Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Выполнил:**

Студенты группы 19ВВ2:

Корнаухова А.С.

Гуськова Н.Д.

**Приняли:**

Митрохин М.А.

Юрова О.В.

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Пенза 2020

### 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.

1. \* Для представления графов в виде списков смежности 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*1X *G*2.

Результат выполнения операции выведите на экран.

### Цель работы:

Разработать программы по данным заданиям.

**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* = *Q1Q2.* 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*

### Описание метода решения задачи:

**Задача 1:**

### Листинг:

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

#include<time.h>

#include<locale.h>

typedef struct Node{

char ver;

struct Node\* next;

} Node;

void push(Node\*\* head, char data)

{

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

tmp->ver = data;

tmp->next = (\*head);

(\*head) = tmp;

}

Node\* getLast(Node\* head) {

if (head == NULL) {

return NULL;

}

while (head->next) {

head = head->next;

}

return head;

}

void printLinkedList(const Node\* head) {

if (head) {

printf("%c", head->ver);

head = head->next;

}

while (head) {

printf("--->%c", head->ver);

head = head->next;

}

printf("\n");

}

void pushBack(Node\* head, char value) {

Node\* last = getLast(head);

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

tmp->ver = value;

tmp->next = NULL;

last->next = tmp;

}

void printarray(int\*\* a, int n)

{

char letter = 'A';

if (n > 0)

printf("%10c", letter);

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

{

letter++;

printf("%5c", letter);

}

printf("\n\n");

letter = 'A';

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

{

printf("%5c", letter);

letter++;

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

{

printf("%5d", a[i][j]);

}

printf("\n\n");

}

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

}

void identification(int v1, int v2, int\*\* a, int n)

{

int del, field;

if (v1 > v2)

{

del = v1 - 1;

field = v2 - 1;

}

else

{

del = v2 - 1;

field = v1 - 1;

}

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

{

if (a[del][i] == 1)

a[field][i] = 1;

if (a[i][del] == 1)

a[i][field] = 1;

}

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

{

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

{

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

}

}

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

{

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

{

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

}

}

printarray(a, n - 1);

}

void initializinglist(int\*\* a, Node\*\* head, int n, char letter)

{

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

{

push(&head[i], letter + i);

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

{

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

{

pushBack(head[i], letter + j);

}

}

}

}

void cleanlist(Node\*\* head, int n)

{

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

head[i] = NULL;

}

void cleanlistfield(Node\*\* head, int del, int n)

{

char letter = 'A', pressF;

letter += del;

Node\* tmp;

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

{

tmp = head[i];

while (tmp)

{

if (tmp->next)

if (tmp->next->ver == letter)

{

tmp->next = tmp->next->next;

}

if (tmp->ver >= letter)

{

pressF = tmp->ver;

pressF--;

tmp->ver = pressF;

}

tmp = tmp->next;

}

}

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

{

head[i] = head[i + 1];

}

head[n - 1] = NULL;

}

void paste(Node\* head, char letter)

{

Node\* tmp;

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

tmp->ver = letter;

tmp->next = head->next;

head->next = tmp;

}

void addver(Node\*\* head, int field, int n)

{

Node\* ptr, \* tmp;

char letter = head[field]->ver, search;

tmp = head[field];

tmp = tmp->next;

while (tmp)

{

search = tmp->ver;

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

{

if (head[i]->ver == search)

{

if (!head[i]->next)

{

push(&head[i], letter);

}

ptr = head[i];

while (ptr->next)

{

if (ptr->next->ver == letter)

{

break;

}

if (ptr->next->ver > letter)

{

paste(ptr, letter);

}

}

break;

}

}

tmp = tmp->next;

}

}

void identificationlist(int v1, int v2, Node\*\* head, int n)

{

int del, field;

char letter = 'A';

Node\* ptr = NULL, \* tmp;

if (v1 > v2)

{

del = v1 - 1;

field = v2 - 1;

}

else

{

del = v2 - 1;

field = v1 - 1;

}

tmp = head[del]->next;

ptr = (Node\*)malloc(sizeof(Node));

ptr->ver = head[field]->ver;

ptr->next = NULL;

head[field] = head[field]->next;

while (head[field] || tmp)

{

if (head[field] && tmp)

{

if (head[field]->ver > tmp->ver)

{

pushBack(ptr, tmp->ver);

tmp = tmp->next;

continue;

}

if (head[field]->ver < tmp->ver)

{

pushBack(ptr, head[field]->ver);

head[field] = head[field]->next;

continue;

}

if (head[field]->ver == tmp->ver)

{

pushBack(ptr, tmp->ver);

tmp = tmp->next;

head[field] = head[field]->next;

continue;

}

}

if (head[field])

{

while (head[field])

{

pushBack(ptr, head[field]->ver);

head[field] = head[field]->next;

}

}

else

{

while (tmp)

{

pushBack(ptr, tmp->ver);

tmp = tmp->next;

}

}

}

head[field] = ptr;

fflush(stdin);

cleanlistfield(head, del, n);

addver(head, field, n);

}

void spliting(int\*\* a, int v, int n)

{

v--;

int i = v;

