Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

# по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»

на тему: «Унарные и бинарные операции над графами»

Выполнили:

студенты группы 20ВВ3

Горожанин Я. А.

Тихонов А. А.

Скирдова В. М.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2021

**Название**

Унарные и бинарные операции над графами

**Цель работы**

Исследование унарных, бинарных операций над графами и приобретение практических навыков решения задач с использованием основ теории графов.

**Методические указания**

*Отождествление вершин.* В графе *G*1 выделяются вершины *и,v.* Определяют окружение *Q*1 вершины *u*,и окружение *Q*2 вершины *v,* вычисляют их объединение *Q* = *Q1* * Q2.* Затем над графом *G*1 выполняются следующие преобразования:

* из графа *G*1 удаляют вершины *u,* *v (H*1 *= G*1 *- u - v);*
* к графу *Н*1присоединяют новую вершину *z (H*1 *= H*1 *+z);*
* вершину *z* соединяют ребром с каждой из вершин *w*1*Q*

*(G*2 *= H*1 *+ zwi*, *i =* 1,2,3*,…).*

*Стягивание ребра.* Данная операция является операцией отождествления смежных вершин *и, v* в графе *G*1.

Наиболее важными бинарными операциями являются: объединение, пересечение, декартово произведение и кольцевая сумма.

*Объединение.* Граф *G* называется объединением или наложением графов *G*1 и *G*2, если *VG = V*1*V*2*; UG = U*1* U*2.

Объединение графов *G*1 и *G*2 называется дизъюнктным, если *V*1*V*2 *= *. При дизъюнктном объединении никакие два из объединяемых графов не должны иметь общих вершин.

*Пересечение.* Граф *G* называется пересечением графов *G*1, *G*2,если *VG = V*1*V*2и *UG = U*1*U*2. Операция "пересечения" записывается следующим образом: *G = G*1*G*2*.*

*Декартово произведение.* Граф *G* называется декартовым произведением графов *G*1 и *G*2 если *VG* = *V*1*V*2 —декартово произведение множеств вершин графов *G*1, *G*2, а множество ребер *U*c задается следующим образом: вершины (*zi*, *vk*) и (*zj*, *vl*) смежны в графе *G* тогда и только тогда, когда *zi* = *zj*(*i* = *j*), a *v*k и *vl* смежны в *G*2 или *vk* = *vl*(*k* = *l*), смежны в графе *G*1.

*Кольцевая сумма* графов представляет граф, который не имеет изолированных вершин и состоит из ребер, присутствующих либо в первом исходном графе, либо во втором. Кольцевая сумма определяется следующим соотношением: *G* = *G*1  *G*2.

**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.

**Листинг**

**Файл Lab3.cpp**

#include "Header.h"

int Menu;

int\*\* a1;

graf\* grafon;

int size;

void select() {

switch (Menu) {

case 0:

system("cls");

versionsMenu();

break;

case 1:

system("cls");

versionsMenu();

system("pause");

break;

case 2:

system("cls");

number3Menu();

system("pause");

break;

case 3:

system("cls");

number4();

system("pause");

break;

case 4:

exit(0);

}

}

void versionsMenu() {

char cur;

int isf = 0;

versionsMenuG(isf);

do {

cur = \_getch();

if (cur == 80) {

isf++;

switch (isf) {

case 2:

isf = 0;

versionsMenuG(isf);

break;

default:

versionsMenuG(isf);

break;

}

}

if (cur == 72) {

isf--;

switch (isf) {

case -1:

isf = 2;

versionsMenuG(isf);

break;

default:

versionsMenuG(isf);

break;

}

}

if (cur == 27) {

return;

}

} while (cur != 13);

switch (Menu) {

case 0:

switch (isf)

{

case 0:

system("cls");

number1\_1();

system("pause");

return;

case 1:

system("cls");

number1\_2();

system("pause");

return;

default:

break;

}

case 1:

if (isf == 0) {

int inside\_menu = versions2Menu();

switch (inside\_menu) {

case 0:

system("cls");

a1 = number2\_create();

number2\_a(a1);

break;

case 1:

system("cls");

a1 = number2\_create();

number2\_b(a1);

break;

case 2:

system("cls");

a1 = number2\_create();

number2\_v(a1);

break;

}

break;

}

else {

int inside\_menu = versions2Menu();

switch (inside\_menu) {

case 0:

system("cls");

grafon = num2\_list\_create();

number2\_a\_list(grafon);

break;

case 1:

system("cls");

grafon = num2\_list\_create();

number2\_b\_list(grafon);

break;

case 2:

system("cls");

grafon = num2\_list\_create();

number2\_v\_list(grafon);

break;

}

break;

