Пензенский государственный университет

Кафедра ”Вычислительной техники ”

**Отчёт**

По лабораторной №3

по дисциплине "ЛиОАвИТ"

на тему "Унарные и бинарные операции над графами"

**Выполнили студенты группы 19ВВ1:**

Буданов Д.

**Приняли:**

Митрохин М. А.

Юрова О. В.

Пенза 2020

**Общие сведения.**

Все унарные операции над графами можно объединить в две группы. Первую группу составляют операции, с помощью которых из исходного графа *G*1*,* можно построить граф *G*2 с меньшим числом элементов. В группу входят операции удаления ребра или вершины, отождествления вершин, стягивание ребра. Вторую группу составляют операции, позволяющие строить графы с большим числом элементов. В группу входят операции расщепления вершин, добавления ребра.

*Отождествление вершин.* В графе *G*1 выделяются вершины *и,v.* Определяют окружение *Q*1 вершины *u*,и окружение *Q*2 вершины *v,* вычисляют их объединение *Q* = *Q1* * Q2.* Затем над графом *G*1 выполняются следующие преобразования:

* из графа *G*1 удаляют вершины *u,* *v (H*1 *= G*1 *- u - v);*
* к графу *Н*1присоединяют новую вершину *z (H*1 *= H*1 *+z);*
* вершину *z* соединяют ребром с каждой из вершин *w*1*Q*

*(G*2 *= H*1 *+ zwi*, *i =* 1,2,3*,…).*

*Стягивание ребра.* Данная операция является операцией отождествления смежных вершин *и, v* в графе *G*1.

Наиболее важными бинарными операциями являются: объединение, пересечение, декартово произведение и кольцевая сумма.

*Объединение.* Граф *G* называется объединением или наложением графов *G*1 и *G*2, если *VG = V*1*V*2*; UG = U*1* U*2 (рис. 1).

**U**

*v*1

*v*2

*v*3

*v*4

*v*3

*v*4

*v*5

*v*2

*v*1

*v*3

*v*4

*v*5

Рис. 1. Объединение графов *G*1, *G*2

Объединение графов *G*1 и *G*2 называется дизъюнктным, если *V*1*V*2 *= *. При дизъюнктном объединении никакие два из объединяемых графов не должны иметь общих вершин.

*Пересечение.* Граф *G* называется пересечением графов *G*1, *G*2,если *VG = V*1*V*2и *UG = U*1*U*2 (риc.2). Операция "пересечения" записывается следующим образом: *G = G*1*G*2*.*

**∩**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*6

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

*v*3

*v*2

Рис.2. Пересечение графов *G*1, *G*2*.*

*Декартово произведение.* Граф *G* называется декартовым произведением графов *G*1 и *G*2 если *VG* = *V*1*V*2 —декартово произведение множеств вершин графов *G*1, *G*2, а множество ребер *U*c задается следующим образом: вершины (*zi*, *vk*) и (*zj*, *vl*) смежны в графе *G* тогда и только тогда, когда *zi* = *zj*(*i* = *j*), a *v*k и *vl* смежны в *G*2 или *vk* = *vl*(*k* = *l*), смежны в графе *G*1 (см. рис.3).

**X**

*z*1

*z*2

*v*1

*v*3

*v*2

*z*1*v*1

*z*1*v*2

*z*1*v*3

*z*2*v*1

*z*2*v*2

*z*2*v*3

Рис. 3. Декартово произведение графов *G*1, *G*2

*Кольцевая сумма* графов представляет граф, который не имеет изолированных вершин и состоит из ребер, присутствующих либо в первом исходном графе, либо во втором. Кольцевая сумма определяется следующим соотношением: *G* = *G*1  *G*2 (рис.4).

