Министерство 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

студент группы 21ВВ3:

Савицкий Макар

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Цель работы**

Реализация матрицы смежности и инцидентности графов и определение характеристик графов по реализованным матрицам.

**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 https://lh3.googleusercontent.com/-h7O4PeQbI-WC994Z_FFfdDxHItHvnVTcMgcgCS6v0r6PT4uvnbsgjzToRpEEU4hPjppuqW76ik9l37ezXDAgzG3zXAlXtJThaCoJKMU1IaTXB_N5nfaij0hJHpqI09t8cmRfLSy8KH9BYy7NwkRHLJzJxEBj7WHrMSqI78NTqTJG-EAxizqooddM38bg-J4lHdMRW9Sig *G*2

б) пересечения  *G* = *G*1 https://lh6.googleusercontent.com/RP1NKSPonKNLeG7U-t_kmtxf22vHhcXoorbryxGK93F-FtGUd2SRoKbPEEO5M7M_bfSglExsVtB1eUKSl9NJmEfVi3XUWmXrXqve2YN3aWps2y4nraXryeG-OQnA9mRdWV2oMp8WB3ZfIB-k5JaaUh3AWAOYW_u98Jv7GB-0VovPXJ7_TbuM6KiGBl3RIIXjqBuuEHjsxA *G*2

в) кольцевой суммы *G* = *G*1 https://lh3.googleusercontent.com/aJnHmjiOAL08aXy8flIgMz30h6ij7y3ptrAjUQT0stLGGgi2hR4P_5jbW61RlSWRxenC_bKeT-jB9_dy5jvRv5L4qetx4k6GkG_WR7FAV6bk9CvEuz9H3Um1grrCwxArirUxlsI292TTgXUmnigYtHFQW-eUbT9n7PqYzd-7SSBjeYNJCykPbCQuFFi4Y-JDm617SI4jig *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.

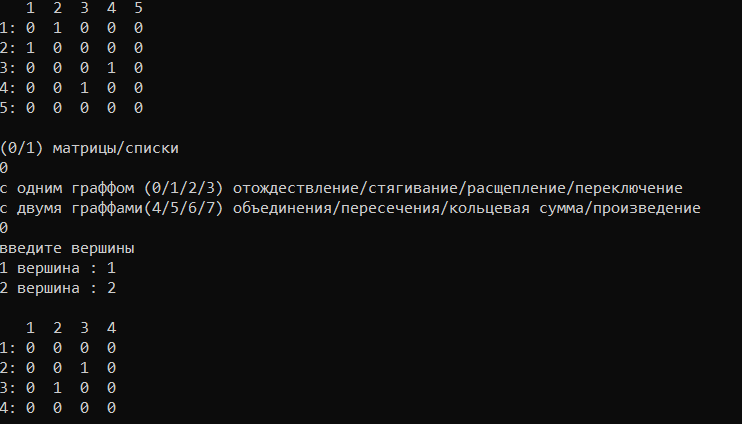
**Описание метода решения задачи**

**Задание 1**

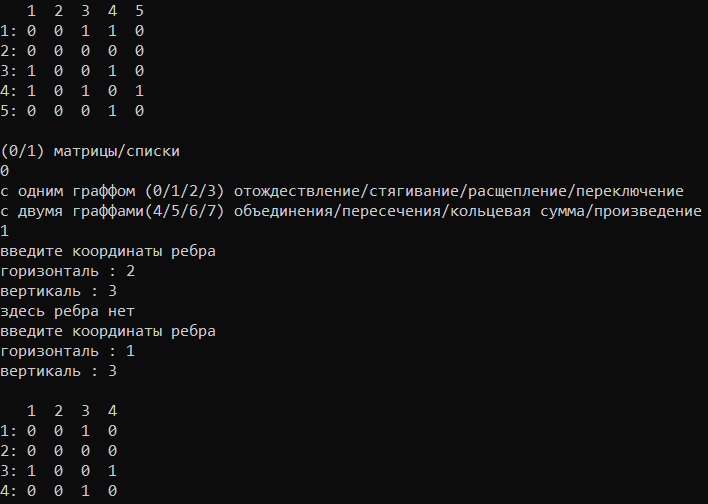
1. *Генерируем 2 графа с помощью матрицы смежности определенного размера и выводим его на экран*
2. *Составляем списки смежности на сгенерированные матрицы смежности*
3. *Выбираем с каким режимом будем работать (матрицы или списки)*
4. *В зависимости от введённого значения выполняем ту или иную команду*
5. *С помощью команды переключение, можно менять матрицу, с которой идет работа*
6. *Выводим результат на экран*
7. *Переходим к пункту 4*