}

}

}

void number3Menu() {

int inside\_menu = versions3Menu();

system("cls");

number3\_create(inside\_menu);

}

void main() {

setlocale(LC\_ALL, "Russian");

do {

create\_menu();

mainMenu();

select();

} while (1);

}

**Файл MenuG.cpp**

#include "Header.h"

menu \_menu;

void create\_menu() {

strcpy(\_menu.up, "\n"

" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

" \\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \n");

strcpy(\_menu.wall, " || || \n");

strcpy(\_menu.wallBottom, " ||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\n"

" ||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\n");

strcpy(\_menu.bottom, " ||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\n"

" /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\ \n");

\_menu.text[0] = " || Задание 1 ||\n";

\_menu.text[1] = " || Задание 2 ||\n";

\_menu.text[2] = " || Задание 3 ||\n";

\_menu.text[3] = " || Задание 4 ||\n";

\_menu.text[4] = " || Выход ||\n";

\_menu.active\_text[0] = " || > Задание 1 < ||\n";

\_menu.active\_text[1] = " || > Задание 2 < ||\n";

\_menu.active\_text[2] = " || > Задание 3 < ||\n";

\_menu.active\_text[3] = " || > Задание 4 < ||\n";

\_menu.active\_text[4] = " || > Выход < ||\n";

\_menu.versions[0] = " || Версия через матрицы ||\n";

\_menu.versions[1] = " || Версия через списки ||\n";

\_menu.active\_versions[0] = " || > Версия через матрицы < ||\n";

\_menu.active\_versions[1] = " || > Версия через списки < ||\n";

\_menu.versions2[0] = " || Отождествление вершин ||\n";

\_menu.versions2[1] = " || Стягивание ребра ||\n";

\_menu.versions2[2] = " || Расщепление вершины ||\n";

\_menu.active\_versions2[0] = " || > Отождествление вершин < ||\n";

\_menu.active\_versions2[1] = " || > Стягивание ребра < ||\n";

\_menu.active\_versions2[2] = " || > Расщепление вершины < ||\n";

\_menu.versions3[0] = " || Объединение ||\n";

\_menu.versions3[1] = " || Пересечение ||\n";

\_menu.versions3[2] = " || Кольцевая сумма ||\n";

\_menu.active\_versions3[0] = " || > Объединение < ||\n";

\_menu.active\_versions3[1] = " || > Пересечение < ||\n";

\_menu.active\_versions3[2] = " || > Кольцевая сумма < ||\n";

}

void mainMenuPos(int i) {

int k = 0;

int max = 4;

system("cls");

printf("%s", \_menu.up);

printf("%s", \_menu.wall);

do {

if (k == i) {

printf("%s", \_menu.active\_text[i]);

}

else {

printf("%s", \_menu.text[k]);

}

printf("%s", \_menu.wallBottom);

printf("%s", \_menu.wall);

k++;

} while (k < max);

if (k == i) {

printf("%s", \_menu.active\_text[k]);

}

else {

printf("%s", \_menu.text[k]);

}

printf("%s", \_menu.bottom);

return;

}

void Version2G(int i) {

int k = 0;

int max = 2;

system("cls");

printf("%s", \_menu.up);

printf("%s", \_menu.wall);

do {

if (k == i) {

printf("%s", \_menu.active\_versions2[i]);

}

else {

printf("%s", \_menu.versions2[k]);

}

printf("%s", \_menu.wallBottom);

printf("%s", \_menu.wall);

k++;

} while (k < max);

if (k == i) {

printf("%s", \_menu.active\_versions2[k]);

}

else {

printf("%s", \_menu.versions2[k]);

}

printf("%s", \_menu.bottom);

return;

}

void mainMenu() {

Menu = 0;

system("cls");

mainMenuPos(Menu);

char cur;

do {

cur = \_getch();

if (cur == 80) {

Menu++;

switch (Menu) {

case 5:

Menu = 0;

mainMenuPos(Menu);

break;

default:

mainMenuPos(Menu);

}

}

if (cur == 72) {

Menu--;

switch (Menu) {

case -1:

Menu = 4;

mainMenuPos(Menu);

break;

default:

mainMenuPos(Menu);

}

}

} while (cur != 13);

}

int versions2Menu() {

char cur;

int isf = 0;

Version2G(isf);

do {

cur = \_getch();

if (cur == 80) {

isf++;

switch (isf) {

case 3:

isf = 0;

Version2G(isf);

break;

default:

Version2G(isf);

break;

}

}

if (cur == 72) {

isf--;

switch (isf) {

case -1:

isf = 2;

Version2G(isf);

break;

default:

Version2G(isf);

break;

