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**»

**Выполнил студент группы 19ВВ1:**

Артемов К.А.

**Приняли:**

д.т.н. профессор Митрохин М. А.

к.т.н. Юрова О.В.

Пенза 2020.

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

**Лабораторное задание:**

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.

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.

Задание 4 \*

1. Для матричной формы представления графов выполните операцию

декартова произведения графов G = G 1 X G 2 .

Результат выполнения операции выведите на экран.

**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 .

**Декартово произведение** графов G1 = (X,A) и G2=(Y,B), обозначаемое как

G1 x G2, представляет собой граф, множество вершин которого является декартовым произведением множеств X x Y, т.е. вершины этого графа – все упорядоченные пары (x, y). При этом вершины (x1, y1) и (x2, y2) в новом графе смежны, если x1=x2 и y1 смежна с y2 в графе G2 , или y1=y2 и x1 смежна с x2 в графе G1.

**Практическая часть:**

**Листинг:**

// Lab3.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <locale.h>

#define SIZE 5

void rand\_Zap(int k, int\* Matrix, int n) {

srand(time(NULL) \* k);

printf("G%d \n",k);

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

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

if (i == j) {

Matrix[i \* n + j] = 0;

}

if (i < j) {

Matrix[i \* n + j] = rand() % 2;

Matrix[j \* n + i] = Matrix[i \* n + j];

}

}

}

void print\_G(int \*Matrix, 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", Matrix[i\*n+j]);

}

printf("\n");

}

printf("\n");

}

void copy(int(&Matrix1)[SIZE][SIZE], int(&Matrix2)[SIZE][SIZE]) {

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

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

Matrix2[i][j] = Matrix1[i][j];

}

void otj(int (&constant)[SIZE][SIZE], int (&work)[SIZE][SIZE],int x1, int x2, int stig ){

int G4[SIZE - 1][SIZE - 1], i, j, i1, j1;

copy(constant, work);

for (i = 0; i < SIZE; i++) {

work[i][x1] = constant[i][x1] || constant[i][x2];

}

for (j = 0; j < SIZE; j++) {

work[x1][j] = constant[x1][j] || constant[x2][j];

}

for (i = 0, i1=x2+1; i < SIZE - 1; i++) {

for (j = 0, j1 = x2; j < SIZE - 1; j++) {

if (j >= x2)

j1++;

if (i < x2 && j < x2)

G4[i][j] = work[i][j];

if (i < x2 && j >= x2)

G4[i][j] = work[i][j1];

if (i >= x2 && j < x2)

G4[i][j] = work[i1][j];

if (i >= x2 && j >= x2)

G4[i][j] = work[i1][j1];

}

if (i >= x2) i1++;

}

if (constant[x1][x2] == 1 && stig==0)

G4[x1][x1] = 1;

else

G4[x1][x1] = 0;

printf("Результат:\n");

print\_G(&G4[0][0], SIZE - 1);

}

void ras(int (&constant)[SIZE][SIZE]) {

int x, i, j, i1, Matrix[SIZE + 1][SIZE + 1];

printf("Введите вершину для расщепления\n");

scanf("%d", &x);

x--;

for (i = 0; i < SIZE; i++) {

for (j = 0; j < SIZE; j++) {

Matrix[i][j] = constant[i][j];

}

}

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

for (j = 0; j < SIZE + 1; j++) {

if (i == 5 || j == 5) {

Matrix[5][j] = 0;

Matrix[j][5] = 0;

}

}

}

for (j = 0; j < SIZE; j++) {

if (Matrix[x][j] == 1)

if (j % 2 == 0) {

Matrix[5][j] = 1;

Matrix[j][5] = 1;

Matrix[x][j] = 0;

Matrix[j][x] = 0;

}

}

Matrix[x][5] = 1;

Matrix[5][x] = 1;

printf("Результат расщепления\n\n");

print\_G(&Matrix[0][0], SIZE + 1);

}

void obed(int(&Matrix1)[SIZE][SIZE], int(&Matrix2)[SIZE][SIZE]) {

int res[SIZE][SIZE];

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

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

res[i][j] = Matrix1[i][j] || Matrix2[i][j];

}

printf("Результат объединения G1 и G2\n");

print\_G(&res[0][0], SIZE);

