Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «унарные и бинарные операции над графами»

**Выполнил:**

студент группы 20ВВ3

Духнов Олег

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

# Унарные и бинарные операции над графами.

# Лабораторное задание

**Задание 1**

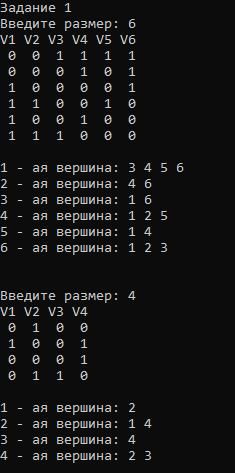
1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.
2. Определите радиус и диаметр графа *G*, используя матрицу смежности графа.
3. Определите подмножества периферийных и центральных вершин графа *G*, используя матрицу смежности.
4. Найдите изолированные, концевые и доминирующие вершины.

**Задание 2\***

1. Постройте для графа G матрицу инцидентности.
2. Определите радиус и диаметр графа *G*, используя матрицу инцидентности графа.
3. Определите подмножества периферийных и центральных вершин графа *G*, используя матрицу инцидентности.

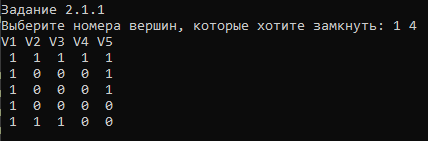
# Результат работы программы

**Задание 1.** Результаты работы программы показаны на рисунке 1.



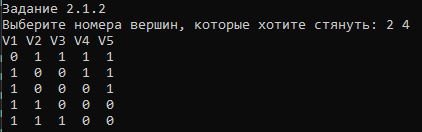
**Рисунок 1 – Результат работы программы**

**Задание 2.1.1.** Отождествление вершин для графа в матричной форме.



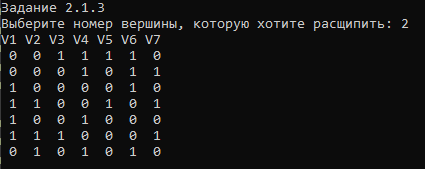
**Рисунок 2 – Результат работы программы**

**Задание 2.1.2.** Стягивание ребра для графа в матричной форме.



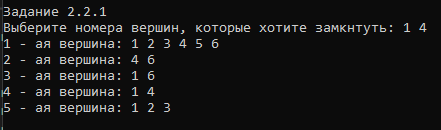
**Рисунок 3 – Результат работы программы**

**Задание 2.1.3.** Расщепление вершин для графа в матричной форме.



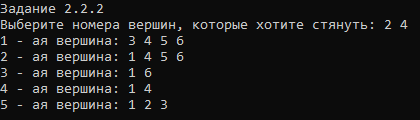
**Рисунок 4 – Результат работы программы**

**Задание 2.2.1.** Отождествление вершин для графа, представленным списком смежности.



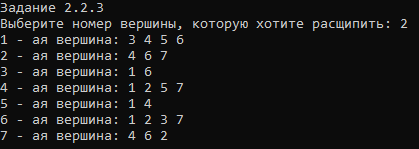
**Рисунок 5 – Результат работы программы**

**Задание 2.2.2.** Стягивание ребра для графа, представленным списком смежности.



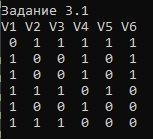
**Рисунок 6 – Результат работы программы**

**Задание 2.2.3.** Расщепление вершины для графа, представленным списком смежности.



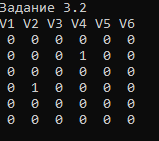
**Рисунок 7 – Результат работы программы**

**Задание 3.1.** Операция объединения.



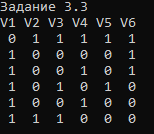
**Рисунок 8 – Результат работы программы**

**Задание 3.2.** Операция пересечения.



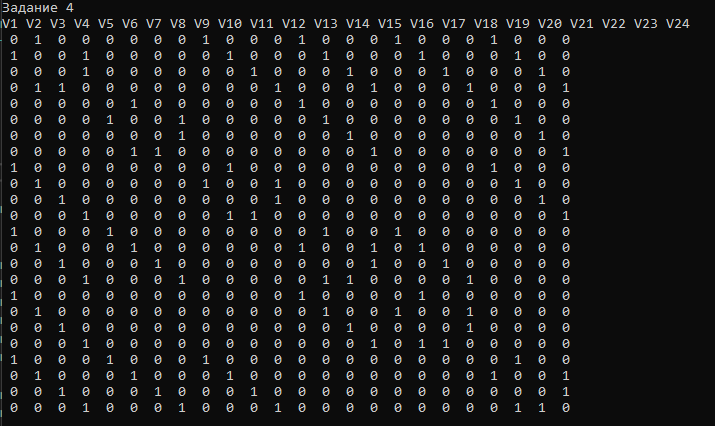
**Рисунок 9 – Результат работы программы**