**⊕**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

*v*3

*v*2

Рис.4. Кольцевая сумма графов *G1, G2*

Листинг:

#include "stdafx.h"

#include <iostream>

#include <stdio.h>

#include <locale>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

void Ver1(int \*\*M1, int x, int y, int arr) { //отождествление

for (int i = x; i<arr;i++) {

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

if (M1[x][j] == 1) {

M1[x][j] = 1;

}

else {

M1[x][j] = M1[y][j];

}

}

}

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

for (int j = x;j<arr;j++) {

if (M1[i][x] == 1) {

M1[i][x] = 1;

}

else {

M1[i][x] = M1[i][y];

}

}

}

for (int i = y;i<arr;i++) {

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

M1[i][j] = M1[i + 1][j];

}

}

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

for (int j = y;j<arr;j++) {

M1[i][j] = M1[i][j + 1];

}

}

}

void Ver2(int \*\*M2, int x, int y, int arr) { // стягивание

for (int i = x; i<arr;i++) {

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

if (M2[x][j] == 1) {

M2[x][j] = 1;

}

else {

M2[x][j] = M2[y][j];

}

}

}

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

for (int j = x;j<arr;j++) {

if (M2[i][x] == 1) {

M2[i][x] = 1;

}

else {

M2[i][x] = M2[i][y];

}

}

}

for (int i = y;i<arr;i++) {

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

M2[i][j] = M2[i + 1][j];

}

}

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

for (int j = y;j<arr;j++) {

M2[i][j] = M2[i][j + 1];

}

}

}

void Splitting(int \*\*M3, int x, int arr, int y=0) { //расщепление

for (int i = arr; i>0;i--) {

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

if (i>x) {

M3[i][j] = M3[i - 1][j];

}

}

}

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

for (int j = arr;j>0;j--) {

if (j>x) {

M3[i][j] = M3[i][j - 1];

}

}

}

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

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

M3[x + 1][x] = 1;

M3[x][x + 1] = 1;

if (j>x + 1 && M3[i][j] != 0) {

M3[x][j] = y;

}

M3[i][x] = M3[x][j];

if (i == x + 1 && j>x + 1 && M3[i - 1][j] != 0) {

if (M3[i][j] == 0) {

M3[i][j] = 1;

}

else {

M3[i][j] = 0;

}

}

}

}

}

void Unification(int \*\*G1, int arr1, int \*\*M4, int arr) { // логическое сложение

int Num = 0;

if (arr1>arr) {

Num = arr1;

}

else if (arr1 == arr) {

Num = arr;

}

else if (arr1<arr) {

Num = arr;

}

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

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

if (arr1 >= arr) {

G1[i][j] = G1[i][j] || M4[i][j];

}

else {

M4[i][j] = G1[i][j] || M4[i][j];

}

}

}

}

void Intersections(int \*\*M5, int \*\*G2, int arr, int arr1) { // логическое умножение

int Num1 = 0;

if (arr1>arr) {

Num1 = arr1;

}

else if (arr1 == arr) {

Num1 = arr;

}

else if (arr1<arr) {

Num1 = arr;

}

for (int i = 0; i<Num1;i++) {

for (int j = 0; j<Num1;j++) {

if (arr1 >= arr) {

G2[i][j] = G2[i][j] && M5[i][j];

}

else {

M5[i][j] = G2[i][j] && M5[i][j];

}

}

}

}

void RingSum(int \*\*M6, int \*\*G3, int arr, int arr1) { //Кольцевая сумма

int Num2 = 0;

if (arr1>arr) {

Num2 = arr1;

}

else if (arr1 == arr) {

Num2 = arr;

}

else if (arr1<arr) {

Num2 = arr;

}

for (int i = 0; i<Num2;i++) {

for (int j = 0; j<Num2;j++) {

if (arr1 >= arr) {

G3[i][j] = G3[i][j] + M6[i][j];

if (G3[i][j] % 2 == 0) {

G3[i][j] = 0;

