Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

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

на тему «унарные и бинарные операции над графами»

**Выполнил:**

студент группы 20ВВ3

Шмелёв Д.В.

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

# Унарные и бинарные операции над графами.

# Лабораторное задание

**Задание 1**

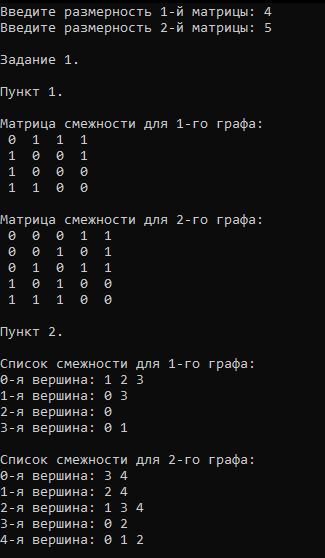
1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.
2. Определите радиус и диаметр графа *G*, используя матрицу смежности графа.
3. Определите подмножества периферийных и центральных вершин графа *G*, используя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

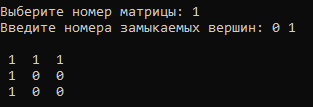
# Результат работы программы

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



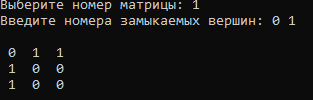
**Рисунок 1 – Результат работы программы**

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



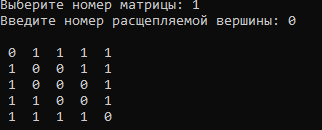
**Рисунок 2 – Результат работы программы**

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



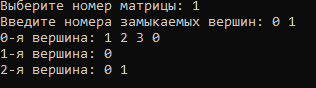
**Рисунок 3 – Результат работы программы**

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



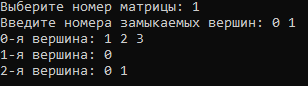
**Рисунок 4 – Результат работы программы**

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



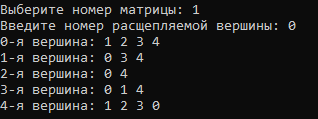
**Рисунок 5 – Результат работы программы**

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



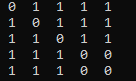
**Рисунок 6 – Результат работы программы**

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



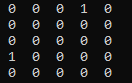
**Рисунок 7 – Результат работы программы**

**Задание 3.1.** Операция объединения.



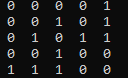
**Рисунок 8 – Результат работы программы**

**Задание 3.2.** Операция пересечения.



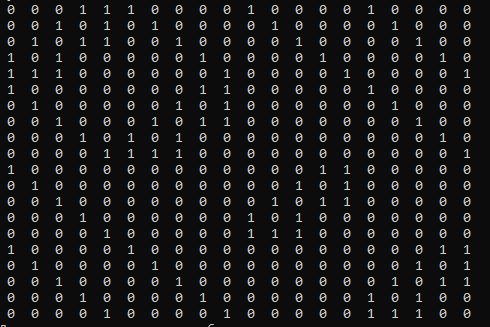
**Рисунок 9 – Результат работы программы**

**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\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#define HEADER ("Лабораторная работа №3\nВыполнили: Шмелёв Д. и Пантюшов Е.\n\n")

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

#include <queue>

#include <stack>

using namespace std;

struct node

{

int v;

node\* next;

};

struct graph

{

int n;

node\*\* adjList;

};

node\* createnode(int v)

{

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

newnode->v = v;

newnode->next = NULL;

return newnode;

}

void addedge(graph\* G, int i, int j)//i = 0 j = 4

{

node\* newNode = createnode(j);

if (G->adjList[i] == NULL)

{

G->adjList[i] = newNode;

newNode = NULL;

}

node\* temp = G->adjList[i];// int j = int k int j = int\* k

while (temp->next)

{

temp = temp->next;

}

temp->next = newNode;

newNode = createnode(i);

if (G->adjList[j] == 0)

{

G->adjList[j] = newNode;

newNode = NULL;

}

temp = G->adjList[j];

while (temp->next)

{

temp = temp->next;

}

temp->next = newNode;

}

graph\* creategraph(int\*\* M, int n)

{

graph\* G = (graph\*)(malloc(sizeof(graph)));

G->n = n;

G->adjList = (node\*\*)(malloc(n \* sizeof(node\*)));

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

G->adjList[i] = NULL;

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

{

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

{

if ((M[i][j] == 1) && (i < j))

{

addedge(G, i, j);

}

}

}

return G;

}

void printgraph(graph\* G)

{

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

{

node\* temp = G->adjList[i];

printf("%d-я вершина: ", i);

while (temp)

{

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

temp = temp->next;

}

printf("\n");

}

}

int\*\* creatematrix(int n)

{

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

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

{

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

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

{

M[i][j] = rand() % 100;

if (M[i][j] < 40)