**Задание 3.3.** Операция кольцевой суммы.



**Рисунок 10 – Результат работы программы**

**Задание 4.** Декартово произведение.



**Рисунок 11 – Результат работы программы**

**Вывод:** яизучил унарные и бинарные операции над графами.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <stdbool.h>

#include <conio.h>

#include <time.h>

#include <stack>

#include <queue>

using namespace std;

typedef struct Node

{

int vertex;

Node\* Next;

}Node;

typedef struct Graf

{

int num;

Node\*\* A2;

}Graf;

Node\* init\_node(int v)

{

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

NewNode->vertex = v;

NewNode->Next = NULL;

return NewNode;

}

Graf\* init\_graf(int n)

{

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

graf->num = n;

graf->A2 = (Node\*\*)malloc(n \* sizeof(Node\*));

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

graf->A2[i] = NULL;

return graf;

}

void print(Graf\* graf)

{

for (int i = 0; i < graf->num; i++)

{

Node\* temp = graf->A2[i];

printf("%d - ая вершина: ", i + 1);

while (temp)

{

printf("%d ", temp->vertex + 1);

temp = temp->Next;

}

printf("\n");

}

}

void sort\_spisok(Graf\* graf, int n)

{

int vertex;

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

{

Node\* temp = graf->A2[i], \* temp1 = graf->A2[i];

if (temp1 != NULL && temp1->Next != NULL)

temp1 = temp1->Next;

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

{

while (temp != NULL)

{

if (temp->vertex > temp1->vertex)

{

vertex = temp->vertex;

temp->vertex = temp1->vertex;

temp1->vertex = vertex;

}

temp = temp->Next; if (temp1->Next != NULL) temp1 = temp1->Next;

}

temp = graf->A2[i], temp1 = graf->A2[i];

if (temp1 != NULL && temp1->Next != NULL)

temp1 = temp1->Next;

}

}

}

void add\_spisok(int i, int j, Graf\* graf)

{

Node\* NewNode = init\_node(j);

if (graf->A2[i] == NULL)

{

graf->A2[i] = NewNode;

NewNode = NULL;

}

Node\* temp = graf->A2[i];

while (temp->Next != NULL)

temp = temp->Next;

temp->Next = NewNode;

NewNode = init\_node(i);

if (graf->A2[j] == NULL)

{

graf->A2[j] = NewNode;

NewNode = NULL;

}

temp = graf->A2[j];

while (temp->Next != NULL)

temp = temp->Next;

temp->Next = NewNode;

}

int\*\* create\_arr(int x)

{

int s = 1, \*\*A;

A = (int\*\*)malloc(x \* sizeof(int\*));

srand(time(NULL));

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

{

A[i] = (int\*)malloc(x \* sizeof(int));

A[i][i] = 0;

for (int j = s; j < x; j++)

{

if (rand() % 100 > 62)

A[i][j] = 0;

else

A[i][j] = 1;

}

s++;

}

s = 1;

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

{

for (int j = s; j < x; j++)

{

A[j][i] = A[i][j];

}

s++;

}

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

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

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

{

printf("\n");

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

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

}

return A;

}

void union\_graf(int\*\* M1, int\*\* M2, int n, int x)

{

int\*\* A1 = (int\*\*)malloc(n \* sizeof(int\*));

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

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

A1[i][j] = M1[i][j];

}

int\*\* A2 = (int\*\*)malloc(x \* sizeof(int\*));

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

{

A2[i] = (int\*)malloc(x \* sizeof(int));

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

A2[i][j] = M2[i][j];

}

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

{

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

{

if (A1[i][j] != A2[i][j])

A1[i][j] = A1[i][j] + A2[i][j];

}

}

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

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

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

{

printf("\n");

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

printf("%2d ", A1[i][j]);

}

}

void crossing\_graf(int\*\* M1, int\*\* M2, int n, int x)