}

}

else {

M6[i][j] = G3[i][j] + M6[i][j];

if (M6[i][j] % 2 == 0) {

M6[i][j] = 0;

}

}

}

}

}

int main()

{

setlocale(0, "rus");

//Массивы для 2-ого задания

int \*\*M1;

int \*\*M2;

int \*\*M3;

//Массивы для 3-его задания

int \*\*G1;

int \*\*M4;

int \*\*G2;

int \*\*M5;

int \*\*G3;

int \*\*M6;

int arr, arr1;//размер массива

int x, y; //номера вершин

printf("Введем размер матрицы:");

scanf\_s("%d %d", &arr, &arr1);

M1 = (int \*\*)malloc(100 \* sizeof(int));

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

M1[i] = (int\*)malloc(100 \* sizeof(int));

}

M2 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

M2[i] = (int\*)malloc(100 \* sizeof(int));

}

M3 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

M3[i] = (int\*)malloc(100 \* sizeof(int));

}

M4 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

M4[i] = (int\*)malloc(100 \* sizeof(int));

}

M5 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

M5[i] = (int\*)malloc(100 \* sizeof(int));

}

M6 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

M6[i] = (int\*)malloc(100 \* sizeof(int));

}

G1 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

G1[i] = (int\*)malloc(100 \* sizeof(int));

}

G2 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

G2[i] = (int\*)malloc(100 \* sizeof(int));

}

G3 = (int \*\*)malloc(100 \* sizeof(int));

for (int i = 0; i<100;i++) {

G3[i] = (int\*)malloc(100 \* sizeof(int));

}

int k;

// Матрица 1

printf("\nВыведем матрицу M1:\n");

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

for (int j = 0;j<i + 1;j++) {

k= rand() % 2;

M1[i][j] = k;

M1[j][i] = k;

}

}

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

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

if (i == j) {

M1[i][j] = 0;

}

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

}

printf("\n");

}

printf("\nG1 ");

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

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

M2[i][j] = M1[i][j];

}

}

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

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

M3[i][j] = M1[i][j];

}

}

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

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

M4[i][j] = M1[i][j];

}

}

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

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

M5[i][j] = M1[i][j];

}

}

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

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

M6[i][j] = M1[i][j];

}

}

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

printf("\n 1 вершина: ");

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

printf("\n 2 вершина: ");

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

if (x<arr && y<arr && x != y) {

printf("Вы ввели правельные координаты\n");

}

else {

printf("Вы ввели не правельные координаты\n");

i--;

}

}

printf("\n\nОтождествления вершин \n");

Ver1(M1, x, y, arr);

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

for (int j = 0; j < arr - 1; j++) {

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

}

printf("\n");

}

printf("\n\nСтягивания ребра\n");

Ver2(M2, x, y, arr);

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

for (int j = 0; j < arr - 1; j++) {

if (i == j) {

M2[i][j] = 0;

}

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

}

printf("\n");

}

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

printf("\n 1 вершина: ");

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

if (x<arr) {

printf("Вы ввели правельные координаты\n");

}

else {

printf("Вы ввели не правельные координаты\n");

i--;

}

}

printf("\n\nРазщепление вершин\n");

Splitting(M3, x, arr);

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

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

printf("%d ", M3[i][j]);

}

printf("\n");

}

// Матрица 2

printf("\n\nВыведем матрицу M2:\n");

for (int i = 0; i<arr1;i++) {

for (int j = 0;j<i + 1;j++) {

int k = rand() % 2;

G1[i][j] = k;

G1[j][i] = k;

}

}

for (int i = 0; i < arr1; i++) {

for (int j = 0; j < arr1; j++) {

if (i == j) {

G1[i][j] = 0;

}

printf("%d ", G1[i][j]);

}

printf("\n");

}

printf("\n G2 \n");

for (int i = 0; i < arr1; i++) {

for (int j = 0; j < arr1; j++) {

G2[i][j] = G1[i][j];

