Пензенский государственный университет  
Кафедра «Вычислительной техники»

## 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 19ВВ1:**

Тебнев Р.  
Балалаев А.

**Приняли:**

д.т.н. профессор Митрохин М. А.

Юрова О.В.

Пенза 2020

**Цель работы**: изучить унарные и бинарные операции над графами и реализовать их в программе.

**Теоретическая часть:**

**Замыкание или отождествление.**

Пара вершин хi и xj в графе G замыкается (или отождествляется), если они заменяются такой новой вершиной, что все дуги в графе G, инцидентные хi и xj , становятся инцидентными новой вершине.При этом ребро, соединяющее вершины хi и xj становится петлёй. Матрица смежности графа после выполнения операции замыкания вершин хi и xj получается путем поэлементного логического сложения i - го и j – го столбцов и i -ой и j - строк в исходной матрице.

**Стягивание.**

Под стягиванием подразумевают операцию удаления дуги или ребра и отождествление его концевых вершин.

**Расщепление вершины.**

В графе G1 выделяется вершина ν. Окружение Q вершины ν разбивается на две части М, N (Mhttps://cde.osu.ru/courses2/temp4/images/und.gifN=Q, Mhttps://cde.osu.ru/courses2/temp4/images/sh.gifN=ø). После этого над графом G1 выполняются следующие преобразования:

1. удаляют вершину ν вместе с инцидентными ей ребрами (Н1=G1=ν-νzi, zihttps://cde.osu.ru/courses2/temp4/images/in.gifQ, i=1,2,3 ..);
2. добавляют новые вершины u,w и соединяющее их ребро uw(H2=H1+u+w+uw);
3. вершину u соединяют ребром с каждой вершиной из множества М(H3=H1+uzi, zihttps://cde.osu.ru/courses2/temp4/images/in.gifM);
4. вершину w соединяют ребром с каждой вершиной из множества N(G2=H3+uzi, zihttps://cde.osu.ru/courses2/temp4/images/in.gifM).

Говорят, что граф G2 получен из графа G1 расщеплением вершины ν.

**Объединение графов** G1 и G2, обозначаемое как G1 U G2, представляет собой граф G3 = (X1 U X2, A1 U A2) такой, что множество его вершин является объединением Х1 и Х2, а множество ребер – объединением A1 и A2 .

Матрица смежности результирующего графа G3 получается операцией поэлементного логического сложения матриц смежности исходных графов G1 и G2 .

**Пересечение графов** G1 и G2 , обозначаемое как G1 ∩ G2, представляет собой граф G3 = (X1 ∩ X2, A1 ∩ A2) такой, что множество его вершин и дуг состоит из вершин и дуг, присутствующих одновременно в G1 и G2.

Результирующая матрица смежности получается операцией поэлементного логического умножения матриц смежности исходных графов G1 и G2.

**Кольцевая сумма** графов G1 и G2, обозначаемая как G1 ⊕ G2, представляет собой граф, порожденный на множестве ребер A1 ⊕ A2.

Т.е. новый граф не имеет изолированных вершин и состоит только из ребер, присутствующих либо в G1 , либо в G2 , но не в обоих одновременно.

Результирующая матрица смежности получается операцией поэлементного логического сложения по mod 2 матриц смежности исходных графов G1 и G2 .

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <locale.h>

#include <queue>

using namespace std;

void NewGraph(int\*\* G, int n) {

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

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

if (i == j) {

G[i][j] = 0;

}

if (i < j) {

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

G[j][i] = G[i][j];

}

}

}

}

void print\_G(int\*\* G, int n) {

printf(" ");

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

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

}

printf("\n\n");

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

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

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

printf("%3d", G[i][j]);

}

printf("\n");

}

printf("\n");

}

void otojd(int v1, int v2, int\*\* G, int n) {

int x1, x2;

int\*\* mat;

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

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

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

}

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

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

mat[i][j] = G[i][j];

}

}

if (v1 > v2) {

x1 = v1 - 1;

x2 = v2 - 1;

}

else {

x1 = v2 - 1;

x2 = v1 - 1;

}

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

if (mat[x1][i] == 1)

mat[x2][i] = 1;

if (mat[i][x1] == 1)

mat[i][x2] = 1;

}

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

for (int j = x1; j < n - 1; j++) {

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

}

}

for (int i = x1; i < n - 1; i++) {

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

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

}

}

print\_G(mat, n - 1);

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

free(mat[i]);

}

free(mat);

}

void obed(int\*\*G1, int\*\* G2, int n) {

int\*\* mat1, \*\* mat2, \*\*res;

mat1 = (int\*\*)malloc(n \* sizeof(int\*));

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

mat1[i] = (int\*)malloc(n \* sizeof(int));

}

mat2 = (int\*\*)malloc(n \* sizeof(int\*));

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

mat2[i] = (int\*)malloc(n \* sizeof(int));

}

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

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

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

}

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

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

mat1[i][j] = G1[i][j];

mat2[i][j] = G2[i][j];

}

}

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

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

res[i][j] = mat1[i][j] || mat2[i][j];

}

}

printf("Результат объединения графов: \n");

print\_G(res, n);

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

free(mat1[i]);

}

free(mat1);

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

free(mat2[i]);

}

free(mat2);

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

free(res[i]);

}

free(res);

}

void perese4(int\*\* G1, int\*\* G2, int n) {

int\*\* mat1, \*\* mat2, \*\* res;

mat1 = (int\*\*)malloc(n \* sizeof(int\*));

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

mat1[i] = (int\*)malloc(n \* sizeof(int));

}

mat2 = (int\*\*)malloc(n \* sizeof(int\*));