**Результаты работ программы**

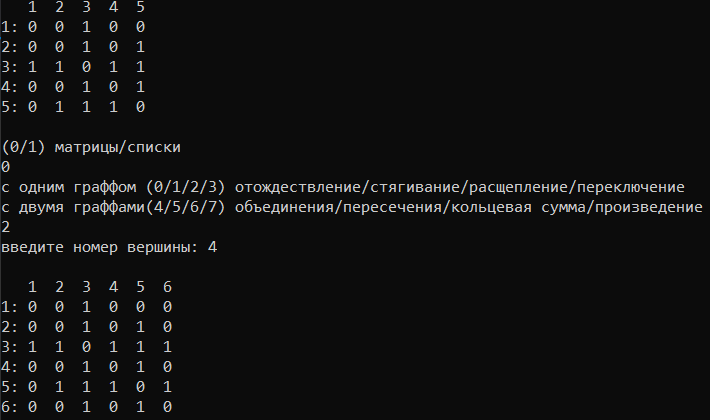
**Отождествление**

****

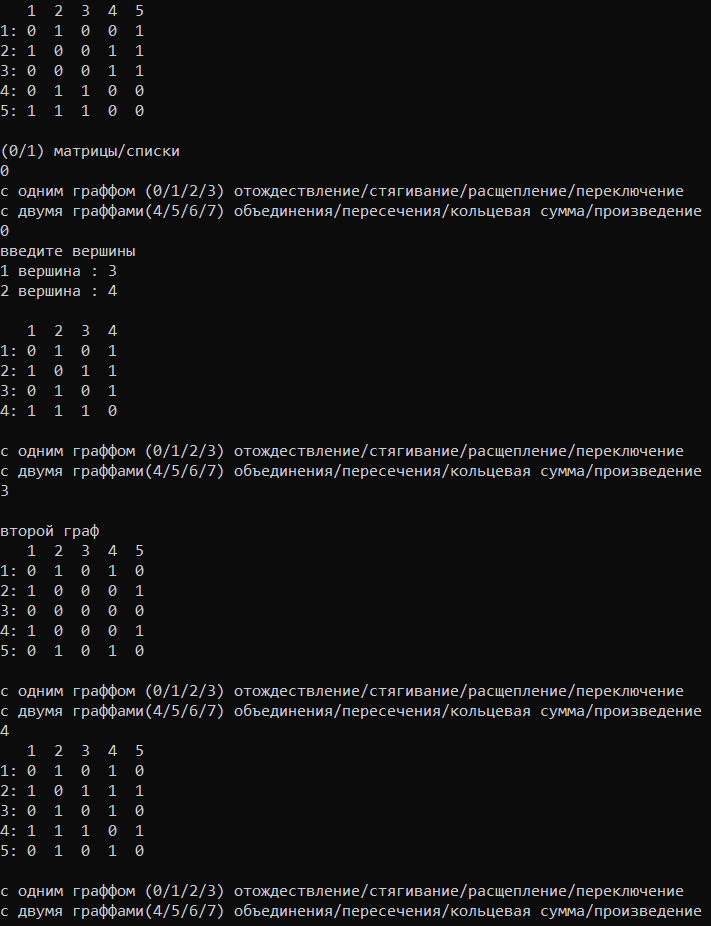
**Стягивание**

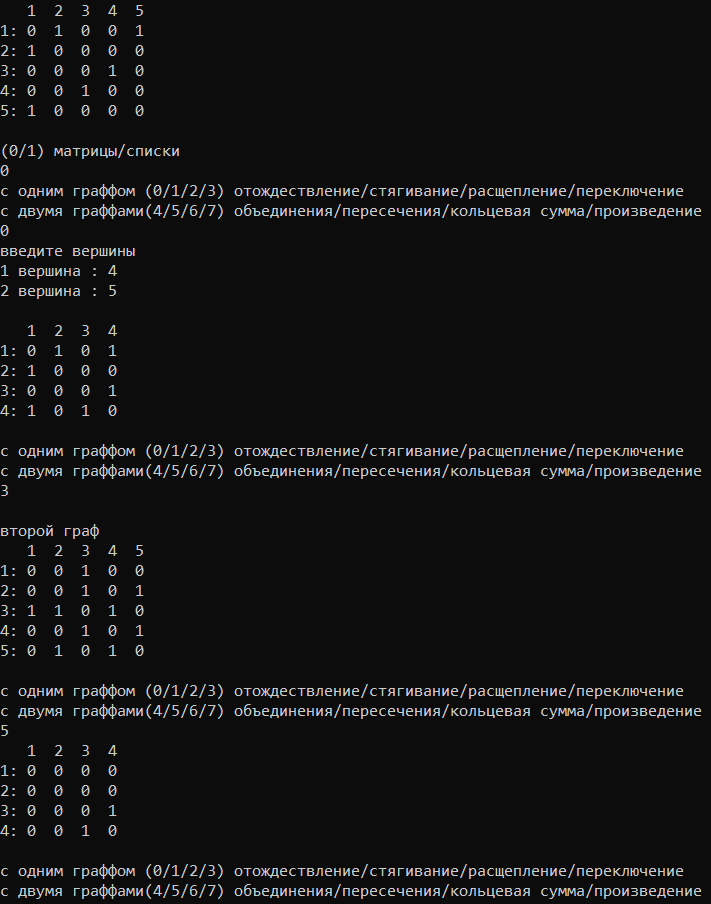
****

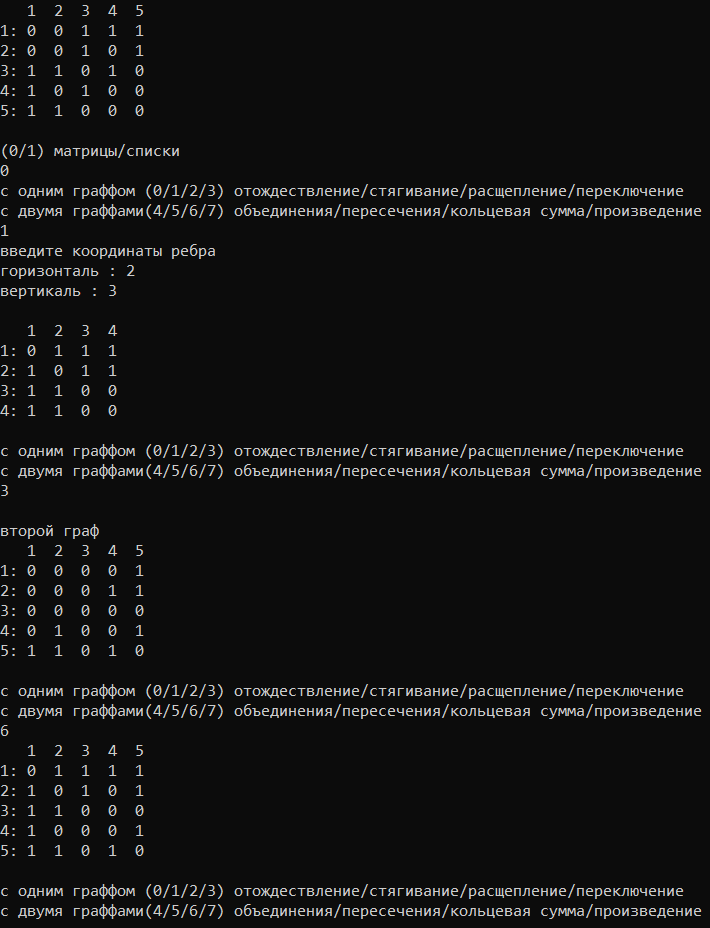
**Расщепление**

****

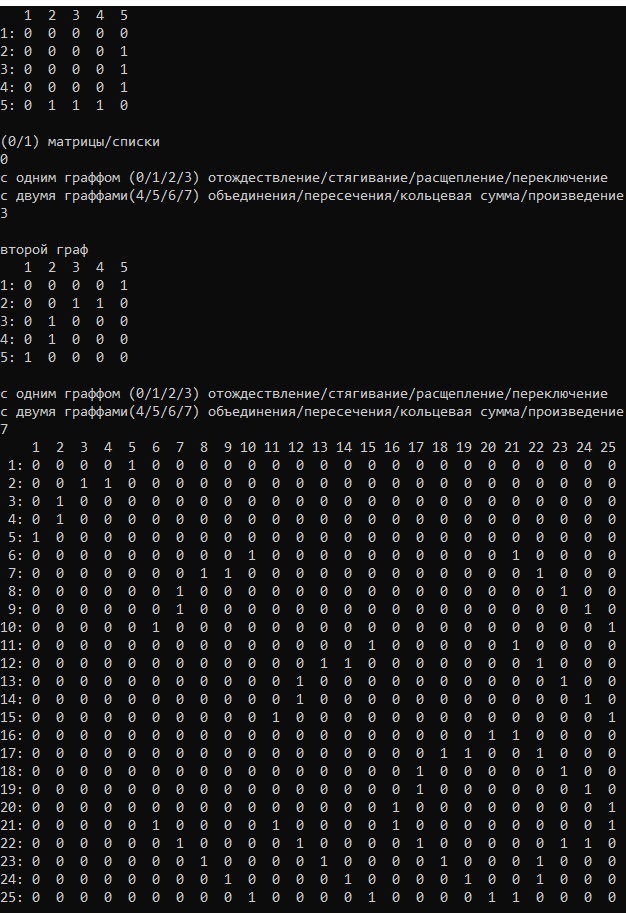
**Объединение**

****

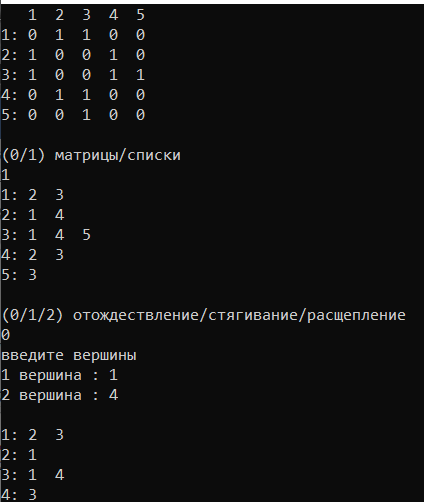
**Пересечение **

**Кольцевая сумма**

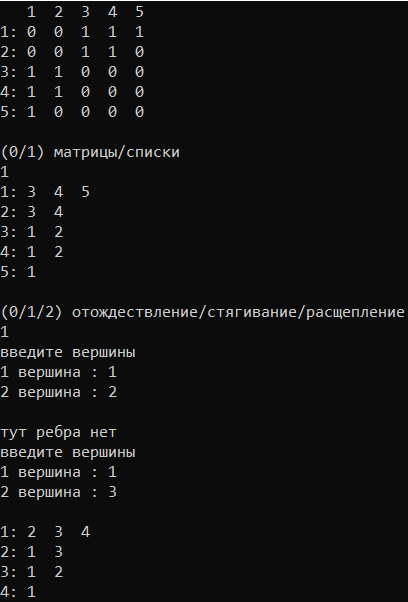
**Произведение**

****

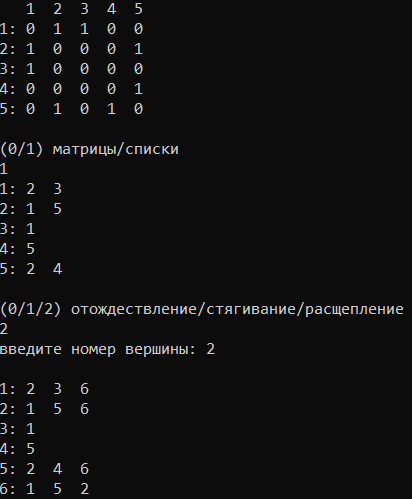
**Отождествление в списках смежности**

****

**Стягивание в списках смежности**

****

**Расщепление в списках смежности**

****

**Листинг**#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h> //указание прототипов

#include <stdlib.h> //библиотечных функций

#include <windows.h>

#include <malloc.h>

#include <string.h>

#include <time.h>

struct node{

int i;

struct node\* next; // ссылка на следующий элемент

};

struct uz{

int t;

struct node\* head;

struct node\* last;

struct uz\* next; // ссылка на следующий элемент

};

struct uz\* head1 = NULL, \*last1 = NULL;

void spstore\_uz(int t) {

struct uz\* p = NULL;

if ((p = (uz\*)malloc(sizeof(struct uz))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->t = t;

p->head = NULL;

p->last = NULL;

p->next = NULL;

if (p != NULL) {

if (head1 == NULL) {

head1 = p;

last1 = p;

}

else{

last1->next = p;

last1 = p;

}

}

return;

}

void del\_all\_node(struct node\* head) {

struct node\* p = head;

while (head != NULL) {

head = head->next;

free(p);

p = head;