}

}

for (int i = 0; i < arr1; i++) {

for (int j = 0; j < arr1; j++) {

G3[i][j] = G1[i][j];

}

}

//Выведем обьединение матриц

printf("\n\nВыведем обьединеную матрицу:\n");

Unification(G1, arr1, M4, arr);

int Num = 0;

if (arr1>arr) {

Num = arr1;

}

else if (arr1 == arr) {

Num = arr;

}

else if (arr1<arr) {

Num = arr;

}

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

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

if (arr1 >= arr) {

printf("%d ", G1[i][j]);

}

else {

printf("%d ", M4[i][j]);

}

}

printf("\n");

}

printf("\n\nВыведем пересечение матриц:\n");

Intersections(M5, G2, arr, arr1);

int Num1 = 0;

if (arr1>arr) {

Num1 = arr1;

}

else if (arr1 == arr) {

Num1 = arr;

}

else if (arr1<arr) {

Num1 = arr;

}

for (int i = 0; i < Num1; i++) {

for (int j = 0; j < Num1; j++) {

if (arr1 >= arr) {

printf("%d ", G2[i][j]);

}

else {

printf("%d ", M5[i][j]);

}

}

printf("\n");

}

printf("\n\nВыведем кольцевое сложение матриц:\n");

RingSum(M6, G3, arr, arr1);

int Num2 = 0;

if (arr1>arr) {

Num2 = arr1;

}

else if (arr1 == arr) {

Num2 = arr;

}

else if (arr1<arr) {

Num2 = arr;

}

for (int i = 0; i < Num2; i++) {

for (int j = 0; j < Num2; j++) {

if (arr1 >= arr) {

printf("%d ", G3[i][j]);

}

else {

printf("%d ", M6[i][j]);

}

}

printf("\n");

}

free(M1);

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

free(M1[i]);

}

free(M2);

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

free(M2[i]);

}

free(M3);

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

free(M3[i]);

}

free(M4);

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

free(M4[i]);

}

free(M5);

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

free(M5[i]);

}

free(M6);

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

free(M6[i]);

}

free(G1);

for (int i = 0; i < arr1; i++) {

free(G1[i]);

}

free(G2);

for (int i = 0; i < arr1; i++) {

free(G2[i]);

}

free(G3);

for (int i = 0; i < arr1; i++) {

free(G3[i]);

}

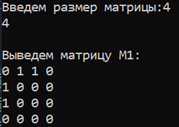
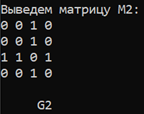
getchar();

return 0;

}

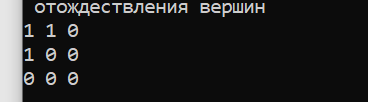
Задание 1

1. Сгенерируйте (используя генератор случайных чисел) две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Выведите сгенерированные матрицы на экран.

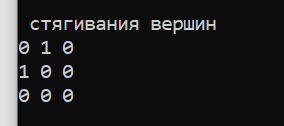
 

Задание 2

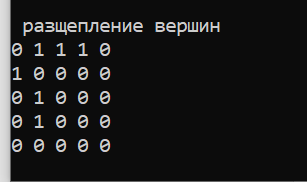
а) отождествления вершин



б) стягивания ребра

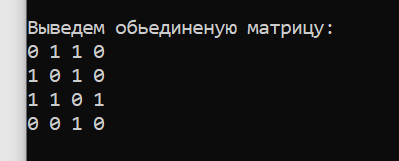


в) расщепления вершины

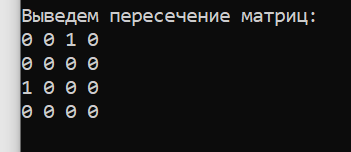


Задание 3

а) объединения *G* = *G*1  *G*2



б) пересечения *G* = *G*1  *G*2



в) кольцевой суммы *G* = *G*1  *G*2