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

{

if (a[v][i] == 1)

{

a[n - 1][i] = a[i][n - 1] = 1;

a[v][i] = a[i][v] = 0;

i++;

break;

}

a[n - 1][i] = a[i][n - 1] = 0;

}

while (i <= n)

{

a[n - 1][i] = a[i][n - 1] = 0;

i++;

}

a[n - 1][v] = a[v][n - 1] = 1;

}

void combiningarray(int\*\* a, int\*\* b, int\*\* c, int n)

{

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

{

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

{

if (a[i][j] == 1 || b[i][j] == 1)

{

c[i][j] = 1;

}

else

{

c[i][j] = 0;

}

}

}

}

void intersectionarray(int\*\* a, int\*\* b, int\*\* c, int n)

{

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

{

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

{

if (a[i][j] == 1 && b[i][j] == 1)

{

c[i][j] = 1;

}

else

{

c[i][j] = 0;

}

}

}

}

void ringsumarray(int\*\* a, int\*\* b, int\*\* c, int n)

{

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

{

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

{

if ((a[i][j] == 1 && b[i][j] == 0) || (a[i][j] == 0 && b[i][j] == 1))

{

c[i][j] = 1;

}

else

{

c[i][j] = 0;

}

}

}

}

void splitinglist(Node\*\* head, int v, int n)

{

push(&head[n - 1], 'A' + n - 1);

Node\* ptr = head[v - 1];

if (ptr->next)

{

if (ptr->ver < ptr->next->ver)

{

pushBack(head[n - 1], ptr->ver);

pushBack(head[n - 1], ptr->next->ver);

ptr->next = ptr->next->next;

}

else

{

pushBack(head[n - 1], ptr->next->ver);

pushBack(head[n - 1], ptr->ver);

ptr->next = ptr->next->next;

}

}

else

{

pushBack(head[n], ptr->ver);

}

}

void main()

{

setlocale(LC\_ALL, "RUS");

char letter = 'A';

int\*\* a, \*\* b, n = 5, v1, v2, del;

Node\*\* head = (Node\*\*)malloc(n \* sizeof(Node\*));

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

head[i] = NULL;

srand(time(NULL));

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

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

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

{

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

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

}

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

{

for (int j = n - 1; j >= 0; j--)

{

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

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

}

}

printarray(a, n);

printarray(b, n);

initializinglist(a, head, n, letter);

printf("list A:\n");

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

printLinkedList(head[i]);

cleanlist(head, n);

printf("\n\n");

initializinglist(b, head, n, letter);

printf("list B:\n");

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

printLinkedList(head[i]);

cleanlist(head, n);

printf("\n\n");

initializinglist(a, head, n, letter);

printf("Выберите вершины в матрице А для отождествления\n Вершина 1: ");

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

printf("\n Вершина 2: ");

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

identification(v1, v2, a, n);

identificationlist(v1, v2, head, n);

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

printLinkedList(head[i]);

cleanlist(head, n);

printf("\n\n");

initializinglist(b, head, n, letter);

printf("Выберите вершины в матрице В для отождествления\n Вершина 1: ");

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

printf("\n Вершина 2: ");

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

identification(v1, v2, b, n);

identificationlist(v1, v2, head, n);

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

printLinkedList(head[i]);

cleanlist(head, n);

n--;

initializinglist(a, head, n, letter);

printf("\nВыберите вершину в матрице А для расщепления");

for (;;)

{

printf("\n Вершина 1: ");

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

printf("\n Вершина 2: ");

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

if (a[v1 - 1][v2 - 1] == 1)

break;

else

printf("Error enter again!!!");

}

identification(v1, v2, a, n);

identificationlist(v1, v2, head, n);

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

printLinkedList(head[i]);

cleanlist(head, n);

initializinglist(b, head, n, letter);

printf("\nВыберите вершину в матрице В для расщепления");

for (;;)

{

printf("\n Вершина 1: ");

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

printf("\n Вершина 2: ");

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

if (b[v1 - 1][v2 - 1] == 1)

break;

else

printf("Error enter again!!!");

}

identification(v1, v2, b, n);

identificationlist(v1, v2, head, n);

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

printLinkedList(head[i]);

cleanlist(head, n);

initializinglist(a, head, n - 1, letter);

printf("Выберите вершину в матрице А для рсщепления: ");

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

spliting(a, v1, n);

splitinglist(head, v1, n);

printarray(a, n);

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

printLinkedList(head[i]);

cleanlist(head, n);

printf("Выберите вершину в матрице В для рсщепления: ");

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

spliting(b, v1, n);

printarray(b, n);

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

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

{

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

}

printf("\n\nОбъединиение матриц А и В:\n\n");

combiningarray(a, b, c, n);

printarray(c, n);

printf("\n\nПересечение мартиц А и В:\n\n");

intersectionarray(a, b, c, n);

printarray(c, n);

printf("\n\nКольцевая сумма матриц А и В:\n\n");

ringsumarray(a, b, c, n);

printarray(c, n);

printf("Декартово произведение A и B");

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

{

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

{

printf("\n\n");

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

{

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

{

if (i == j) {

printf("%5d", b[q][k]);