}

}

if (cur == 27) {

return 0;

}

} while (cur != 13);

return (isf);

}

void versionsMenuG(int isf) {

int k = 0;

system("cls");

printf("%s", \_menu.up);

printf("%s", \_menu.wall);

if (isf == 0) {

printf("%s", \_menu.active\_versions[k]);

printf("%s", \_menu.wallBottom);

printf("%s", \_menu.wall);

printf("%s", \_menu.versions[k + 1]);

}

else {

printf("%s", \_menu.versions[k]);

printf("%s", \_menu.wallBottom);

printf("%s", \_menu.wall);

printf("%s", \_menu.active\_versions[k + 1]);

}

printf("%s", \_menu.bottom);

return;

}

void Version3G(int i) {

int k = 0;

int max = 2;

system("cls");

printf("%s", \_menu.up);

printf("%s", \_menu.wall);

do {

if (k == i) {

printf("%s", \_menu.active\_versions3[i]);

}

else {

printf("%s", \_menu.versions3[k]);

}

printf("%s", \_menu.wallBottom);

printf("%s", \_menu.wall);

k++;

} while (k < max);

if (k == i) {

printf("%s", \_menu.active\_versions3[k]);

}

else {

printf("%s", \_menu.versions3[k]);

}

printf("%s", \_menu.bottom);

return;

}

int versions3Menu() {

char cur;

int isf = 0;

Version3G(isf);

do {

cur = \_getch();

if (cur == 80) {

isf++;

switch (isf) {

case 3:

isf = 0;

Version3G(isf);

break;

default:

Version3G(isf);

break;

}

}

if (cur == 72) {

isf--;

switch (isf) {

case -1:

isf = 2;

Version3G(isf);

break;

default:

Version3G(isf);

break;

}

}

if (cur == 27) {

return 0;

}

} while (cur != 13);

return (isf);

}

**Файл number1.cpp**

#include "Header.h"

graf\* sozdat(int versh) {

graf\* grafon = (graf\*)malloc(sizeof(graf));

grafon->size = versh;

grafon->nodes = (node\*\*)malloc(versh \* sizeof(node\*));

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

grafon->nodes[i] = NULL;

}

return grafon;

}

node\* sozdatnode(int index) {

node\* newnode = (node\*)malloc(sizeof(node));

newnode->numb = index;

newnode->next = NULL;

return newnode;

}

void addgran(graf\* grafon, int from, int to) {

node\* newnode = sozdatnode(from);

newnode->next = grafon->nodes[to];

grafon->nodes[to] = newnode;

newnode = sozdatnode(to);

newnode->next = grafon->nodes[from];

grafon->nodes[from] = newnode;

}

void number1\_1() {

srand(time(0));

setlocale(LC\_ALL, "Russian");

int\*\* a, \*\* b, i, n, m, n1, m1, j;

printf("\n Введите количество вершин: ");

scanf("%d", &n);

m = n;

a = (int\*\*)malloc(n \* sizeof(int\*));

j = 1;

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

a[i] = (int\*)malloc(m \* sizeof(int));

}

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

a[i][i] = 0;

for (j; j < m; j++) {

a[i][j] = rand() % 2;

a[j][i] = a[i][j];

}

j = j - m + i + 1;

}

printf("\n");

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

a[i][i] = 0;

}

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

printf(" ");

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

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

}

printf("\n");

}

printf("\n");

}

void number1\_2() {

srand(time(0));

setlocale(LC\_ALL, "Russian");

int\*\* a, \*\* b, i, n, m, n1, m1, j;

printf("\n Введите количество вершин: ");

scanf("%d", &n);

m = n;

a = (int\*\*)malloc(n \* sizeof(int\*));

j = 1;

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

a[i] = (int\*)malloc(m \* sizeof(int));

}

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

a[i][i] = 0;

for (j; j < m; j++) {

a[i][j] = rand() % 2;

a[j][i] = a[i][j];

}

j = j - m + i + 1;

}

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

a[i][i] = 0;

}

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

printf(" ");

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

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

}

printf("\n");

}

printf("\n");

graf\* grafon = sozdat(n);

j = 1;

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

for (j; j < m; j++) {

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

addgran(grafon, i, j);

}

}

j = j - m + i + 1;

}

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

node\* temp = grafon->nodes[i];

printf(" %d", i);

while (temp) {

printf("->%d", temp->numb);

temp = temp->next;

}

printf("\n");

}

}

**Файл number2.cpp**

#include "Header.h"

int equation\_matrix(int\*\* a, int n, int b, int c) {

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

if (a[c][j] == 1) {

a[b][j] = 1;

a[j][b] = a[b][j];

