Пензенский государственный университет Кафедра «Вычислительная техника»

**Отчет**о выполнении лабораторной работы №3.1 по дисциплине “Логика и основа алгоритмизации в инженерных задачах” на тему**:**

**Унарные и бинарные операции над графами**

Выполнили студенты гр. 19ВВ4:   
Хлыстов А. Ю.  
Привалов А. Э.

Проверили:   
Юрова О. В.  
Митрохин М. А.

Пенза, 2020 г.

**Название**Унарные и бинарные операции над графами

**Цель работы**Научиться реализовывать унарные и бинарные операции над графами в программе.

**Лабораторное задание  
  
Задание 1**: Сгенерируйте (используя генератор случайных чисел) две матрицы M1, М2 смежности неориентированных помеченных графов G1, G2. Выведите сгенерированные матрицы на экран.

**Задание 2**: 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**: Для матричной формы представления графов выполните операцию:

а) объединения

б) пересечения

в) кольцевой суммы

Результат выполнения операции выведите на экран.

**Код программы**

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

struct graph {

int\*\* matrix;

int\* names;

int size;

}m1, m2, result;

void outputMatrix(graph m) {

printf\_s(" ");

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

printf\_s("%d ", m.names[i]);

}

printf\_s("\n");

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

printf\_s("%d ", m.names[i]);

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

printf\_s("%d ", m.matrix[i][j]);

}

printf\_s("\n");

}

}

//Заполнение массива

void fillMatrix(graph m) {

int value;

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

for (int j = i + 1; j < m.size; j++) {

value = rand() % 10;

if (value < 3)

value = 0;

else

value = 1;

m.matrix[i][j] = value;

m.matrix[j][i] = m.matrix[i][j];

}

}

}

//Создание двумерного массива

int\*\* createMatrix(int size) {

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

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

matrix[i] = (int\*)calloc(size, sizeof(int));

}

return matrix;

}

//Удаление двумерного массива

void deleteMatrix(graph m) {

for (int i = 0; i < m.size - 1; i++) {

free(m.matrix[i]);

}

free(m.matrix);

}

//Удаление вершины графа

graph deleteVertex(int vertex, graph m) {

int\*\* new\_matr = createMatrix(m.size);

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

for (int j = i + 1; j < m.size; j++) {

if (i < vertex - 1 && j < vertex - 1) {

new\_matr[i][j] = m.matrix[i][j];

new\_matr[j][i] = new\_matr[i][j];

}

if (i < vertex - 1 && j >= vertex - 1) {

new\_matr[i][j] = m.matrix[i][j + 1];

new\_matr[j][i] = new\_matr[i][j];

}

if (i >= vertex - 1 && j >= vertex - 1) {

new\_matr[i][j] = m.matrix[i + 1][j + 1];

new\_matr[j][i] = new\_matr[i][j];

}

}

}

//Редактируем массив имён

for (int i = vertex - 1; i < m.size -1; i++) {

m.names[i] = m.names[i + 1];

}

m.size--;

m.matrix = new\_matr;

return m;

}

//Соединение столбцов \ строк матрицы смежности

int\* connectionVertex(int\* vertex1, int\* vertex2, graph m) {

int\* result = (int\*)calloc(m.size, sizeof(int));

for (int i = 0; i < m.size - 1; i++) {

if (vertex1[i] == 1 || vertex2[i] == 1) {

result[i] = 1;

}

}

free(vertex1);

free(vertex2);

return result;

}

//Извлечение значений строк \ cтолбцов матрицы

int\* extractionVertex(graph m, int vertex) {

int\* vert\_matr = (int\*)calloc(m.size, sizeof(int));

for (int i = 0; i < m.size - 1; i++) {

vert\_matr[i] = m.matrix[vertex - 1][i];

}

return vert\_matr;

}

//Добавление вершины графа

graph addVertex(graph m, int vertex, int\* connection, int vertex\_name) {

int\*\* new\_matr = createMatrix(m.size + 1);

int select\_vertex = 0;

for (int i = 0; i < m.size + 1; i++) {

for (int j = i + 1; j < m.size + 1; j++) {

if (i < vertex - 1 && j < vertex - 1) {

new\_matr[i][j] = m.matrix[i][j];

new\_matr[j][i] = new\_matr[i][j];

}

if (i < vertex - 1 && j > vertex - 1) {

new\_matr[i][j] = m.matrix[i][j - 1];

new\_matr[j][i] = new\_matr[i][j];

}

if (i > vertex - 1 && j > vertex - 1) {

new\_matr[i][j] = m.matrix[i - 1][j - 1];

new\_matr[j][i] = new\_matr[i][j];

}

if (i == vertex - 1 || j == vertex - 1) {

new\_matr[i][j] = connection[select\_vertex];

new\_matr[j][i] = new\_matr[i][j];

select\_vertex++;

}

}

}

//Редактируем массив имён

m.names[vertex - 1] = vertex\_name;

for (int i = 0; i < m.size - vertex; i++) {

m.names[m.size - i] = m.names[m.size - i - 1];