M[i][j] = 0;

else

M[i][j] = 1;

}

}

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

{

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

{

if (i == j)

M[i][j] = 0;

else

M[i][j] = M[j][i];

}

}

return M;

}

void f1(int\*\* M1, int\*\* M2, int n1, int n2)

{

printf("\nЗадание 1.\n\nПункт 1.\n");

printf("\nМатрица смежности для 1-го графа:\n");

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

{

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

{

printf(" %d ", M1[i][j]);

}

printf("\n");

}

printf("\nМатрица смежности для 2-го графа:\n");

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

{

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

{

printf(" %d ", M2[i][j]);

}

printf("\n");

}

}

void f2(graph\* G1, graph\* G2)

{

printf("\nПункт 2.\n\nСписок смежности для 1-го графа:\n");

printgraph(G1);

printf("\nСписок смежности для 2-го графа:\n");

printgraph(G2);

}

void IOV(int\*\* A, int n)

{

if (n < 2)

{

printf("Данная операция невозможна!");

return;

}

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

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

{

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

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

{

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

}

}

int k, l, r = 0;

printf("Введите номера замыкаемых вершин: ");

scanf("%d %d", &k, &l);

if (M[k][l])

r = 1;

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

if (M[l][i])

M[k][i] = M[l][i];

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

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

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

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

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

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

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

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

free(M[n - 1]);

n = n - 1;

if (r)

{

M[k][k] = 1;

}

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

{

if (M[k][i] > 0)

{

M[i][k] = 1;

}

}

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

{

printf("\n");

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

{

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

}

}

}

void IOV\_list(graph\* A, int n)

{

graph\* G = (graph\*)(malloc(sizeof(graph)));

G->n = n;

G->adjList = (node\*\*)(malloc(n \* sizeof(node\*)));

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

G->adjList[i] = NULL;

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

{

node\* temp = A->adjList[i];

G->adjList[i] = (node\*)(malloc(n \* sizeof(node)));

node\* temp1 = G->adjList[i];

if (temp)

{

temp1->v = temp->v;

temp = temp->next;

temp1->next = NULL;

}

else

{

G->adjList[i] = NULL;

temp1 = G->adjList[i];

}

while (temp)

{

temp1->next = (node\*)(malloc(sizeof(node)));

temp1 = temp1->next;

temp1->v = temp->v;

temp1->next = NULL;

temp = temp->next;

}

}

int k, l, r = 0;

printf("Введите номера замыкаемых вершин: ");

scanf("%d %d", &k, &l);

node\* temp1 = G->adjList[k];

node\* temp2 = G->adjList[l];

node\* temp3 = G->adjList[k];

node\* temp4 = temp1;

if (temp3)

{

while (temp3->next)

{

temp3 = temp3->next;

}

while (temp2)

{

temp4 = temp1;

while (temp1)

{

if (temp1->v == temp2->v)

{

r++;

}

temp1 = temp1->next;

}

temp1 = temp4;

if (r == NULL)

{

temp3->next = (node\*)(malloc(sizeof(node)));

temp3 = temp3->next;

temp3->v = temp2->v;

temp3->next = NULL;

}

temp2 = temp2->next;

r = 0;

}

}

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

{

G->adjList[i] = G->adjList[i + 1];

}

G->n--;

printgraph(G);

}

void RR(int\*\* A, int n)

{

if (n < 2)

{

printf("Данная операция невозможна!");

return;

}

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

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

{

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

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

{

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

}

}

int k, l;

printf("Введите номера замыкаемых вершин: ");

scanf("%d %d", &k, &l);

if (M[k][l] == 0)

{

printf("Вершины не связаны!");

return;

}

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

if (M[l][i])

M[k][i] = M[l][i];

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

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

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

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

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

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

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

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

free(M[n - 1]);

n = n - 1;

M[k][k] = 0;

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

{

if (M[k][i] > 0)

{

M[i][k] = 1;

}

}

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

{

printf("\n");

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

{

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

}

}

}

void printMatrix(int\*\* M, int n)

{

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

{

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

{

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

}

printf("\n");

}

}

void RR\_list(graph\* A, int n)

{

graph\* G = (graph\*)(malloc(sizeof(graph)));

G->n = n;

G->adjList = (node\*\*)(malloc(n \* sizeof(node\*)));

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

G->adjList[i] = NULL;

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

{

node\* temp = A->adjList[i];

G->adjList[i] = (node\*)(malloc(n \* sizeof(node)));

node\* temp1 = G->adjList[i];

if (temp)

{

temp1->v = temp->v;

temp = temp->next;

temp1->next = NULL;

}

else

{

G->adjList[i] = NULL;

temp1 = G->adjList[i];

}

while (temp)