}

}

void del\_uz(int t){

struct uz\* struc = head1; // указатель, проходящий по списку установлен на начало списка

struct uz\* prev = NULL;// указатель на предшествующий удаляемому элемент

struct node\* p = NULL;

if (head1 == NULL){ // если голова списка равна NULL, то список пуст

return;

}

if (t == struc->t){ // если удаляемый элемент - первый

head1 = struc->next; // установливаем голову на следующий элемент

if(struc->head) del\_all\_node(struc->head); // удаляем подэлементы

free(struc); // удаляем первый элемент

struc = head1; // устанавливаем указатель для продолжения поиска

}else{

prev = struc;

struc = struc->next;

}

while (struc) { // проход по списку и поиск удаляемого элемента

if (t == struc->t) { // если нашли, то

if (struc->next) { // если найденный элемент не последний в списке

prev->next = struc->next; // меняем указатели

if (struc->head) del\_all\_node(struc->head); // удаляем подэлементы

free(struc); // удаляем элемент

struc = prev->next; // устанавливаем указатель для продолжения поиска

}

else { // если найденный элемент последний в списке

prev->next = NULL; // обнуляем указатель предшествующего элемента

if (struc->head) del\_all\_node(struc->head); // удаляем подэлементы

free(struc); // удаляем элемент

break;

}

}

else { // если не нашли, то

prev = struc; // устанавливаем указатели для продолжения поиска

struc = struc->next;

}

}

struc = head1;

while (struc) { // проход по списку и поиск уменьшаем значение вершин

struct node\* p = struc->head;

while (p != NULL) {

if (p->i > t + 1)

p->i -= 1;

p = p->next;

}

if (struc->t > t) {

struc->t--;

}

struc = struc->next;

}

}

struct node\* get\_struct(int i) {

struct node\* p = NULL;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL) // выделяем память под новый элемент списка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

p->i = i;

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

void spstore\_node(int t, int i) {

struct uz\* struc1 = head1; // указатель, проходящий по списку установлен на начало списка

struct uz\* prev1 = NULL;

if (head1 == NULL) {

return;

}

while (struc1) {

if (t == struc1->t) {

break;

}

struc1 = struc1->next;

}

struct node\* p = NULL, \* f = NULL;

p = get\_struct(i);

if (struc1->head == NULL && p != NULL) { // если списка нет, то устанавливаем голову списка

struc1->head = p;

struc1->last = p;

}

else if (struc1->head != NULL && p != NULL) {

f = struc1->head;

int ii = 0;

if (f->i > p->i) { //если в начало ставим

p->next = struc1->head;

struc1->head = p;

ii = 1;

}

if (struc1->last->i < p->i) { //если в конец ставим

struc1->last->next = p;

struc1->last = p;

ii = 1;

}

if (struc1->last->i > p->i && f->i < p->i) {

while (f->next->i < p->i) {

f = f->next;

}

p->next = f->next;

f->next = p;

ii = 1;

}

if (ii == 0) {

free(p);

}

}

return;

}

void del\_node(int t, int i){

struct uz\* struc1 = head1; // указатель, проходящий по списку установлен на начало списка

struct uz\* prev1 = NULL;

if (head1 == NULL) {

return;

}

while (struc1) {

if (t == struc1->t) {

break;

}

struc1 = struc1->next;

}

struct node\* struc = struc1->head; // указатель, проходящий по списку установлен на начало списка

struct node\* prev = NULL;// указатель на предшествующий удаляемому элемент

if (struc1->head == NULL){ // если голова списка равна NULL, то список пуст

return;

}

if (i == struc->i){ // если удаляемый элемент - первый

struc1->head = struc->next; // установливаем голову на следующий элемент

free(struc); // удаляем первый элемент

struc = struc1->head; // устанавливаем указатель для продолжения поиска

}

else{

prev = struc;

struc = struc->next;

}

while (struc){ // проход по списку и поиск удаляемого элемента

if (i == struc->i){ // если нашли, то

if (struc->next){ // если найденный элемент не последний в списке

prev->next = struc->next; // меняем указатели

free(struc); // удаляем элемент

struc = prev->next; // устанавливаем указатель для продолжения поиска

}

else{ // если найденный элемент последний в списке

prev->next = NULL; // обнуляем указатель предшествующего элемента

free(struc); // удаляем элемент

return;

}

}

else{ // если не нашли, то

prev = struc; // устанавливаем указатели для продолжения поиска

struc = struc->next;