}

void perseh(int(&Matrix1)[SIZE][SIZE], int(&Matrix2)[SIZE][SIZE]) {

int res[SIZE][SIZE];

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

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

if (Matrix1[i][j]==1 && Matrix2[i][j]==1)

res[i][j] = 1;

else

res[i][j] = 0;

}

printf("Результат пересечения G1 и G2\n");

print\_G(&res[0][0], SIZE);

}

void kolc\_sum(int(&Matrix1)[SIZE][SIZE], int(&Matrix2)[SIZE][SIZE]) {

int res[SIZE][SIZE];

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

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

if ((Matrix1[i][j] == 1 && Matrix2[i][j] == 0) || (Matrix1[i][j] == 0 && Matrix2[i][j] == 1))

res[i][j] = 1;

else

res[i][j] = 0;

}

printf("Результат кольцевой суммы G1 и G2\n");

print\_G(&res[0][0], SIZE);

}

void dekart(int\* Matrix1, int\* Matrix2) {

int i = 0, j = 0, Gdek[SIZE \* SIZE][SIZE \* SIZE];

for (int i1 = 0; i1 < SIZE; i1++)

for (int j1 = 0; j1 < SIZE; j1++, i++)

for (int i2 = 0, j = 0; i2 < SIZE; i2++)

for (int j2 = 0; j2 < SIZE; j2++) {

if (i1 == i2 && j1 != j2)

{

Gdek[i][j] = Matrix2[j1\* SIZE +j2];

}

if (i1 != i2 && j1 == j2)

{

Gdek[i][j] = Matrix1[i1\* SIZE +i2];

}

if (i1 != i2 && j1 != j2)

{

Gdek[i][j] = 0;

}

if (i1 == i2 && j1 == j2)

{

Gdek[i][j] = 0;

}

j++;

}

printf("\n\n");

print\_G(&Gdek[0][0], SIZE \* SIZE);

}

typedef struct Node {

int value;

struct Node\* next;

}Spisok;

Spisok\* G\_S1[SIZE];

Spisok\* G\_S2[SIZE];

Spisok\* create\_V(int name)

{

// Выделение памяти под корень списка

Spisok\* tmp = (Spisok\*)malloc(sizeof(Spisok));

// Присваивание имя вершине

tmp->value = name;

// Присваивание указателю на следующий элемент значения NULL

tmp->next = NULL;

return(tmp);

}

void add\_element\_end(int data, Spisok\* head)

{

// Выделение памяти под корень списка

Spisok\* tmp = (Spisok\*)malloc(sizeof(Spisok));

// Присваивание значения узлу

tmp->value = data;

// Присваивание указателю на следующий элемент значения NULL

tmp->next = NULL;

// Присваивание новому указателю указателя head.

// Присваивание выполняется для того, чтобы не потерять указатель на «голову» списка

Spisok\* p = head;

// Сдвиг указателя p в самый конец первоначального списка

while (p->next != NULL)

p = p->next;

// Присваивание указателю p -> next значения указателя tmp (созданный новый узел)

p->next = tmp;

}

Spisok\* remove\_all(Spisok\* head)

{

// Сдвиг указателя head в самый конец первоначального списка

while (head != NULL)

{

// Присваивание новому указателю указателя head

Spisok\* p = head;

head = head->next;

// Освобождение памяти для указателя p

free(p);

}

return NULL;

}

Spisok\* remove\_element(int data, Spisok\* head)

{

// Присваивание новому указателю tmp указателя head, p - NULL

Spisok\* tmp = head, \* p = NULL;

// Проверка списка на пустоту

if (head == NULL)

return NULL;

// Если список не пуст, поиск указателя на искомый элемент

while (tmp && tmp->value != data)

{

p = tmp;

tmp = tmp->next;

}

// Если удаляемый элемент первый

if (tmp == head)

{

free(head);

return tmp->next;

}

// Если в списке нет искомого элемента, то возвращаем первоначальный список

if (!tmp)

return head;

// Присваивание новому указателю указателя tmp

p->next = tmp->next;

// Освобождение памяти для указателя tmp

free(tmp);

return head;

}

void print(Spisok\* head) {

Spisok\* v = head;

if (v != NULL) {

printf("%d: ", v->value + 1);

v = v->next;

