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:

Студенты группы 23ВВВ2

Герасимов В.Р.

Мадамкин В. М.

Приняли:

Юрова О. В.

Митрохин М. А.

Пенза 2024

**Лабораторное задание.**

**Задание 1**:

1. Сгенерируйте (используя генератор случайных чисел) две матрицы M1, М2 смежности неориентированных помеченных графов G1, G2. 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.

**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.

**Приложение A**

**Lab6**

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

int\*\* createG(int size) {

int n = 10;

int\*\* G = NULL;

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

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

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

}

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

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

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

if (i == j) {

G[i][j] = 0;

}

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

}

}

return G;

}

void printG(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

return;

}

void listing(int\*\* G, int size) {

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

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

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

printf("%d ", j);

}

}

printf("\n");

}

}

int\*\* delV(int\*\* G, int size, int v) {

int\*\* Gtemp = createG(size - 1);

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

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

if (i < v && j < v) {

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

}

if (i<v && j>v) {

Gtemp[i][j - 1] = G[i][j];

}

if (i > v && j < v) {

Gtemp[i - 1][j] = G[i][j];

}

if (i > v && j > v) {

Gtemp[i - 1][j - 1] = G[i][j];

}

}

}

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

free(G[i]);

}

free(G);

G = NULL;

return Gtemp;

}

int\*\* unionV(int\*\* G, int size, int v1, int v2) {

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

if (G[v2][i] == 1) {

G[v1][i] = G[v2][i];

G[i][v1] = G[i][v2];

}

}

G = delV(G, size, v2);

return G;

}

int\*\* contrE(int\*\* G, int size, int v1, int v2) {

if (G[v1][v2] == 0) {

printf("rebra net\n");

return G;

}

else {

G[v1][v2] = 0;

G[v2][v1] = 0;

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

if (G[v2][i] == 1) {

G[v1][i] = G[v2][i];

G[i][v1] = G[i][v2];

}

}

G = delV(G, size, v2);

}

return G;

}

int\*\* splitV(int\*\* G, int size, int V) {

int\*\* Gtemp = createG(size + 1);

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

Gtemp[i][size + 1] = G[i][V];

Gtemp[size + 1][i] = G[V][i];

}

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

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

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

}

}

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

free(G[i]);

}

free(G);

G = NULL;

return Gtemp;

}

int\*\* unionGraphs(int\*\* G1, int\*\* G2, int size) {

int\*\* result = (int\*\*)malloc(size \* sizeof(int\*));

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

result[i] = (int\*)malloc(size \* sizeof(int));

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

result[i][j] = G1[i][j] || G2[i][j]; // Объединение

}

}

return result;

}

int\*\* intersectionGraphs(int\*\* G1, int\*\* G2, int size) {

int\*\* result = (int\*\*)malloc(size \* sizeof(int\*));

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

result[i] = (int\*)malloc(size \* sizeof(int));

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

result[i][j] = G1[i][j] && G2[i][j]; // Пересечение

}

}

return result;

}

int\*\* ringSumGraphs(int\*\* G1, int\*\* G2, int size) {

int\*\* result = (int\*\*)malloc(size \* sizeof(int\*));

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

result[i] = (int\*)malloc(size \* sizeof(int));

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

result[i][j] = (G1[i][j] + G2[i][j]) % 2; // Кольцевая сумма

}

}

return result;

}

int main()

{

setlocale(LC\_ALL, "Rus");

int r, v1, v2;

printf("Введите размер матриц \n");

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

int\*\* G1 = createG(r);

int\*\* G2 = createG(r);

int\*\* G3 = createG(r);

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

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

G3[i][j] = G2[i][j];

}

}

int\*\* G4 = createG(r);

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

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

G4[i][j] = G2[i][j];

}

}

int\*\* G5 = createG(r);

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

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

G5[i][j] = G2[i][j];

}

};

//дописать

int\*\* unionResult = unionGraphs(G1, G2, r);

int\*\* intersectionResult = intersectionGraphs(G1, G2, r);

int\*\* ringSumResult = ringSumGraphs(G1, G2, r);

int N=1;

while (N != 0) {

printf("0.Выход\n");

printf("1.Вывод в виде матриц смежности\n");

printf("2.Вывод в виде списка смежности\n");

printf("3.Стягивание\n");

printf("4.Расщепление\n");

printf("5.Отождествление\n");

printf("6.Объединение\n");

printf("7.Пересечение\n");

printf("8.Кольцевая сумма\n");

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

switch (N) {

case 0:

printf("До свидания!\n");

break;

case 1://вывод матр

printf("Матрица 1\n");

printG(G1, r);

printf("Матрица 2\n");

printG(G2, r);

break;

case 2://вывод список

printf("Матрица 1\n");

listing(G1, r);

printf("Матрица 2\n");

listing(G2, r);

break;

case 3://стягивание

printf("Введите вершины v1 v2\n");

scanf\_s("%d %d", &v1, &v2);

G4 = contrE(G4, r - 1, v1, v2);

printG(G4, r - 1);

break;

case 4://расщепление

printf("Введите вершины v1\n");

scanf\_s("%d", &v1);

G5 = splitV(G5, r + 1, v1);

printG(G5, r + 1);

break;

case 5://отождествление

printf("Введите вершины v1 v2\n");

scanf\_s("%d %d", &v1, &v2);

G3 = unionV(G3, r - 1, v1, v2);

printG(G3, r - 1);

break;

case 6://объединение-элемент равен 1, если хотя бы один из графов имеет ребро между соответствующими вершинами.

printG(unionResult, r);

break;

case 7://пересечение-элемент равен 1, только если оба графа имеют ребро между соответствующими вершинами

printG(intersectionResult, r);

break;

case 8://кольцевая сумма-элемент равен 1, если сумма значений в соответствующих ячейках двух графов нечетная

printG(ringSumResult, r);

break;

default:

printf("Неправильный выбор\n");

}

}

return 0;

}