{

temp1->next = (node\*)(malloc(sizeof(node)));

temp1 = temp1->next;

temp1->v = temp->v;

temp1->next = NULL;

temp = temp->next;

}

}

int k, l, r = 0;

printf("Введите номера замыкаемых вершин: ");

scanf("%d %d", &k, &l);

node\* temp1 = G->adjList[k];

node\* temp2 = G->adjList[l];

node\* temp3 = G->adjList[k];

node\* temp4 = temp1;

int e = 0;

while (temp1->next)

{

if (temp1->v == l)

{

e = 1;

}

temp1 = temp1->next;

}

if (e == 0)

{

printf("Ошибка!\n");

return;

}

if (temp3)

{

while (temp3->next)

{

temp3 = temp3->next;

}

while (temp2)

{

temp4 = temp1;

while (temp1)

{

if (temp1->v == temp2->v || temp1->v != l)

{

r++;

}

temp1 = temp1->next;

}

temp1 = temp4;

if (r == NULL)

{

temp3->next = (node\*)(malloc(sizeof(node)));

temp3 = temp3->next;

temp3->v = temp2->v;

temp3->next = NULL;

}

temp2 = temp2->next;

r = 0;

}

}

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

{

G->adjList[i] = G->adjList[i + 1];

}

G->n--;

printgraph(G);

}

void SV(int\*\* A, int n)

{

int k;

printf("Введите номер расщепляемой вершины: ");

scanf("%d", &k);

n++;

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

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

{

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

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

{

if (i < n - 1 && j < n - 1)

{

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

}

else

{

M[i][j] = 0;

}

}

}

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

if (M[k][i])

{

M[n - 1][i] = M[k][i];

M[i][n - 1] = M[i][k];

}

M[n - 1][k] = 1;

M[k][n - 1] = 1;

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

{

if (M[n - 1][i] > 0)

{

M[i][n - 1] = 1;

}

}

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

{

printf("\n");

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

{

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

}

}

}

void SV\_list(graph\* A, int n)

{

graph\* G = (graph\*)(malloc(sizeof(graph)));

G->n = n + 1;

G->adjList = (node\*\*)(malloc((n + 1) \* sizeof(node\*)));

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

G->adjList[i] = NULL;

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

{

node\* temp = A->adjList[i];

G->adjList[i] = (node\*)(malloc(n \* sizeof(node)));

node\* temp1 = G->adjList[i];

if (temp)

{

temp1->v = temp->v;

temp = temp->next;

temp1->next = NULL;

}

else

{

G->adjList[i] = NULL;

temp1 = G->adjList[i];

}

while (temp)

{

temp1->next = (node\*)(malloc(sizeof(node)));

temp1 = temp1->next;

temp1->v = temp->v;

temp1->next = NULL;

temp = temp->next;

}

}

int k;

printf("Введите номер расщепляемой вершины: ");

scanf("%d", &k);

G->adjList[n] = (node\*)(malloc(sizeof(node)));

node\* temp1 = G->adjList[k];

node\* temp2 = G->adjList[n];

if (temp1)

{

temp2->v = temp1->v;

temp2->next = NULL;

temp1 = temp1->next;

}

else

{

G->adjList[n] = NULL;

temp2 = G->adjList[n];

}

while (temp1)

{

temp2->next = (node\*)(malloc(sizeof(node)));

temp2 = temp2->next;

temp2->v = temp1->v;

temp2->next = NULL;

temp1 = temp1->next;

}

temp1 = G->adjList[k];

if (temp1)

{

while (temp1->next)

{

temp1 = temp1->next;

}

temp1->next = (node\*)(malloc(sizeof(node)));

temp1 = temp1->next;

temp1->v = n;

temp1->next = NULL;

temp2 = G->adjList[n];

while (temp2->next)

{

temp2 = temp2->next;

}

temp2->next = (node\*)(malloc(sizeof(node)));

temp2 = temp2->next;

temp2->v = k;

temp2->next = NULL;

}

temp2 = G->adjList[k];

node\* temp3 = temp2;

node\* temp4 = G->adjList[n];

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

{

temp1 = G->adjList[i];

while (temp1)

{

if (temp1->v == k)

{

while (temp1->next)

{

temp1 = temp1->next;

}

temp1->next = (node\*)(malloc(sizeof(node)));

temp1 = temp1->next;

temp1->v = n;

temp1->next = NULL;

}

temp1 = temp1->next;

}

}

if (G->adjList[k] == NULL)

{

G->adjList[k] = (node\*)(malloc(sizeof(node)));

G->adjList[n] = (node\*)(malloc(sizeof(node)));

G->adjList[k]->v = n;

G->adjList[n]->v = k;

G->adjList[k]->next = NULL;

G->adjList[n]->next = NULL;

}

printgraph(G);

}

void f3(int\*\* M1, int\*\* M2, int n1, int n2)