}

}

}

int find(int t, int i){

struct uz\* struc1 = head1; // указатель, проходящий по списку установлен на начало списка

struct uz\* prev1 = NULL;

if (head1 == NULL) {

return 0;

}

while (struc1) {

if (t == struc1->t) {

break;

}

struc1 = struc1->next;

}

struct node\* struc = struc1->head;

if (struc1->head == NULL){

return 0;

}

while (struc){

if (i == struc->i){

return 1;

}

struc = struc->next;

}

return 0;

}

void review(struct node\* struc) {

if (struc != NULL) {

while (struc){

printf("%-2d", struc->i);

struc = struc->next;

}

}

return;

}

void review\_uz(void) {

struct uz\* struc = head1;

while (struc){

printf("%d: ", struc->t+1);

review(struc->head);

printf("\n");

struc = struc->next;

}

return;

}

void print(int \*d, int k) {

printf(" ");

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

printf("%3d", l + 1);

}

printf("\n");

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

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

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

printf("%-3d", \*(d + i \* k + l));

}

printf("\n");

}

printf("\n");

}

int \*contraction(int \*d, int k, int x, int y) {

if (x > y) {

x = x + y;

y = y - x;

y = -y;

x = x - y;

}

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

if (l != x - 1) \*(d + (x - 1) \* k + l) = \*(d + (y - 1) \* k + l) || \*(d + (x - 1) \* k + l);

if (l != x - 1) \*(d + l \* k + (x - 1)) = \*(d + l \* k + (x - 1)) || \*(d + l \* k + (y - 1));

\*(d + (y - 1) \* k + l) = 0;

\*(d + l \* k + (y - 1)) = 0;

}

for (int i = y; i < k; i++) {

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

\*(d + (i - 1) \* k + l) = \*(d + i \* k + l);

}

}

for (int i = y; i < k; i++) {

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

\*(d + l \* k + (i - 1)) = \*(d + l \* k + i);

}

}

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

\*(d + l \* k + (k - 1)) = 0;

\*(d + (k - 1) \* k + l) = 0;

}

int t = 0;

k -= 1;

int\* dd = (int\*)malloc(k \* k \* sizeof(int));

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

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

\*(dd + i \* k + l) = \*(d + i \* k + l + t);

}

t += 1;

}

free(d);

return dd;

}

int \*splitting(int\* d, int dd, int x) {

x -= 1;

int\* ddd = (int\*)malloc((dd + 1) \* (dd + 1) \* sizeof(int));

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

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

\*(ddd + i \* (dd + 1) + l) = \*(d + i \* dd + l);

}

\*(ddd + i \* (dd + 1) + dd) = \*(d + i \* dd + x);

\*(ddd + dd \* (dd + 1) + i) = \*(d + x \* dd + i);

}

\*(ddd + (dd + 1) \* (dd + 1) - 1) = 0;

free(d);

return ddd;

}

void main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int k = 5;

int t1, t2, x, y, dd1 = k, dd2 = k, dd;

int\* d, \*mm, \* dddd, D = 1;

int\* d1 = (int\*)malloc(k \* k \* sizeof(int));

int\* d2 = (int\*)malloc(k \* k \* sizeof(int));

d = d1;

for (int i = 0; i < 2; i++) {

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

for (int l = i; l < k; l++) {

\*(d + i \* k + l) = (rand() % 5 + 7) / 10;

\*(d + l \* k + i) = \*(d + i \* k + l);

if (l == i) \*(d + i \* k + l) = 0;

}

}

d = d2;

}

print(d1, dd1);

D = 1;

d = d1;

dd = dd1;

printf("(0/1) матрицы/списки\n");

scanf("%d", &t1);

if (t1 == 0) {

while (1) {

printf("с одним граффом (0/1/2/3) отождествление/стягивание/расщепление/переключение\n");

printf("с двумя граффами(4/5/6/7) объединения/пересечения/кольцевая сумма/произведение\n");

scanf("%d", &t1);

switch (t1) {

case 0:

do {

printf("введите вершины\n1 вершина : ");

scanf("%d", &x);

printf("2 вершина : ");

scanf("%d", &y);

if (x == y) printf("введите разные вершины\n");

if (x < 0 || y < 0 || x > dd || y > dd) printf("\nнеподходящее значение\n");

if(dd == 1) printf("\nуже мало вершин\n");