{

int\*\* A1 = (int\*\*)malloc(n \* sizeof(int\*));

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

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

A1[i][j] = M1[i][j];

}

int\*\* A2 = (int\*\*)malloc(x \* sizeof(int\*));

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

{

A2[i] = (int\*)malloc(x \* sizeof(int));

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

A2[i][j] = M2[i][j];

}

int s = 1;

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

{

for (int j = s; j < x; j++)

{

if (A1[i][j] == 1 && A2[i][j] == 1)

A1[i][j] = 1;

else

A1[i][j] = 0;

}

s++;

}

s = x;

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

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

A1[i][j] = 0;

s = 1;

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

{

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

A1[j][i] = A1[i][j];

s++;

}

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

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

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

{

printf("\n");

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

printf("%2d ", A1[i][j]);

}

}

void annular\_sum\_graf(int\*\* M1, int\*\* M2, int n, int x)

{

int\*\* A1 = (int\*\*)malloc(n \* sizeof(int\*));

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

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

A1[i][j] = M1[i][j];

}

int\*\* A2 = (int\*\*)malloc(x \* sizeof(int\*));

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

{

A2[i] = (int\*)malloc(x \* sizeof(int));

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

A2[i][j] = M2[i][j];

}

int s = 1;

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

{

for (int j = s; j < x; j++)

{

if (A1[i][j] == A2[i][j])

A1[i][j] = 0;

else

A1[i][j] = 1;

}

s++;

}

s = 1;

int l = 0;

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

{

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

A1[j][i] = A1[i][j];

s++;

}

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

{

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

{

if (A1[p][j] == 1)

l++;

}

if (l == 0)

{

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

{

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

A1[i][j] = A1[i + 1][j];

}

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

{

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

A1[i][j] = A1[i][j + 1];

}

n--;

p--;

}

l = 0;

}

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

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

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

{

printf("\n");

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

printf("%2d ", A1[i][j]);

}

}

void closure\_graf(int\*\* M1, int n, int x)

{

int a, b, \*\* A1 = (int\*\*)malloc(n \* sizeof(int\*));

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

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

A1[i][j] = M1[i][j];

}

printf("Выберите номера вершин, которые хотите замкнуть: ");

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

a--; b--;

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

{

if (A1[a][i] != A1[b][i])

{

A1[a][i] = A1[a][i] + A1[b][i];

A1[i][a] = A1[a][i];

}

}

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

A1[b][i] = 0;

if (A1[a][a] == 0)

A1[a][a] = 1;

int s = 1, l = 0;

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

{

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

{

if (A1[p][j] == 1)

l++;

}

if (l == 0)

{

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

{

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

A1[i][j] = A1[i + 1][j];

}

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

{

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

A1[i][j] = A1[i][j + 1];

}

n--;

p--;

}

l = 0;

}

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

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

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

{

printf("\n");

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

printf("%2d ", A1[i][j]);

}

}

void closure\_graf\_spisok(int n, int x, Graf\* g)

{

Graf\* graf = init\_graf(n);

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

{

Node\* head, \* head1, \* temp = g->A2[i], \* temp1;

graf->A2[i] = (Node\*)malloc(sizeof(Node));

temp1 = graf->A2[i];

if (temp != NULL)

{

temp1->vertex = temp->vertex;

temp = temp->Next;

temp1->Next = NULL;

}

else

{

graf->A2[i] = NULL;

temp1 = graf->A2[i];

}

while (temp)

{

temp1->Next = (Node\*)malloc(sizeof(Node));

temp1 = temp1->Next;

temp1->vertex = temp->vertex;

temp1->Next = NULL;

temp = temp->Next;

}

}

int l = 0, a, b;

printf("Выберите номера вершин, которые хотите замкнтуть: ");

scanf("%d %d", &a, &b); a--; b--;

while (b <= a)

{

printf("Введите другие вершины: ");

scanf("%d %d", &a, &b); a--; b--;

}

Node\* head, \* heder, \* kemp = graf->A2[b], \* pemp = graf->A2[a], \* temp = graf->A2[a];

if (temp != NULL)

{

while (temp->Next != NULL)

temp = temp->Next;

heder = kemp;

head = pemp;

while (kemp != NULL)

{

while (pemp != NULL)

{

if (kemp->vertex == pemp->vertex || kemp->vertex == a)

l++;

pemp = pemp->Next;