}

}

if (a[b][c] == 0)

a[b][b] = 0;

a[b][c] = 0;

a[c][b] = 0;

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

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

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

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

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

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

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

a[i] = (int)realloc(a[i], (n - 1)sizeof(int));

free(a[n - 1]);

n--;

return n;

}

int pullin\_matrix(int\*\* a, int n, int b, int c) {

if (a[b][c] == 0) {

printf("\n Указанные вершины не соединены");

return 0;

}

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

if (a[c][j] == 1) {

a[b][j] = 1;

a[j][b] = a[b][j];

}

}

a[b][b] = 0;

a[b][c] = 0;

a[c][b] = 0;

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

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

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

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

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

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

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

realloc(a[i], (n - 1) \* sizeof(int));

free(a[n - 1]);

n--;

return n;

}

int split\_matrix(int\*\* a, int n, int c) {

realloc(a, (n + 1) \* sizeof(int));

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

realloc(a[i], (n + 1) \* sizeof(int));

a[n] = (int\*)malloc((n + 1) \* sizeof(int));

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

a[i][n] = a[i][c];

a[i][c] = 0;

a[n][i] = a[i][n];

}

for (int i = n / 2; i < (n + 1); i++) {

a[i][n] = 0;

a[n][i] = a[i][n];

}

a[c][n] = 1;

a[n][c] = a[c][n];

n++;

return n;

}

int\*\* number2\_create() {

srand(time(0));

setlocale(LC\_ALL, "Russian");

int\*\* a, \*\* b, i, n, m, n1, m1, j;

printf("\n Введите количество вершин: ");

scanf("%d", &n);

m = n;

size = n;

a = (int\*\*)malloc(n \* sizeof(int\*));

j = 1;

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

a[i] = (int\*)malloc(m \* sizeof(int));

}

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

a[i][i] = 0;

for (j; j < m; j++) {

a[i][j] = rand() % 2;

a[j][i] = a[i][j];

}

j = j - m + i + 1;

}

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

a[i][i] = 0;

}

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

printf(" ");

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

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

}

printf("\n");

}

printf("\n");

return(a);

}

void number2\_a(int \*\*a) {

int b, c, i, n, j;

n = size;

printf(" Введите номера вершин для отождествления: ");

scanf("%d %d", &b, &c);

n = equation\_matrix(a, n, b, c);

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

printf(" ");

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

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

}

printf("\n");

}

}

void number2\_b(int \*\*a) {

int b, c, i, n, j;

n = size;

printf(" Введите номера вершин для стягивания: ");

scanf("%d %d", &b, &c);

n = pullin\_matrix(a, n, b, c);

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

printf(" ");

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

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

}

printf("\n");

}

}

void number2\_v(int\*\* a) {

int b, c, i, n, j;

n = size;

printf(" Выберите номера для расщепления: ");

scanf("%d", &c);

n = split\_matrix(a, n, c);

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

printf(" ");

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

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

}

printf("\n");

}

}

//////// List part

int equation\_list(graf\* grafon, int versh, int b, int c) {

node\* temp\_c = grafon->nodes[c];

node\* temp\_b = grafon->nodes[b];

while (temp\_c) {

temp\_b = grafon->nodes[b];

while (temp\_b->next) {

if (temp\_b->numb == temp\_c->numb)

break;

temp\_b = temp\_b->next;

}

if (temp\_b->numb != temp\_c->numb) {

node\* temp = sozdatnode(temp\_c->numb);

temp\_b->next = temp;

}

temp\_c = temp\_c->next;

}

temp\_b = grafon->nodes[b];

node\* temp\_b\_prev = NULL;

while (temp\_b) {

if (temp\_b->numb == c) {

if (temp\_b\_prev == NULL)

grafon->nodes[b] = temp\_b->next;

else {

if (temp\_b->next == NULL)

temp\_b\_prev->next = NULL;

else

temp\_b\_prev->next = temp\_b->next;

}

free(temp\_b);

break;

}

temp\_b\_prev = temp\_b;

temp\_b = temp\_b->next;

}

for (int i = c; i < versh - 1; i++)

grafon->nodes[i] = grafon->nodes[i + 1];

realloc(grafon->nodes, (versh - 1) \* sizeof(node));

versh--;

grafon->size--;

node\* temp = NULL;

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

temp = grafon->nodes[i];

while (temp) {

if (temp->numb == c)

temp->numb = b;

if (temp->numb >= c)

temp->numb--;

temp = temp->next;

}

}

return versh;

}

int pullin\_list(graf\* grafon, int versh, int b, int c) {

node\* temp\_b = grafon->nodes[b];

while (temp\_b) {

if (temp\_b->numb == c)

break;

temp\_b = temp\_b->next;