} while (x == y || x > dd || y > dd || dd == 1);

d = contraction(d, dd, x, y);

dd -= 1;

printf("\n");

print(d, dd);

break;

case 1:

do {

printf("введите координаты ребра\nгоризонталь : ");

scanf("%d", &x);

printf("вертикаль : ");

scanf("%d", &y);

if (\*(d + (x - 1) \* dd + (y - 1)) == 0) printf("здесь ребра нет\n");

if (x < 0 || y < 0 || x > dd || y > dd) printf("\nнеподходящее значение\n");

if (dd == 1) printf("\nуже мало вершин\n");

} while (\*(d + (x - 1) \* dd + (y - 1)) == 0 || x > dd || y > dd || dd == 1);

d = contraction(d, dd, x, y);

dd -= 1;

printf("\n");

print(d, dd);

break;

case 2:

do {

printf("введите номер вершины: ");

scanf("%d", &x);

if (x > dd) printf("\nслишком большое значение\n");

if (x < 0) printf("\nслишком маленькое значение\n");

} while (x > dd || x < 0);

d = splitting(d, dd, x);

dd += 1;

printf("\n");

print(d, dd);

break;

case 3:

printf("\n");

if (D == 1) {

printf("второй граф\n");

D = 2;

dd1 = dd;

d1 = d;

d = d2;

dd = dd2;

}

else {

printf("первый граф\n");

D = 1;

dd2 = dd;

d2 = d;

d = d1;

dd = dd1;

}

print(d, dd);

break;

case 4:

if (D == 1) {

dd1 = dd;

d1 = d;

}

else {

dd2 = dd;

d2 = d;

}

if (dd2 > dd1) {

for (int i = 0; i < dd1; i++) {

for (int l = 0; l < dd1; l++) {

\*(d2 + i \* (dd1 + dd2 - dd1) + l) = \*(d1 + i \* dd1 + l) || \*(d2 + i \* (dd1 + dd2 - dd1) + l);

}

}

D = 0;

d = d2;

dd = dd2;

}

else {

for (int i = 0; i < dd2; i++) {

for (int l = 0; l < dd2; l++) {

\*(d1 + i \* (dd2 + dd1 - dd2) + l) = \*(d1 + i \* (dd2 + dd1 - dd2) + l) || \*(d2 + i \* dd2 + l);

}

}

D = 1;

d = d1;

dd = dd1;

}

print(d, dd);

break;

case 5:

if (D == 1) {

dd1 = dd;

d1 = d;

}

else {

dd2 = dd;

d2 = d;

}

if (dd2 > dd1) {

for (int i = 0; i < dd1; i++) {

for (int l = 0; l < dd1; l++) {

\*(d1 + i \* dd1 + l) = \*(d1 + i \* dd1 + l) && \*(d2 + i \* (dd1 + dd2 - dd1) + l);

}

}

D = 1;

d = d1;

dd = dd1;

}

else {

for (int i = 0; i < dd2; i++) {

for (int l = 0; l < dd2; l++) {

\*(d2 + i \* dd2 + l) = \*(d1 + i \* (dd2 + dd1 - dd2) + l) && \*(d2 + i \* dd2 + l);

}

}

D = 0;

d = d2;

dd = dd2;

}

print(d, dd);

break;

case 6:

if (D == 1) {

dd1 = dd;

d1 = d;

}

else {

dd2 = dd;

d2 = d;

}

if (dd2 > dd1) {

for (int i = 0; i < dd1; i++) {

for (int l = 0; l < dd1; l++) {

\*(d2 + i \* (dd1 + dd2 - dd1) + l) = \*(d1 + i \* dd1 + l) ^ \*(d2 + i \* (dd1 + dd2 - dd1) + l);

}

}

D = 0;

d = d2;

dd = dd2;

}

else {

for (int i = 0; i < dd2; i++) {

for (int l = 0; l < dd2; l++) {

\*(d1 + i \* (dd2 + dd1 - dd2) + l) = \*(d1 + i \* (dd2 + dd1 - dd2) + l) ^ \*(d2 + i \* dd2 + l);

}

}

D = 1;

d = d1;

dd = dd1;

}

print(d, dd);

break;

case 7:

if (D == 1) {

dd1 = dd;

d1 = d;

}

else {

dd2 = dd;

d2 = d;

}

dddd = (int\*)malloc(dd1 \* dd2 \* dd1 \* dd2 \* sizeof(int));

for (int i = 0; i < dd1 \* dd2; i++) {

for (int l = 0; l < dd1 \* dd2; l++) {

\*(dddd + i \* dd1 \* dd2 + l) = 0;