}

if (q == k)

{

printf("%5d", a[i][j]);

}

if ((i != j) && (q != k))

{

printf("%5d", 0);

}

}

}

}

}

printf("\n\n");

system("pause");

}

### Результаты работы программы:

### 

Рисунок 1 Вывод матриц и списков смежности

### 

Рисунок 2 Отождествление вершин

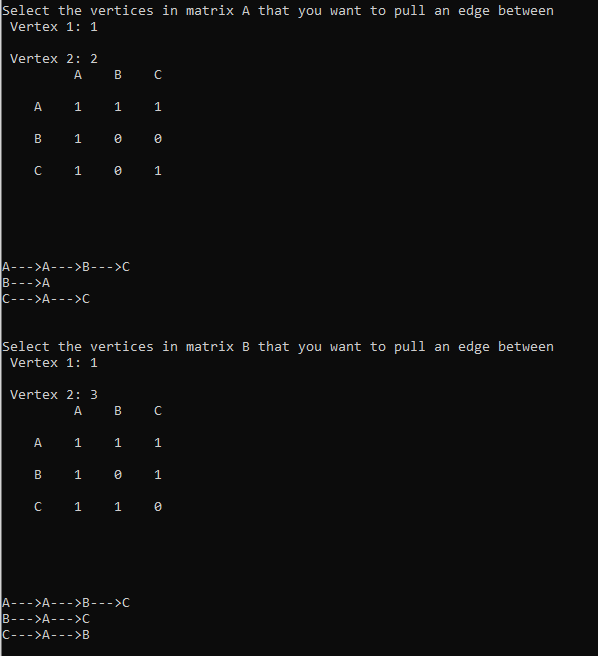
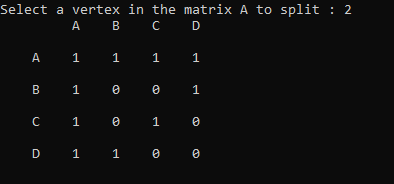


Рисунок 3 Стягивание рёбер



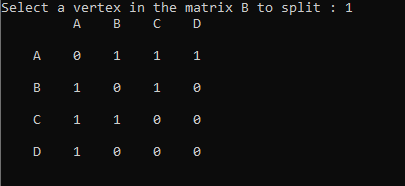


Рисунок 4 Расщепление вершин

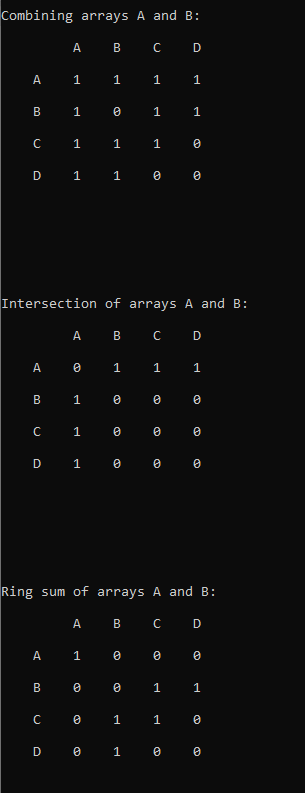


Рисунок 5 Объединение, пересечение и кольцевая сумма

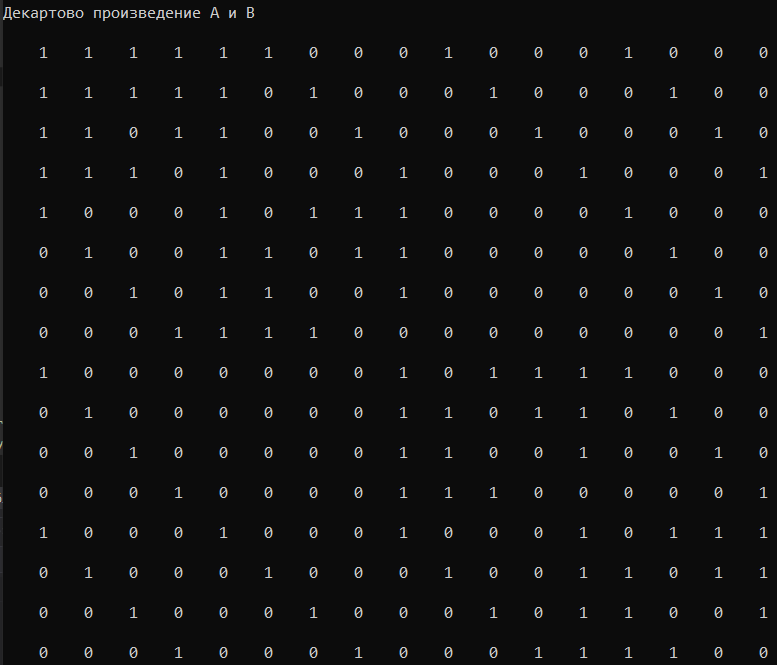


Рисунок 6 Декартово произведение

### Вывод:

В ходе выполнения лабораторной работы мы научились производить унарные операции над графами, заданными списками и матрицами смежности. Были разработаны программы, соответствующие данным заданиям.