}

while (v != NULL)

{

// Вывод значения узла

printf("%d ", v->value+1);

// Сдвиг указателя к следующему узлу

v = v->next;

}

}

void graf\_iz\_matrix(int(&Matrix)[SIZE][SIZE], Spisok\* (&P)[SIZE]) {

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

Node\* tmp = create\_V(i);

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

if (Matrix[i][j] == 1)

add\_element\_end(j, tmp);

}

P[i] = tmp;

}

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

print(P[i]);

printf("\n");

}

printf("\n");

}

/\*void otj\_S(Spisok\* (&P)[SIZE],int x1, int x2) {

Spisok\* fx1;

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

Spisok\* v = P[i];

fx1=create\_V(x1);

if (v->value != x2) {

if (v->next != NULL) v = v->next;

while (v != NULL) {

if (v->value == x2) {

v->value = x1;

add\_element\_end(v->value, fx1);

}

v = v->next;

}

}

}

//P[x2] = remove\_all(P[x2]);

//P[x1] = remove\_all(P[x1]);

P[x1] = fx1;

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

if (i == x2) continue;

print(P[i]);

printf("\n");

}

printf("\n");}

\*/

int main(){

int G1[SIZE][SIZE], G2[SIZE][SIZE], work[SIZE][SIZE], i, x1, x2, j, A[20][20];

setlocale(LC\_ALL, "Rus");

rand\_Zap(1, &G1[0][0], SIZE);

print\_G(&G1[0][0], SIZE);

rand\_Zap(2, &G2[0][0], SIZE);

print\_G(&G2[0][0], SIZE);

graf\_iz\_matrix(G1,G\_S1);

graf\_iz\_matrix(G2,G\_S2);

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_отождествление\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

printf("Введите 2 вершины для их отождествления \n");

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

if (x1 > x2) {

i = x1;

x1 = x2;

x2 = i;

}

if ((x1 >= 1) && (x2 <= SIZE)) {

x1--; x2--;

otj(G1, work, x1, x2,0);

}

else

printf("\n ошибка Нет вершин(ы)\n");

/\*printf("Введите 2 вершины для их отождествления \n");

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

if (x1 > x2) {

i = x1;

x1 = x2;

x2 = i;

}

if ((x1 >= 1) && (x2 <= SIZE)) {

x1--; x2--;

otj\_S(G\_S1, x1, x2);

}

else

printf("\n ошибка Нет вершин(ы)\n");\*/

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_стян\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

printf("Введите 2 вершины, смежные ребру, которое нужно стянуть \n");

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

if (x1 > x2) {

i = x1;

x1 = x2;

x2 = i;

}

if ((x1 >= 1) && (x2 <= SIZE)) {

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

{

x1--; x2--;

otj(G1, work, x1, x2,1);

}

else printf("\n ошибка Нет ребра\n");

}

else

printf("\n ошибка Нет вершин(ы)\n");

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Расщепление\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

ras(G1);

printf("\n Нажмите любую клавишу, чтобы продолжить\n");

getchar();

obed(G1, G2);

printf("\n Нажмите любую клавишу, чтобы продолжить\n");

getchar();

perseh(G1, G2);

printf("\n Нажмите любую клавишу, чтобы продолжить\n");

getchar();

kolc\_sum(G1, G2);

printf("\n Нажмите любую клавишу, чтобы продолжить\n");

getchar();

dekart(&G1[0][0], &G2[0][0]);