}

if (temp\_b == NULL) {

printf("\n Указанные вершины не соединены \n");

system("pause");

return versh;

}

node\* temp\_c = grafon->nodes[c];

while (temp\_c) {

temp\_b = grafon->nodes[b];

while (temp\_b->next) {

if ((temp\_b->numb == temp\_c->numb) || (temp\_c->numb == b))

break;

temp\_b = temp\_b->next;

}

if ((temp\_b->numb != temp\_c->numb) && (temp\_c->numb != b)) {

node\* temp = sozdatnode(temp\_c->numb);

temp\_b->next = temp;

}

temp\_c = temp\_c->next;

}

temp\_b = grafon->nodes[b];

node\* temp\_b\_prev = NULL;

while (temp\_b) {

if (temp\_b->numb == c) {

if (temp\_b\_prev == NULL)

grafon->nodes[b] = temp\_b->next;

else {

if (temp\_b->next == NULL)

temp\_b\_prev->next = NULL;

else

temp\_b\_prev->next = temp\_b->next;

}

free(temp\_b);

break;

}

temp\_b\_prev = temp\_b;

temp\_b = temp\_b->next;

}

for (int i = c; i < versh - 1; i++)

grafon->nodes[i] = grafon->nodes[i + 1];

realloc(grafon->nodes, (versh - 1) \* sizeof(node));

versh--;

grafon->size--;

node\* temp = NULL;

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

temp = grafon->nodes[i];

while (temp) {

if (temp->numb == c)

temp->numb = b;

if(temp->numb>=c)

temp->numb--;

temp = temp->next;

}

}

return versh;

}

int split\_list(graf\* grafon, int versh, int b) {

realloc(grafon->nodes, (versh + 1) \* sizeof(node));

node\* temp\_b = grafon->nodes[b];

node\* temp\_b\_prev = NULL;

node\* temp;

grafon->nodes[versh] = sozdatnode(b);

int i = 0;

while (temp\_b->next) {

if ((i % 2) == 1) {

temp\_b\_prev->next = temp\_b->next;

temp\_b->next = grafon->nodes[versh];

grafon->nodes[versh] = temp\_b;

temp\_b = temp\_b\_prev->next;

}

else {

temp\_b\_prev = temp\_b;

temp\_b = temp\_b->next;

}

i++;

}

if ((i % 2) == 1) {

temp\_b\_prev->next = NULL;

temp\_b->next = grafon->nodes[versh];

grafon->nodes[versh] = temp\_b;

}

temp\_b = sozdatnode(versh);

temp\_b->next = grafon->nodes[b];

grafon->nodes[b] = temp\_b;

grafon->size++;

versh++;

return versh;

}

graf\* num2\_list\_create() {

int\*\* a, \*\* b, i, n, m, n1, m1, j;

printf("\n Введите количество вершин: ");

scanf("%d", &n);

m = n;

a = (int\*\*)malloc(n \* sizeof(int\*));

j = 1;

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

a[i] = (int\*)malloc(m \* sizeof(int));

}

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

a[i][i] = 0;

for (j; j < m; j++) {

a[i][j] = rand() % 2;

a[j][i] = a[i][j];

}

j = j - m + i + 1;

}

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

a[i][i] = 0;

}

graf\* grafon = sozdat(n);

j = 1;

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

for (j; j < m; j++) {

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

addgran(grafon, i, j);

}

}

j = j - m + i + 1;

}

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

node\* temp = grafon->nodes[i];

printf(" %d", i);

while (temp) {

printf("->%d", temp->numb);

temp = temp->next;

}

printf("\n");

}

return grafon;

}

void number2\_a\_list(graf\* grafon) {

int b, c, i, n;

printf(" Введите номера вершин для отождествления: ");

scanf("%d %d", &b, &c);

n = grafon->size;

n = equation\_list(grafon, n, b, c);

node\* temp = NULL;

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

temp = grafon->nodes[i];

printf("%d", i);

while (temp) {

printf("->%d", temp->numb);

temp = temp->next;

}

printf("\n");

}

}

void number2\_b\_list(graf\* grafon) {

int b, c, i, n;

printf(" Введите номера вершин для стягивания: ");

scanf("%d %d", &b, &c);

n = grafon->size;

n=pullin\_list(grafon, n, b, c);

//n = split\_list(grafon, n, b);

node\* temp = NULL;

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

temp = grafon->nodes[i];

printf("%d", i);

while (temp) {

printf("->%d", temp->numb);

temp = temp->next;

}

printf("\n");