}

m.size++;

m.matrix = new\_matr;

return m;

}

//Возвращает значение соединения 2х вершин в матрице

int checkConnect(graph m, int vertex1, int vertex2) {

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

if (vertex1 == m.names[i]) {

vertex1 = i;

break;

}

else if (i == m.size - 1)

return 0;

}

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

if (vertex2 == m.names[i]) {

vertex2 = i;

break;

}

else if (i == m.size - 1)

return 0;

}

return m.matrix[vertex1][vertex2];

}

//Удаляет все рёбра вершины

graph clearVertex(graph m, int vertex) {

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

m.matrix[vertex][i] = 0;

m.matrix[i][vertex] = 0;

}

return m;

}

//Объединение матриц

graph connectMatrix(graph m, graph n) {

graph con;

con = m;

int \* clear\_vertex = (int\*)calloc(m.size + n.size, sizeof(int));

//Добавляем уникальные вершины

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

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

if (m.names[j] == n.names[i]) {

break;

}

else if (j == m.size - 1) {

con = addVertex(con, con.size + 1, clear\_vertex, n.names[i]);

}

}

}

//Добавляем рёбра в рамках первой матрицы

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

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

if (m.names[i] == n.names[j]) {

for (int k = 0; k < m.size; k++) {

if (con.matrix[i][k] == 0)

con.matrix[i][k] = checkConnect(n, m.names[i], m.names[k]);

}

}

}

}

//Добавляем остальные рёбра

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

for (int j = m.size; j < con.size; j++) {

con.matrix[i][j] = checkConnect(n, con.names[i], con.names[j]);

con.matrix[j][i] = con.matrix[i][j];

}

}

free(clear\_vertex);

return con;

}

//Пересечение матриц

graph crossMatrix(graph m, graph n) {

graph cross = m;

int\* clear\_vertex = (int\*)calloc(m.size + n.size, sizeof(int));

//Добавляем уникальные вершины

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

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

if (m.names[j] == n.names[i]) {

break;

}

else if (j == m.size - 1) {

cross = addVertex(cross, cross.size + 1, clear\_vertex, n.names[i]);

}

}

}

//Определяем оставшиеся рёбра

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

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

if (m.names[i] == n.names[j]) {

for (int k = 0; k < m.size; k++) {

if(cross.matrix[i][k] == 1)

cross.matrix[i][k] = checkConnect(n, m.names[i], m.names[k]);

}

break;

}

else if (j == n.size - 1) {

cross = clearVertex(cross, i);

}

}

}

free(clear\_vertex);

return cross;

}

//Кольцевая сумма

graph ringMatrix(graph m, graph n) {

graph ring = m;

int\* clear\_vertex = (int\*)calloc(m.size + n.size, sizeof(int));

int count;

//Добавляем уникальные вершины

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

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

if (m.names[j] == n.names[i]) {

break;

}

else if (j == m.size - 1) {

ring = addVertex(ring, ring.size + 1, clear\_vertex, n.names[i]);

}

}

}

//Редактируем рёбра в рамках первой матрицы

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

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

if (m.names[i] == n.names[j]) {

int count = 0;

for (int k = 0; k < m.size; k++) {

if (ring.matrix[i][k] != checkConnect(n, m.names[i], m.names[k])) {

ring.matrix[i][k] = 1;

count++;

}

else

ring.matrix[i][k] = 0;

}

if (count == 0)

ring = deleteVertex(i + 1, ring);

}

}

}

//Добавляем остальные рёбра

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

for (int j = m.size; j < ring.size; j++) {

ring.matrix[i][j] = checkConnect(n, ring.names[i], ring.names[j]);

ring.matrix[j][i] = ring.matrix[i][j];

}

}

free(clear\_vertex);

return ring;

}

//Проверяем, существует ли указанная вершина

int checkVertex(graph m, int\* vertex) {

int index = 0;

int check\_name = 0;

int name;

while (check\_name == 0) {

if (\*vertex == m.names[index]) {

check\_name = 1;

name = \*vertex;

\*vertex = index + 1;

return name;

break;

}

else if(index == m.size - 1) {

printf\_s("Не существует такой вершины, выберите другую\n");

index = 0;

scanf\_s("%d", vertex);

}

index++;

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

printf\_s("Количество вершин графа 1:");

scanf\_s("%d", &m1.size);

printf\_s("Количество вершин графа 2:");

scanf\_s("%d", &m2.size);

m1.matrix = createMatrix(m1.size);

m2.matrix = createMatrix(m2.size);

m1.names = (int\*)calloc(m1.size, sizeof(int));

m2.names = (int\*)calloc(m2.size, sizeof(int));

for (int i = 0; i < m1.size; i++) {

m1.names[i] = i + 1;

}

for (int i = 0; i < m2.size; i++) {

m2.names[i] = i + 3;