}

pemp = head;

if (l == 0)

{

temp->Next = (Node\*)malloc(sizeof(Node));

temp = temp->Next;

temp->vertex = kemp->vertex;

}

temp->Next = NULL;

kemp = kemp->Next;

l = 0;

}

temp->Next = (Node\*)malloc(sizeof(Node));

temp = temp->Next;

temp->vertex = a;

temp->Next = NULL;

}

else

{

g->A2[a] = (Node\*)malloc(sizeof(Node));

temp = g->A2[a];

temp->vertex = b;

temp->Next = NULL;

}

if (temp->Next != NULL)

{

temp = graf->A2[a]; l = 0;

while (temp->Next != NULL)

{

if (temp->vertex == a)

l++;

temp = temp->Next;

}

if (l == 0)

{

temp->Next = (Node\*)malloc(sizeof(Node));

temp = temp->Next;

temp->vertex = a;

temp->Next = NULL;

}

}

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

graf->A2[i] = graf->A2[i + 1];

graf->A2[n - 1] = NULL;

sort\_spisok(graf, n);

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

{

Node\* semp = graf->A2[i];

printf("%d - ая вершина: ", i + 1);

while (semp != NULL)

{

printf("%d ", semp->vertex + 1);

semp = semp->Next;

}

printf("\n");

}

}

void delete\_vertex\_graf(int\*\* M1, int n)

{

int a, b, \*\* A1 = (int\*\*)malloc(n \* sizeof(int\*));

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

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

A1[i][j] = M1[i][j];

}

printf("Выберите номера вершин, которые хотите стянуть: ");

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

a--; b--;

if (A1[a][b] == A1[b][a])

{

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

{

if (A1[a][i] != A1[b][i])

{

A1[a][i] = A1[a][i] + A1[b][i];

A1[i][a] = A1[a][i];

}

}

}

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

A1[b][i] = 0;

if (A1[a][a] == 1)

A1[a][a] = 0;

int l = 0;

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

{

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

{

if (A1[p][j] == 1)

l++;

}

if (l == 0)

{

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

{

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

A1[i][j] = A1[i + 1][j];

}

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

{

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

A1[i][j] = A1[i][j + 1];

}

n--;

p--;

}

l = 0;

}

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

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

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

{

printf("\n");

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

printf("%2d ", A1[i][j]);

}

}

void delete\_vertex\_graf\_spisok(int n, int x, Graf\* g)

{

Graf\* graf = init\_graf(n);

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

{

Node\* head, \* head1, \* temp = g->A2[i], \* temp1;

graf->A2[i] = (Node\*)malloc(sizeof(Node));

temp1 = graf->A2[i];

if (temp != NULL)

{

temp1->vertex = temp->vertex;

temp = temp->Next;

temp1->Next = NULL;

}

else

{

graf->A2[i] = NULL;

temp1 = graf->A2[i];

}

while (temp)

{

temp1->Next = (Node\*)malloc(sizeof(Node));

temp1 = temp1->Next;

temp1->vertex = temp->vertex;

temp1->Next = NULL;

temp = temp->Next;

}

}

int a, b, l = 0, q = 0;

printf("Выберите номера вершин, которые хотите стянуть: ");

scanf("%d %d", &a, &b); a--; b--;

Node\* temp = graf->A2[a], \* pemp = graf->A2[a], \* kemp = graf->A2[b], \* head, \* header;

while (temp != NULL && temp->Next != NULL)

temp = temp->Next;

head = pemp; header = kemp;

while (kemp != NULL)

{

if (a == kemp->vertex)

q++;

kemp = kemp->Next;

}

kemp = header;

if (q != 0)

{

while (kemp != NULL)

{

while (pemp != NULL)

{

if (kemp->vertex == pemp->vertex || kemp->vertex == a)

l++;

pemp = pemp->Next;

}

pemp = head;

if (l == 0)

{

temp->Next = (Node\*)malloc(sizeof(Node));

temp = temp->Next;

temp->vertex = kemp->vertex;

}

temp->Next = NULL;

kemp = kemp->Next;

l = 0;

}

}

else

{

printf("\nДанные вершины нельзя стянуть!");

return;

}

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

graf->A2[i] = graf->A2[i + 1];

graf->A2[n - 1] = NULL;

sort\_spisok(graf, n);

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

{

Node\* semp = graf->A2[i];

printf("%d - ая вершина: ", i + 1);

while (semp != NULL)