}

}

void number2\_v\_list(graf\* grafon) {

int b, c, i, n;

printf(" Выберите номера для расщепления: ");

scanf("%d %d", &b, &c);

n = grafon->size;

n = split\_list(grafon, n, b);

node\* temp = NULL;

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

temp = grafon->nodes[i];

printf("%d", i);

while (temp) {

printf("->%d", temp->numb);

temp = temp->next;

}

printf("\n");

}

}

**Файл number3.cpp**

#include "Header.h"

int\*\* a;

int unite(int\*\* b, int n, int m) {

if (m > n) {

a = (int\*\*)realloc(a, m \* sizeof(\*a));

for (int i = n; i < m; i++)

a[i] = (int\*)calloc(m, sizeof(\*\*a));

n = m;

}

int j = 0;

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

for (j; j < m; j++) {

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

a[i][j] = 1;

a[j][i] = a[i][j];

}

else {

if (a[i][j] != 1)

a[i][j] = 0;

a[j][i] = a[i][j];

}

}

j = j - m + i + 1;

}

return n;

}

int cross(int\*\* b, int n, int m) {

if (n > m) {

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

a[i] = (int\*)realloc(a[i], m \* sizeof(\*\*a));

for (int i = m; i < n; i++)

free(a[i]);

n = m;

}

else

m = n;

int j = 0;

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

for (j; j < m; j++) {

if ((a[i][j] != 1) || (b[i][j] != 1)) {

a[i][j] = 0;

a[j][i] = a[i][j];

}

}

j = j - m + i + 1;

}

return n;

}

int short\_circuit(int\*\* b, int n, int m) {

if (n > m) {

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

a[i] = (int\*)realloc(a[i], m \* sizeof(\*\*a));

for (int i = m; i < n; i++)

free(a[i]);

n = m;

}

else

m = n;

int j = 0;

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

for (j; j < m; j++) {

if (((b[i][j] == 1) && (a[i][j] != 1)) || ((b[i][j] != 1) && (a[i][j] == 1))) {

a[i][j] = 1;

a[j][i] = a[i][j];

}

else {

a[i][j] = 0;

a[j][i] = a[i][j];

}

}

j = j - m + i + 1;

}

int k;

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

k = 0;

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

if (a[i][j] == 1)

k++;

}

if (k == 0) {

for (int q = i; q < m - 1; q++)

for (int r = 0; r < m; r++)

a[q][r] = a[q + 1][r];

for (int q = 0; q < m; q++)

for (int r = i; r < m - 1; r++)

a[q][r] = a[q][r + 1];

for (int q = 0; q < m; q++)

a[q] = (int\*)realloc(a[q], (m - 1) \* sizeof(int));

free(a[m - 1]);

m--;

i--;

}

}

return m;

}

void number3\_create(int inside\_menu) {

int\*\* d, b, c, i, n, m, j;

printf("\n Введите количество вершин для первой матрицы: ");

scanf("%d", &n);

a = (int\*\*)malloc(n \* sizeof(\*a));

j = 1;

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

a[i] = (int\*)malloc(n \* sizeof(\*\*a));

}

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

a[i][i] = 0;

for (j; j < n; j++) {

a[i][j] = rand() % 2;

a[j][i] = a[i][j];

}

j = j - n + i + 1;

}

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

a[i][i] = 0;

}

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

printf(" ");

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

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

}

printf("\n");

}

printf("\n");

printf(" Ведите количество вершин для второй матрицы: ");

scanf("%d", &m);

d = (int\*\*)malloc(m \* sizeof(int\*));

j = 1;

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

d[i] = (int\*)malloc(m \* sizeof(int));

}

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

d[i][i] = 0;

for (j; j < m; j++) {

d[i][j] = rand() % 2;

d[j][i] = d[i][j];

}

j = j - m + i + 1;

}

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

d[i][i] = 0;

}

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

printf(" ");

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

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

}

printf("\n");

}

printf("\n");

switch (inside\_menu) {

case 0:

n = unite(d, n, m);

break;

case 1:

n = cross(d, n, m);

break;

case 2:

n = short\_circuit(d, n, m);

break;

}

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

printf(" ");

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

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

}

printf("\n");

}

}

**Файл number4.cpp**

#include "Header.h"

int\*\* e;

int\*\* a2;

int descartes\_multiples\_puta\_madre(int\*\* b, int n, int m) {

e = (int\*\*)calloc(m \* n, sizeof(int\*));

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

e[i] = (int\*)calloc(m \* n, sizeof(int));

}

int j = 0;

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

for (j; j < n \* m; j++) {

if ((i / m == j / m) && (b[i % m][j % m] == 1))

e[i][j] = 1;

else {

if ((a2[i / m][j / m] == 1) && (i % m == j % m))

e[i][j] = 1;

else

e[i][j] = 0;

