0; j < n; j++) { Matrix1[j] = (int\*)calloc(n, 3); }

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

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

Matrix1[i][m] = Matrix3[i][m];

Matrix1[m][m] = 0;

cout << Matrix1[i][m] << " ";

}

cout << endl;

}

cout << "\nВершина " << usrentr + 1 << " была расщеплена в вершину " << n << endl;

}

//Функции для представления графа списками смежности

void OtogdestvlenieSmegn() {

SpecMatrixRezerve = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SmegnElem\* Actual = SpecMatrix[usrentr2]->First->Nextelem;

SmegnElem\* Fresh = SpecMatrix[usrentr]->First;

bool Check = false;

int Dopo = 0;

usrentr2++;

while (Actual != NULL) {

Check = false;

Fresh = SpecMatrix[usrentr]->First;

while (Fresh != NULL) {

if (Fresh->num == Actual->num) { Check = true; }

Fresh = Fresh->Nextelem;

}

if (Check != true) {

AddSmegElem(SpecMatrix[usrentr], Actual->num);

AddSmegElem(SpecMatrix[Actual->num - 1], usrentr + 1);

}

Actual = Actual->Nextelem;

}

\_getch();

cout << "\nEtap 1" << endl;

for (int i = 0; i < n + 1; i++) { PrintSmegElem(SpecMatrix[i]); }

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

Actual = SpecMatrix[i]->First->Nextelem;

Fresh = SpecMatrix[i]->First;

while (Actual != NULL) {

if (Actual->Nextelem == NULL && Actual->num == usrentr2) { Fresh->Nextelem = NULL; }

if (Actual->num == usrentr2 && Actual->Nextelem != NULL) { Fresh->Nextelem = Actual->Nextelem; }

Fresh = Actual;

Actual = Actual->Nextelem;

}

}

\_getch();

cout << "\nEtap 2: " << usrentr2 << endl;

for (int i = 0; i < n + 1; i++) { PrintSmegElem(SpecMatrix[i]); }

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

if (SpecMatrix[Dopo]->First->num == usrentr2) { Dopo++; }

SpecMatrixRezerve[i] = SpecMatrix[Dopo]; Dopo++;

}

\_getch();

cout << "\nEtap 3" << endl;

for (int i = 0; i < n + 1; i++) { PrintSmegElem(SpecMatrix[i]); }

cout << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrixRezerve[i]); }

SpecMatrix = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SpecMatrix = SpecMatrixRezerve;

cout << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix[i]); }

}

void StuagivanieSmegn() {

SpecMatrixRezerve = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SmegnElem\* Actual = SpecMatrix[usrentr2]->First->Nextelem;

SmegnElem\* Fresh = SpecMatrix[usrentr]->First;

bool Check = false;

int Dopo = 0;

usrentr2++;

while (Actual != NULL) {

Check = false;

Fresh = SpecMatrix[usrentr]->First;

while (Fresh != NULL) {

if (Fresh->num == Actual->num) { Check = true; }

Fresh = Fresh->Nextelem;

}

if (Check != true) {

AddSmegElem(SpecMatrix[usrentr], Actual->num);

AddSmegElem(SpecMatrix[Actual->num - 1], usrentr + 1);

}

Actual = Actual->Nextelem;

}

\_getch();

cout << "\nEtap 1" << endl;

for (int i = 0; i < n + 1; i++) { PrintSmegElem(SpecMatrix[i]); }

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

Actual = SpecMatrix[i]->First->Nextelem;

Fresh = SpecMatrix[i]->First;

while (Actual != NULL) {

if (Actual->Nextelem == NULL && Actual->num == usrentr2) { Fresh->Nextelem = NULL; }

if (Actual->num == usrentr2 && Actual->Nextelem != NULL) { Fresh->Nextelem = Actual->Nextelem; }

Fresh = Actual;

Actual = Actual->Nextelem;

}

}

\_getch();

cout << "\nEtap 2: " << usrentr2 << endl;

for (int i = 0; i < n + 1; i++) { PrintSmegElem(SpecMatrix[i]); }

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

if (SpecMatrix[Dopo]->First->num == usrentr2) { Dopo++; }

SpecMatrixRezerve[i] = SpecMatrix[Dopo]; Dopo++;

}

\_getch();

cout << "\nEtap 3" << endl;

for (int i = 0; i < n + 1; i++) { PrintSmegElem(SpecMatrix[i]); }

cout << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrixRezerve[i]); }

SpecMatrix = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SpecMatrix = SpecMatrixRezerve;

cout << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix[i]); }

}

void RaschepSmegn() {

SpecMatrixRezerve = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

int Save = 0;

/\*cout << "Введите номер вершины, которую хотите расщепить: "; //Разблокировать при одиночном использовании(без матричной функции)

cin >> usrentr;

usrentr--;\*/

SmegnElem\* Actual = SpecMatrix[usrentr]->First;

SpecMatrixRezerve = SpecMatrix;

SpecMatrixRezerve[n - 1] = CreateList();