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

mat2[i] = (int\*)malloc(n \* sizeof(int));

}

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

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

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

}

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

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

mat1[i][j] = G1[i][j];

mat2[i][j] = G2[i][j];

}

}

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

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

if (mat1[i][j] == 1 && mat2[i][j] == 1) {

res[i][j] = 1;

}

else {

res[i][j] = 0;

}

}

}

printf("Результат пересечения графов: \n");

print\_G(res, n);

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

free(mat1[i]);

}

free(mat1);

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

free(mat2[i]);

}

free(mat2);

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

free(res[i]);

}

free(res);

}

void kolcsumm(int\*\* G1, int\*\* G2, int n) {

int\*\* mat1, \*\* mat2, \*\* res;

mat1 = (int\*\*)malloc(n \* sizeof(int\*));

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

mat1[i] = (int\*)malloc(n \* sizeof(int));

}

mat2 = (int\*\*)malloc(n \* sizeof(int\*));

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

mat2[i] = (int\*)malloc(n \* sizeof(int));

}

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

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

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

}

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

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

mat1[i][j] = G1[i][j];

mat2[i][j] = G2[i][j];

}

}

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

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

if ((mat1[i][j] == 1 && mat2[i][j] == 1) || (mat1[i][j] == 0 && mat2[i][j] == 0)) {

res[i][j] = 0;

}

if ((mat1[i][j] == 1 && mat2[i][j] == 0) || (mat1[i][j] == 0 && mat2[i][j] == 1)) {

res[i][j] = 1;

}

}

}

printf("Кольцевая сумма графов: \n");

print\_G(res, n);

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

free(mat1[i]);

}

free(mat1);

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

free(mat2[i]);

}

free(mat2);

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

free(res[i]);

}

free(res);

}

int main() {

int\*\* G1 = NULL, \*\* G2 = NULL;;

int x1 = NULL, x2 = NULL;

int n;

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf("Введите размер графа: ");

scanf("%d", &n);

G1 = (int\*\*)malloc(n \* sizeof(int\*));

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

G1[i] = (int\*)malloc(n \* sizeof(int));

}

G2 = (int\*\*)malloc(n \* sizeof(int\*));

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

G2[i] = (int\*)malloc(n \* sizeof(int));

}

NewGraph(G1, n);

print\_G(G1, n);

NewGraph(G2, n);

print\_G(G2, n);

printf("Введите 2 вершины 1 графа для отождествления: ");

scanf("%d %d", &x1, &x2);

otojd(x1, x2, G1, n);

printf("Введите 2 вершины 2 графа для отождествления: ");

scanf("%d %d", &x1, &x2);

otojd(x1, x2, G2, n);

printf("\nВВедите 2 смежные вершины в матрице 1 для стягивания: ");

for (;;) {

scanf\_s("%d", &x1);

scanf\_s("%d", &x2);

if (G1[x1 - 1][x2 - 1] == 1)

break;

else

printf("Ошибка, введите заново: ");

}

otojd(x1, x2, G1, n);

printf("\nВведите 2 смежные вершины в матрице 2 для стягивания: ");

for (;;) {

scanf\_s("%d", &x1);

scanf\_s("%d", &x2);

if (G2[x1 - 1][x2 - 1] == 1)

break;

else

printf("Ошибка, введите заново: ");

}

otojd(x1, x2, G2, n);

obed(G1, G2, n);

perese4(G1, G2, n);

kolcsumm(G1, G2, n);

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

free(G1[i]);

}

free(G1);

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

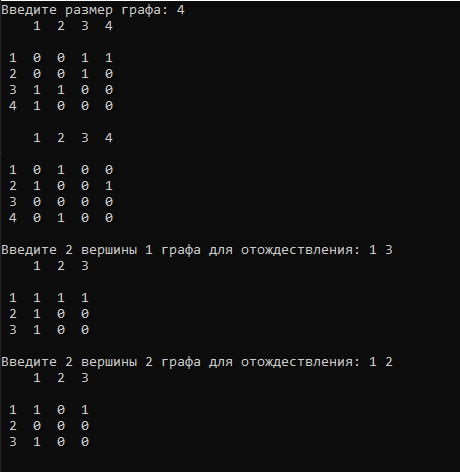
free(G2[i]);

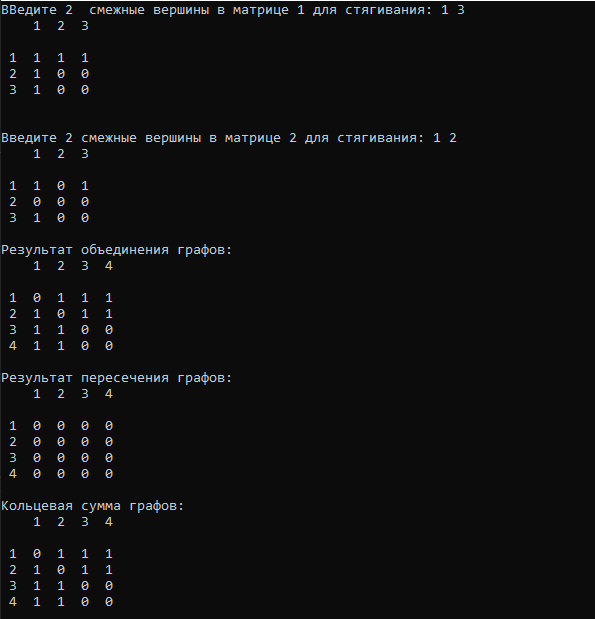
}

free(G2);

}

**Результат работы программы:**





**Вывод:** изучил и реализовал унарные и бинарные операции над графами.