{

printf("%d ", semp->vertex + 1);

semp = semp->Next;

}

printf("\n");

}

}

void split\_graf(int\*\* M1, int n)

{

int l = 0, z = n + 1, a, \*\* A1 = (int\*\*)malloc(n \* sizeof(int\*)), \*\* R1 = (int\*\*)malloc(z \* sizeof(int\*));

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

{

R1[i] = (int\*)malloc(z \* sizeof(int));

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

R1[i][j] = 0;

}

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

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

A1[i][j] = M1[i][j];

}

printf("Выберите номер вершины, которую хотите расщипить: ");

scanf("%d", &a);

a--;

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

{

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

R1[i][j] = A1[i][j];

R1[n][i] = A1[a][i];

R1[i][n] = A1[a][i];

}

R1[n][a] = 1; R1[a][n] = 1;

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

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

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

{

printf("\n");

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

printf("%2d ", R1[i][j]);

}

}

void split\_graf\_spisok(int n, int x, Graf\* gr)

{

Graf\* graf = init\_graf(n);

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

{

Node\* head, \* head1, \* temp = gr->A2[i], \* temp1;

graf->A2[i] = (Node\*)malloc(sizeof(Node));

temp1 = graf->A2[i];

if (temp != NULL)

{

temp1->vertex = temp->vertex;

temp = temp->Next;

temp1->Next = NULL;

}

else

{

graf->A2[i] = NULL;

temp1 = graf->A2[i];

}

while (temp)

{

temp1->Next = (Node\*)malloc(sizeof(Node));

temp1 = temp1->Next;

temp1->vertex = temp->vertex;

temp1->Next = NULL;

temp = temp->Next;

}

}

int a, z = n + 1, l = 0;

Graf\* g = init\_graf(z);

printf("Выберите номер вершины, которую хотите расщипить: ");

scanf("%d", &a);

a--;

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

{

Node \* head, \* head1, \* temp = graf->A2[i], \* temp1;

g->A2[i] = (Node\*)malloc(sizeof(Node));

temp1 = g->A2[i];

if (temp != NULL)

{

temp1->vertex = temp->vertex;

temp = temp->Next;

temp1->Next = NULL;

}

else

{

g->A2[i] = NULL;

temp1 = g->A2[i];

}

while (temp)

{

temp1->Next = (Node\*)malloc(sizeof(Node));

temp1 = temp1->Next;

temp1->vertex = temp->vertex;

temp1->Next = NULL;

temp = temp->Next;

}

}

Node\* temp = graf->A2[a], \* temp1; g->A2[n] = (Node\*)malloc(sizeof(Node));

temp1 = g->A2[n];

if (temp != NULL)

{

temp1->vertex = temp->vertex;

temp = temp->Next;

temp1->Next = NULL;

}

else

{

g->A2[n] = NULL;

temp1 = g->A2[n];

}

while (temp)

{

temp1->Next = (Node\*)malloc(sizeof(Node));

temp1 = temp1->Next;

temp1->vertex = temp->vertex;

temp1->Next = NULL;

temp = temp->Next;

}

temp = g->A2[a], temp1 = g->A2[n];

if (temp != NULL)

{

while (temp->Next != NULL)

temp = temp->Next;

while (temp1->Next != NULL)

temp1 = temp1->Next;

temp->Next = (Node\*)malloc(sizeof(Node));

temp = temp->Next;

temp->vertex = n;

temp->Next = NULL;

temp1->Next = (Node\*)malloc(sizeof(Node));

temp1 = temp1->Next;

temp1->vertex = a;

temp1->Next = NULL;

}

else

{

g->A2[a] = (Node\*)malloc(sizeof(Node)), g->A2[n] = (Node\*)malloc(sizeof(Node));

temp = g->A2[a]; temp1 = g->A2[n];

temp->vertex = n;

temp->Next = NULL;

temp1->vertex = a;

temp1->Next = NULL;

}

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

{

Node\* temp = g->A2[i];

if (temp != NULL)

{

while (temp->Next != NULL)

{

if (temp->vertex == a)

l++;

temp = temp->Next;

}

if (l == 1)

{

temp->Next = (Node\*)malloc(sizeof(Node));

temp = temp->Next;

temp->vertex = n;

temp->Next = NULL;

}

l = 0;

}

if (temp != NULL)

{

if (temp->Next == NULL && temp != NULL)