AddSmegElem(SpecMatrixRezerve[n - 1], n);

while (Actual != NULL) {

if (Actual->num != n) {

AddSmegElem(SpecMatrixRezerve[n - 1], Actual->num);

AddSmegElem(SpecMatrixRezerve[Actual->num - 1], n);

}

Actual = Actual->Nextelem;

}

SpecMatrix = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SpecMatrix = SpecMatrixRezerve;

cout << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix[i]); }

}

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(0));

//Задание 1//

cout << "Введите размерность матриц: ";

cin >> n;

if (n <= 0) { cout << "\nНекорректный ввод. Повторите попытку" << endl; return 0; }

Matrix1 = (int\*\*)calloc(n, 3);

SpecMatrix = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

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

Matrix1[j] = (int\*)calloc(n, 3);

SpecMatrix[j] = CreateList(); //Создаём список смежности

AddSmegElem(SpecMatrix[j], j + 1); //Вставляем в начало каждого списка смежности номер отвечающей вершины

}

cout.precision(3 \* n);

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

for (int m = 0 + pogr; m < n; m++) {

Matrix1[i][m] = 0 + rand() % 2;

Matrix1[m][i] = Matrix1[i][m];

Matrix1[i][i] = 0;

}

pogr++;

}

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

for (int m = 0; m < n; m++) { if (Matrix1[i][m] == 1) { AddSmegElem(SpecMatrix[i], m + 1); } }

}

//Задание 3//

while (choose != 4) {

system("cls");

cout << "\nМатрица №1:" << endl;

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

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

cout << Matrix1[i][m] << " ";

}

cout << endl;

}

cout << "\nСписки смежности для Мартицы №1:" << endl;

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix[i]); }

cout << "\n1.Отождествить вершины" << endl;

cout << "2.Стянуть рёбра" << endl;

cout << "3.Расщепление вершины" << endl;

cout << "4.Выход" << endl;

cout << "\nВыберите операцию для выполнения: ";

cin >> choose;

switch (choose) {

case 1://Отождествление вершин

n--;

Otogdestvlenie();

OtogdestvlenieSmegn();

\_getch();

break;

case 2://Стягивание ребра

n--;

Stuagivanie();

StuagivanieSmegn();

\_getch();

break;

case 3://Расщепление вершины

n++;

Raschep();

RaschepSmegn();

\_getch();

break;

case 4://Выход из цикла

cout << "\nВыход из программы." << endl;

break;

default:

cout << "\nНекорректный ввод. Повторите попытку." << endl;

break;

}

}

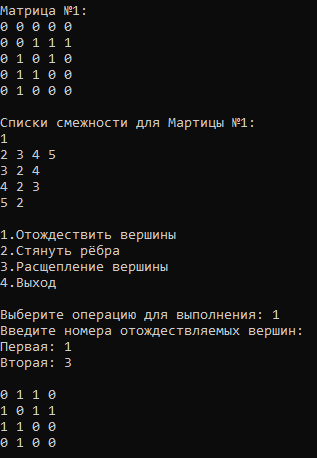
return 0;

}

Пояснительный текст к программе:

Данная программа для матричной формы графов выполняет операцию отождествления вершин (Рис.№6), стягивания ребра (Рис.№7), расщепления вершины (Рис.№8).

Результаты работы программы:



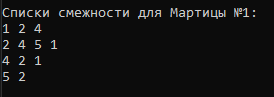
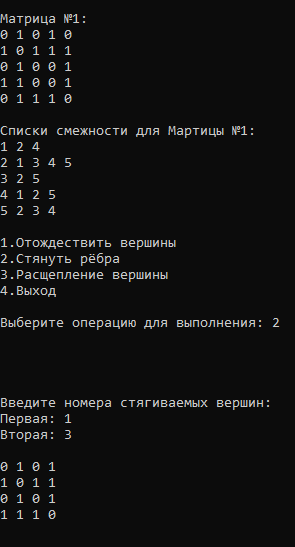


Рисунок №6 – Работа программы из задания №2 (отождествление вершин).



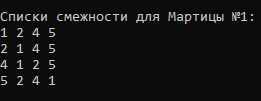


Рисунок №7 – Работа программы из задания №2 (стягивание ребра).

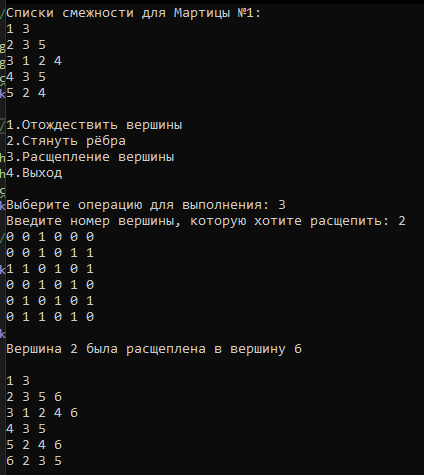


Рисунок №8 – Работа программы из задания №2 (расщепления вершины).

Вывод: При выполнении данной лабораторной работы мы смогли научиться осуществлять унарные и бинарные операции над графами для их матричной формы представления.