}

fillMatrix(m1);

fillMatrix(m2);

//=============== Задание 2 ===============

int v1, v2, name, cross;

printf\_s("Исходная матрица:\n");

outputMatrix(m1);

//Отождествление вершин

printf\_s("\nОтождествить вершины (две):");

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

name = checkVertex(m1, &v1);

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

checkVertex(m1, &v2);

cross = m1.matrix[v1][v2];

m1 = addVertex(m1, v1, connectionVertex(extractionVertex(m1, v1), extractionVertex(m1, v2), m1), name);

m1 = deleteVertex(v1 + 1, m1);

m1 = deleteVertex(v2, m1);

m1.matrix[v1 - 1][v1 - 1] = cross;

printf\_s("Результат:\n");

outputMatrix(m1);

//Стягивание ребра

printf\_s("\nСтянуть ребро между вершинами (две):\n");

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

name = checkVertex(m1, &v1);

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

checkVertex(m1, &v2);

m1 = addVertex(m1, v1, connectionVertex(extractionVertex(m1, v1), extractionVertex(m1, v2), m1), name);

m1 = deleteVertex(v1 + 1, m1);

m1 = deleteVertex(v2, m1);

printf\_s("Результат:\n");

outputMatrix(m1);

//Расщепление вершины

printf\_s("\nРасщепить вершину:");

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

checkVertex(m1, &v1);

//Выбираем имя для новой вершины

for (int i = 1; i < 100; i++) {

name = m1.names[m1.size - 1] + i;

for (int j = 0; j < m1.size; j++) {

if (name == m1.names[j]) {

name = 0;

break;

}

else if (j == m1.size - 1)

break;

}

if (name != 0)

break;

}

//Делим вершины на 2 части

int clear = m1.size / 2;

//Значения для второй вершины

int\* new\_vertex = (int\*)calloc(m1.size, sizeof(int));

//Разделяем значения из первой вершины

for (int i = 0; i < m1.size; i++) {

if (i >= clear) {

new\_vertex[i] = m1.matrix[v1 - 1][i];

m1.matrix[v1 - 1][i] = 0;

}

}

m1 = addVertex(m1, m1.size + 1, new\_vertex, name);

m1.matrix[v1 - 1][m1.size - 1] = 1;

m1.matrix[m1.size - 1][v1 - 1] = 1;

printf\_s("Результат:\n");

outputMatrix(m1);

printf\_s("\n\n");

//=============== Задание 3 ===============

printf\_s("Задание 3:\n");

printf\_s("Исходные матрицы:\n");

outputMatrix(m1);

printf\_s("\n");

outputMatrix(m2);

printf\_s("\n\n");

//Объединения графов

printf\_s("Объединение грфов:\n");

result = connectMatrix(m1, m2);

outputMatrix(result);

printf\_s("\n");

//Пересечение графов

printf\_s("Пересечение грфов:\n");

result = crossMatrix(m1, m2);

outputMatrix(result);

printf\_s("\n");

//Кольцевая сумма

printf\_s("Кольцевая сумма грфов:\n");

result = ringMatrix(m1, m2);

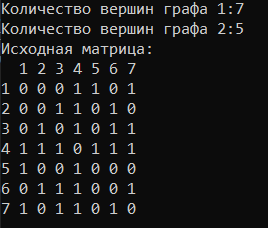
outputMatrix(result);

printf\_s("\n");

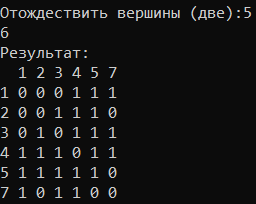
scanf\_s("%d", &v1);

}

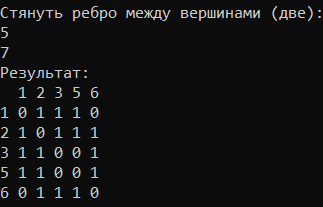
**Результаты работы программы**

****

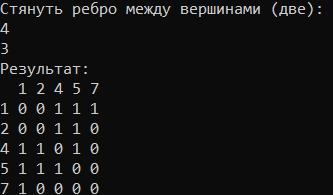
1.Отождествление вершин



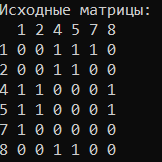
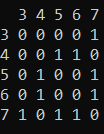
2.Стягивание ребра

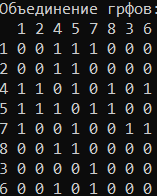


3.Расщепление вершин

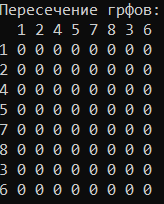


4.Объединение графов

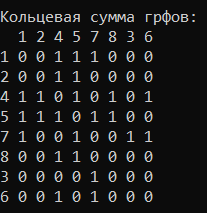
 



5.Пересечение графов



6.Кольцевая сумма графов



**Выводы**В ходе выполнения лабораторной работы была разработана программа, выполняющая унарные и бинарные операции над графами.