{

int n;

printf("\nЗадание 2.\n\nПункт 1.\n\n1)отождествления вершин\n2)стягивания ребра\n3)расщепления вершины\n4)продолжить");

do

{

printf("\n\n->");

scanf("%d", &n);

switch (n)

{

case 1:

int k;

printf("Выберите номер матрицы: ");

scanf("%d", &k);

if (k == 1)

{

IOV(M1, n1);

}

else

IOV(M2, n2);

break;

case 2:

printf("Выберите номер матрицы: ");

scanf("%d", &k);

if (k == 1)

RR(M1, n1);

else

RR(M2, n2);

break;

case 3:

printf("Выберите номер матрицы: ");

scanf("%d", &k);

if (k == 1)

SV(M1, n1);

else

SV(M2, n2);

break;

case 4:

return;

default:

printf("Error!");

}

} while (n != 4);

}

void f4(graph\* G1, graph\* G2, int n1, int n2)

{

int n;

printf("\nЗадание 2.\n\nПункт 1.\n\n1)отождествления вершин\n2)стягивания ребра\n3)расщепления вершины\n4)продолжить");

do

{

printf("\n\n->");

scanf("%d", &n);

switch (n)

{

case 1:

int k;

printf("Выберите номер матрицы: ");

scanf("%d", &k);

if (k == 1)

IOV\_list(G1, n1);

else

IOV\_list(G2, n2);

break;

case 2:

printf("Выберите номер матрицы: ");

scanf("%d", &k);

if (k == 1)

RR\_list(G1, n1);

else

RR\_list(G2, n2);

break;

case 3:

printf("Выберите номер матрицы: ");

scanf("%d", &k);

if (k == 1)

SV\_list(G1, n1);

else

SV\_list(G2, n2);

break;

case 4:

return;

default:

printf("Error!");

}

} while (n != 4);

}

void f5(int\*\* M1, int\*\* M2, int n1, int n2)

{

printf("Задание 3.\nПункт 1.\n");

int n, m;

if (n1 >= n2)

{

n = n1;

}

if (n2 >= n1)

{

n = n2;

}

if (n1 <= n2)

{

m = n1;

}

if (n2 <= n1)

{

m = n2;

}

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

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

{

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

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

{

M[i][j] = 0;

}

}

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

{

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

{

if (M1[i][j] > 0)

{

M[i][j] = 1;

}

}

}

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

{

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

{

if (M2[i][j] > 0)

{

M[i][j] = 1;

}

}

}

printMatrix(M, n);

printf("Пункт 2.\n");

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

{

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

{

M[i][j] = 0;

}

}

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

{

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

{

if (M1[i][j] > 0 && M2[i][j] > 0)

{

M[i][j] = 1;

}

}

}

printMatrix(M, n);

printf("Пункт 3.\n");

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

{

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

{

M[i][j] = 0;

}

}

int r = 0;

if (n == n1)

{

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

{

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

{

M[i][j] = M1[i][j];

}

}

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

{

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

{

if (M1[i][j] > 0 && M2[i][j] > 0)

{

M[i][j] = 0;

}

}

}

}

else

{

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

{

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

{

M[i][j] = M2[i][j];

}

}

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

{

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

{

if (M1[i][j] > 0 && M2[i][j] > 0)

{

M[i][j] = 0;

}

}

}

}

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

{

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

{

if (M[k][j] > 0)

r++;

}

if (r == 0)

{

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

{

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

{

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

}

}

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

{

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

{

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

}

}

n--;

k--;

}

r = 0;

}

printMatrix(M, n);

}

void f6(int\*\* M1, int\*\* M2, int n1, int n2)

{

printf("Пункт 4.\n");

int n = n1 \* n2;

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

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

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

int w = 0;

int h = -1;

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

{

for (int k = 0; k < n2; k++)

{

h++;

w = 0;

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

{

for (int m = 0; m < n2; m++)

{

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

{

M[h][w] = 0;

}

else if (i == j)

{

M[h][w] = M2[k][m];

}

else if (k == m)

{

M[h][w] = M1[i][j];

}

else

{

M[h][w] = 0;

}

w++;

}

}

}

}

printMatrix(M, n);

}

int main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf(HEADER);

int n1, n2;

printf("Введите размерность 1-й матрицы: ");

scanf("%d", &n1);

printf("Введите размерность 2-й матрицы: ");

scanf("%d", &n2);

int\*\* M1 = creatematrix(n1);

int\*\* M2 = creatematrix(n2);

graph\* G1 = creategraph(M1, n1);

graph\* G2 = creategraph(M2, n2);

f1(M1, M2, n1, n2);

f2(G1, G2);

f3(M1, M2, n1, n2);

f4(G1, G2, n1, n2);

f5(M1, M2, n1, n2);

f6(M1, M2, n1, n2);

system("PAUSE");

return 0;

}