}

e[j][i] = e[i][j];

}

j = j - n \* m + i + 1;

}

return n \* m;

}

void number4() {

int\*\* d, b, c, i, n, m, j;

printf("\n Ведите количество вершин для первой матрицы: ");

scanf("%d", &n);

a2 = (int\*\*)malloc(n \* sizeof(\*a2));

j = 1;

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

a2[i] = (int\*)malloc(n \* sizeof(\*\*a2));

}

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

a2[i][i] = 0;

for (j; j < n; j++) {

a2[i][j] = rand() % 2;

a2[j][i] = a2[i][j];

}

j = j - n + i + 1;

}

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

a2[i][i] = 0;

}

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

printf(" ");

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

printf("%2d", a2[i][j]);

}

printf("\n");

}

printf("\n");

printf(" Ведите количество вершин для второй матрицы: ");

scanf("%d", &m);

d = (int\*\*)malloc(m \* sizeof(int\*));

j = 1;

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

d[i] = (int\*)malloc(m \* sizeof(int));

}

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

d[i][i] = 0;

for (j; j < m; j++) {

d[i][j] = rand() % 2;

d[j][i] = d[i][j];

}

j = j - m + i + 1;

}

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

d[i][i] = 0;

}

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

printf(" ");

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

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

}

printf("\n");

}

printf("\n");

int l = descartes\_multiples\_puta\_madre(d, n, m);

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

printf(" ");

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

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

}

printf("\n");

}

}

**Файл Header1.h**

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#include <locale>

#include <time.h>

struct menu {

char up[200];

char bottom[200];

const char\* text[5];

const char\* active\_text[5];

const char\* versions[2];

const char\* active\_versions[2];

const char\* versions2[3];

const char\* active\_versions2[3];

const char\* versions3[3];

const char\* active\_versions3[3];

const char\* number4[3];

const char\* active\_number4[3];

char wall[200];

char wallBottom[200];

};

struct node {

int numb;

node\* next;

};

struct graf {

node\*\* nodes;

int size;

};

//menu

extern int Menu;

extern menu \_menu;

extern int size;

void create\_menu();

void mainMenuPos(int i);

void mainMenu();

void select();

void versionsMenuG(int isf);

void versionsMenu();

void Version2G(int i);

int versions2Menu();

void number3Menu();

int versions3Menu();

void Version3G(int i);

//number1

graf\* sozdat(int versh);

node\* sozdatnode(int index);

void addgran(graf\* grafon, int from, int to);

void number1\_1();

void number1\_2();

//nuber2

int equation\_matrix(int\*\* a, int n, int b, int c);

int pullin\_matrix(int\*\* a, int n, int b, int c);

int split\_matrix(int\*\* a, int n, int c);

int\*\* number2\_create();

void number2\_a(int\*\* a);

void number2\_b(int\*\* a);

void number2\_v(int\*\* a);

void number\_2\_List();

int equation\_list(graf\* grafon, int versh, int b, int c);

int pullin\_list(graf\* grafon, int versh, int b, int c);

int split\_list(graf\* grafon, int versh, int b);

graf\* num2\_list\_create();

void number2\_a\_list(graf\* grafon);

void number2\_b\_list(graf\* grafon);

void number2\_v\_list(graf\* grafon);

// Num3

int unite(int\*\* b, int n, int m);

int cross(int\*\* b, int n, int m);

int short\_circuit(int\*\* b, int n, int m);

void number3\_create(int inside\_menu);

//Num 4

void number4();

int descartes\_multiples\_puta\_madre(int\*\* b, int n, int m);

**Задание 1.**

**Результат работы программы**

1. Матричная форма (рис. 1)

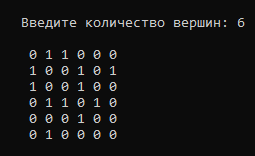


Рисунок 1 Генерация матрицы смежности

1. Списки смежности (рис. 2)

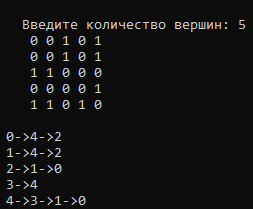


Рисунок 2 Список смежности

**Задание 2.**

**Результат работы программы**

1. Матричная форма

а) Отождествление вершин (рис. 3)

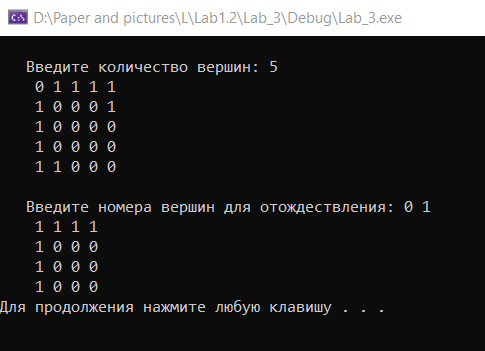


Рисунок 3 Отождествление вершин

б) Стягивание ребра (рис. 4)