{

if (temp->vertex == a)

l++;

if (l == 1)

{

temp->Next = (Node\*)malloc(sizeof(Node));

temp = temp->Next;

temp->vertex = n;

temp->Next = NULL;

}

}

}

}

sort\_spisok(graf, z);

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

{

Node\* semp = g->A2[i];

printf("%d - ая вершина: ", i + 1);

while (semp != NULL)

{

printf("%d ", semp->vertex + 1);

semp = semp->Next;

}

printf("\n");

}

}

void decart\_graf(int\*\* M1, int\*\* M2, int n, int x)

{

int\*\* A1 = (int\*\*)malloc(n \* sizeof(int\*));

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

{

A1[i] = (int\*)malloc(n \* sizeof(int));

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

A1[i][j] = M1[i][j];

}

int\*\* A2 = (int\*\*)malloc(x \* sizeof(int\*));

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

{

A2[i] = (int\*)malloc(x \* sizeof(int));

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

A2[i][j] = M2[i][j];

}

int z = n \* x, s = -1, l = 0;

int\*\* D1= (int\*\*)malloc(z \* sizeof(int\*));

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

{

D1[i] = (int\*)malloc(z \* sizeof(int));

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

D1[i][j] = 0;

}

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

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

{

s++;

l = 0;

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

for (int p = 0; p < x; p++)

{

if (i == p && k == j)

D1[s][l] = 0;

else if (i == j)

D1[s][l] = A2[k][p];

else if (k == p)

D1[s][l] = A1[i][j];

else

D1[s][l] = 0;

l++;

}

}

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

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

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

{

printf("\n");

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

printf("%2d ", D1[i][j]);

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int\*\* t, \*\* r, n, x;

printf("Задание 1\n");

printf("Введите размер: ");

scanf("%d", &n);

t = create\_arr(n);

printf("\n\n");

Graf\* graf = init\_graf(n);

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

{

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

{

if (t[i][j] == 1 && i > j)

add\_spisok(i, j, graf);

}

}

print(graf);

printf("\n\nВведите размер: ");

scanf("%d", &x);

while (x > n)

{

printf("Размер первой матрицы должен быть меньше!");

printf("\nВведите другой размер, пожалуйста: ");

scanf("%d", &x);

}

r = create\_arr(x);

printf("\n\n");

Graf\* graff = init\_graf(x);

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

{

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

{

if (r[i][j] == 1 && i > j)

add\_spisok(i, j, graff);

}

}

print(graff);

//printf("\nЗадание 2.1.1\n"); closure\_graf(t, n, x); printf("\n\n"); \_getch();// closure\_graf(r, x, n); printf("\n\n"); \_getch();

//printf("\nЗадание 2.1.2\n"); delete\_vertex\_graf(t, n); printf("\n\n"); \_getch(); //delete\_vertex\_graf(r, x); printf("\n\n"); \_getch();

//printf("\nЗадание 2.1.3\n"); split\_graf(t, n); printf("\n\n"); \_getch();// split\_graf(t, x); printf("\n\n"); \_getch();

//printf("\nЗадание 2.2.1\n"); closure\_graf\_spisok(n, x, graf); printf("\n\n"); \_getch(); //printf("\nЗадание 2.2.1\n"); closure\_graf\_spisok(x, n, graf); printf("\n\n"); \_getch();

//printf("\nЗадание 2.2.2\n"); delete\_vertex\_graf\_spisok(n, x, graf); printf("\n\n"); \_getch(); //printf("\nЗадание 2.2.2\n"); delete\_vertex\_graf\_spisok(x, n, graf); printf("\n\n"); \_getch();

//printf("\nЗадание 2.2.3\n"); split\_graf\_spisok(n, x, graf); printf("\n\n"); \_getch(); //printf("\nЗадание 2.2.3\n"); split\_graf\_spisok(x, n, graf); printf("\n\n"); \_getch();

//printf("\nЗадание 3.1\n"); union\_graf(t, r, n, x); printf("\n\n"); \_getch();

//printf("\nЗадание 3.2\n"); crossing\_graf(t, r, n, x); printf("\n\n"); \_getch();

//printf("\nЗадание 3.3\n"); annular\_sum\_graf(t, r, n, x); printf("\n\n"); \_getch();

printf("\nЗадание 4\n"); decart\_graf(t, r, n, x); printf("\n\n"); \_getch();

}