}

**Результат работы кода:**

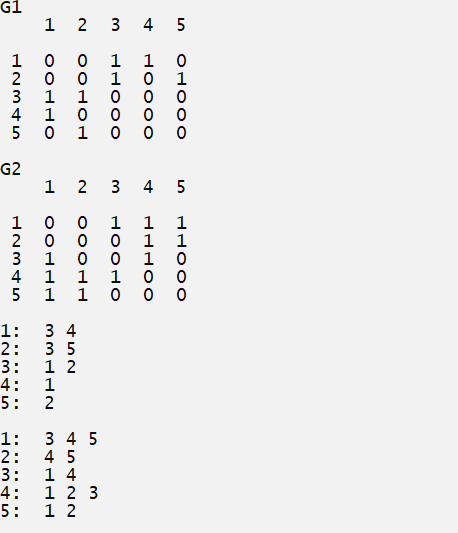
****

Рис.1 вывод матриц в виде матриц смежности и списков смежности

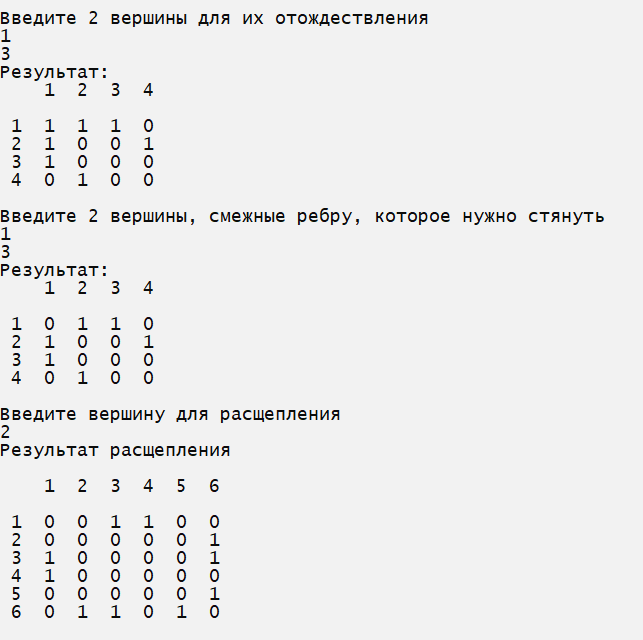
****

Рис.2 унарные операции

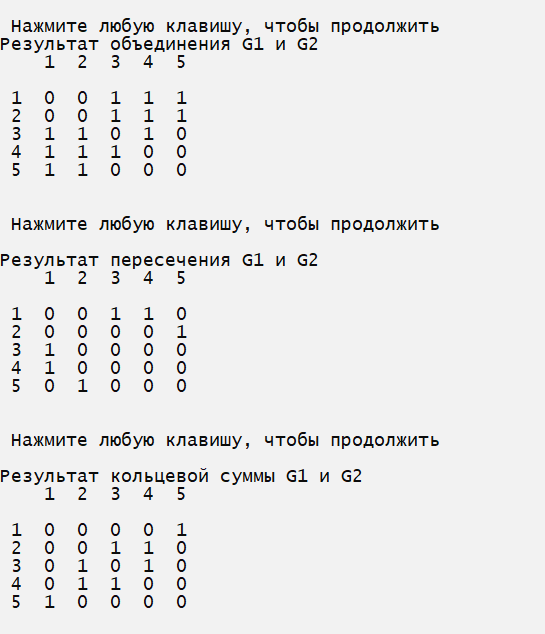
****

Рис.3 бинарные операции

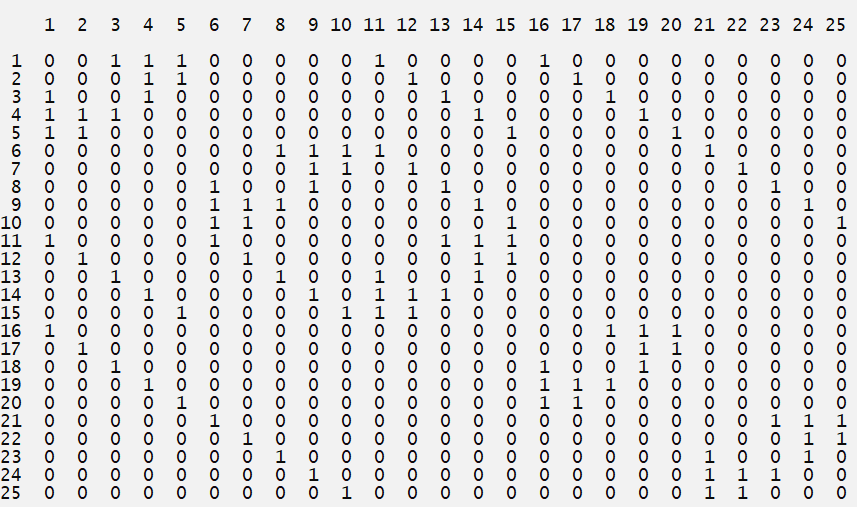
****

Рис.4 декартово произведение

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