}

}

for (int i = 0; i < dd1; i++) {

for (int ii = 0; ii < dd1; ii++) {

if (i == ii) {

for (int l = 0; l < dd2; l++) {

for (int ll = 0; ll < dd2; ll++) {

if (i == ii) \*(dddd + i \* dd1 \* dd2 \* dd2 + ii \* dd2 + l \* dd1 \* dd2 + ll) = \*(d2 + l \* dd2 + ll);

}

}

}

if (\*(d1 + i \* dd1 + ii) == 1) {

for (int l = 0; l < dd2; l++) {

for (int ll = 0; ll < dd2; ll++) {

if (l == ll) \*(dddd + i \* dd1 \* dd2 \* dd2 + ii \* dd2 + l \* dd1 \* dd2 + ll) = 1;

}

}

}

}

}

D = 1;

d = d1;

dd = dd1;

printf(" ");

for (int l = 0; l < dd1 \* dd2; l++) {

printf("%3d", l + 1);

}

printf("\n");

for (int i = 0; i < dd1 \* dd2; i++) {

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

for (int l = 0; l < dd1 \* dd2; l++) {

printf("%-3d", \*(dddd + i \* dd1 \* dd2 + l));

}

printf("\n");

}

printf("\n\n");

break;

}

}

}

else {

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

spstore\_uz(i);

for (int l = 0; l < k; l++)

if (\*(d + i \* k + l) == 1)

spstore\_node(i, l + 1);

}

review\_uz();

while (1) {

printf("(0/1/2) отождествление/стягивание/расщепление\n");

scanf("%d", &t1);

switch (t1) {

case 0:

do {

printf("введите вершины\n1 вершина : ");

scanf("%d", &x);

printf("2 вершина : ");

scanf("%d", &y);

if (x == y) printf("введите разные вершины\n\n");

if (x < 0 || y < 0 || x > dd || y > dd) printf("\nнеподходящее значение\n");

if (dd == 1) printf("\nуже мало вершин\n");

} while (x == y || x < 0 || y < 0 || x > dd || y > dd || dd == 1);

if (x > y) {

x = x + y;

y = y - x;

y = -y;

x = x - y;

}

for (int i = 0; i < dd1; i++){

if (find(y - 1, i + 1) && i != x - 1)

spstore\_node(x - 1, i + 1);

del\_node(i, y);

}

dd1 -= 1;

for (int i = 0; i < dd1; i++) {

if (find(x - 1, i + 1) )

spstore\_node(i, x);

}

del\_uz(y - 1);

review\_uz();

break;

case 1:

do {

printf("введите вершины\n1 вершина : ");

scanf("%d", &x);

printf("2 вершина : ");

scanf("%d", &y);

if (x == y) printf("введите разные вершины\n\n");

if (x < 0 || y < 0 || x > dd || y > dd) printf("\nнеподходящее значение\n");

if (dd == 1) printf("\nуже мало вершин\n");

t2 = 0;

if (t2 == 0) printf("\nтут ребра нет\n");

} while (x == y || x < 0 || y < 0 || x > dd || y > dd || dd == 1 || t2 == 0);

if (x > y) {

x = x + y;

y = y - x;

y = -y;

x = x - y;

}

for (int i = 0; i < dd1; i++) {

if (find(y - 1, i + 1) && i != x - 1)

spstore\_node(x - 1, i + 1);

del\_node(i, y);

}

dd1 -= 1;

for (int i = 0; i < dd1; i++) {

if (find(x - 1, i + 1))

spstore\_node(i, x);

}

del\_uz(y - 1);

review\_uz();

break;

case 2:

do {

printf("введите номер вершины: ");

scanf("%d", &x);

if (x > dd) printf("\nслишком большое значение\n");

if (x < 0) printf("\nслишком маленькое значение\n");

} while (x > dd || x < 0);

spstore\_uz(dd1);

for (int i = 0; i < dd1; i++) {

if (find(x - 1, i + 1)){

spstore\_node(dd1, i + 1);

spstore\_node(i, dd1 + 1);

}

}

dd1 += 1;

review\_uz();

break;

}

}

}

}

**Вывод**

В ходе выполнения данной лабораторной работы мы научились реализовывать матрицы смежности и инцидентности графов и определение характеристик графов по реализованным матрицам.