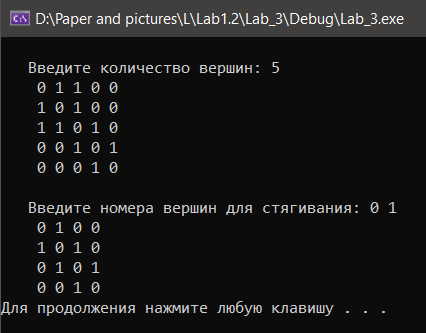


Рисунок 4 Стягивание ребра

в) Расщепление вершины (рис. 5)

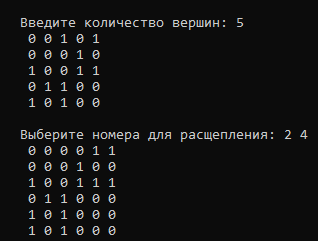


Рисунок 5 Расщепление вершины

1. Списки смежности

а) Отождествление вершин (рис. 6)

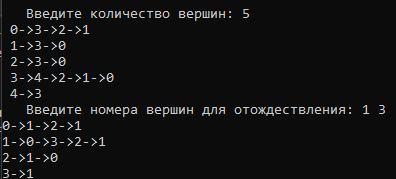


Рисунок 6 Отождествление вершин

б) Стягивание ребра (рис. 7)

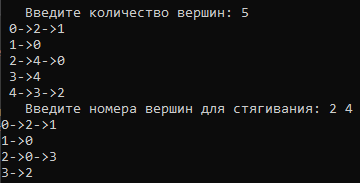


Рисунок 7 Стягивание ребра

в) Расщепление вершины (рис. 8)

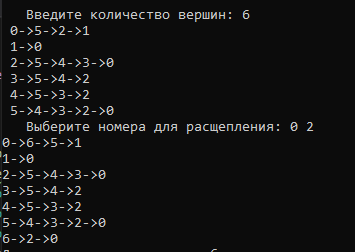


Рисунок 8 Расщепление вершины

**Задание 3.**

**Результат работы программы**

1. Объединение (рис. 9)

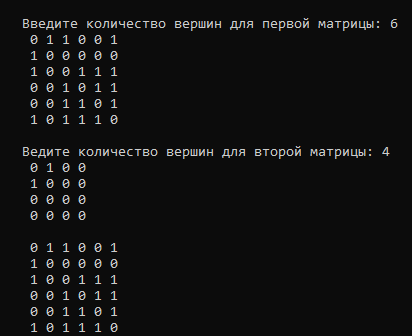


Рисунок 9 Объединение G = G1  G2

1. Пересечение (рис. 10)

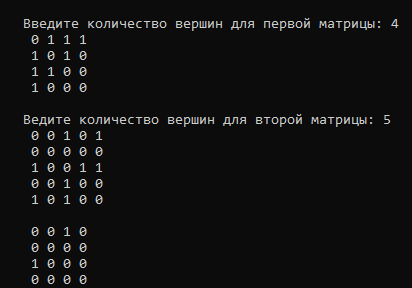


Рисунок 10 Пересечение G = G1  G2

1. Кольцевая сумма (рис. 11)

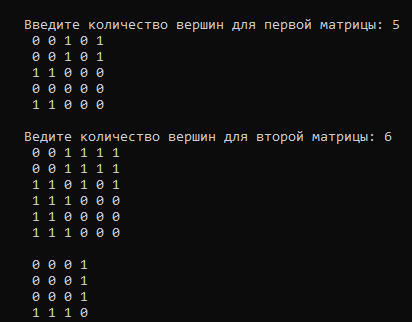


Рисунок 11 Кольцевая сумма G = G1  G2

**Задание 4.**

**Результат работы программы**

1. Декартово произведение графов (рис. 12).

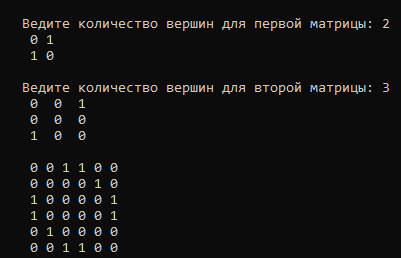


Рисунок 12 Декартово произведение

**Вывод**

В ходе выполнения данной лабораторной работы были исследованы унарные, бинарные операции над графами, приобретены практические навыки решения задач с использованием основ теории графов.