Министерство высшего образования и науки Российской Федерации

Пензенский Государственный Университет

Кафедра «Вычислительная техника»

ОТЧЕТ

Лабораторная работа №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Унарные и бинарные операции над графами»

Выполнил студент группы 20ВВ2:

Пантелеев И. А.

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2021

**Название**

Унарные и бинарные операции над графами.

**Цель работы**

Научиться производить унарные и бинарные операции над графами.

**Лабораторное задание**

**Задание 1:**

1. Сгенерируйте (используя генератор случайных чисел) две матрицы 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.

2. \* 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

б) пересечения

в) кольцевой суммы

Результат выполнения операции выведите на экран.

**Листинг:**

# **Lr3.cpp**

#include <iostream>

#include <windows.h>

using namespace std;

int\*\* M1, \*\* M2, \*\*M3;

void fill();

void create(int size)

{

int i, j;

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

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

srand(time(NULL));

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

{

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

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

for (j = 0 + i; j < size; j++)

{

M1[i][j] = 0;

M2[i][j] = rand() % 2;

}

}

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

{

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

{

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

if (i == j)

M1[i][j] = 0;

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

if (i == j)

M2[i][j] = 0;

}

}

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

{

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

{

cout << M2[i][j] << ' ';

}

cout << endl;

}

}

void InitM3(int size)

{

int i, j;

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

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

{

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

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

{

M3[i][j] = 0;

}

}

}

void identify(int size, int first, int second)

{

int i, j;

int razn = second - first;

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

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

if ((i < second) && (j < second)) { M1[i][j] = M2[i][j]; }

if ((i <= first) && (j == second)) { M1[i][first] += M2[i][j]; }

if ((i == second) && (j < first)) { M1[first][j] += M2[i][j]; }

if ((i < second) && (j > second)) { M1[i][j - 1] = M2[i][j]; }

if ((j < second) && (i > second)) { M1[i - 1][j] = M2[i][j]; }

if ((i == first + 1) && (j == second)) { M1[i - 1][j - razn + 1] += M2[i][j]; }

if ((j == first + 1) && (i == second)) { M1[i - razn + 1][j - 1] += M2[i][j]; }

}

}

for (i = second + 1; i < size; i++) {

for (j = second + 1; j < size; j++) {

M1[i - 1][j - 1] = M2[i][j];

}

}

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

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

if (M1[i][j] > 1) { M1[i][j] = 1; }

}

}

}

void constrict(int size, int first, int second)

{

int i, j, razn;

razn = second - first;

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

{

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

{

if ((i < second) && (j < second)) M1[i][j] = M2[i][j];

if ((i < first) && (j == second)) M1[i][first] += M2[i][j];

if ((i == second) && (j < first)) M1[first][j] += M2[i][j];

if ((i < second) && (j > second)) M1[i][j - 1] = M2[i][j];

if ((j < second) && (i > second)) M1[i - 1][j] = M2[i][j];

if ((i == first + 1) && (j == second)) M1[i - 1][j - razn + 1] += M2[i][j];

if ((j == first + 1) && (i == second)) M1[i - razn + 1][j - 1] += M2[i][j];

}

}

for (i = second + 1; i < size; i++) {

for (j = second + 1; j < size; j++) {

M1[i - 1][j - 1] = M2[i][j];

}

}

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

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

if (M1[i][j] > 1) { M1[i][j] = 1; }

}

}

}

void split(int size, int pos)

{

int i, j;

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

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

M3[i][j] = M2[i][j];

if (j == pos) {

if (i == pos) { M3[i][size] = 1; M3[size][i] = M3[i][size]; }

else { M3[i][size] = M2[i][pos]; M3[size][i] = M2[i][pos]; }

}

}

}

}

void InitM1(int size)

{

int i, j;

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

{

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

{

M1[i][j] = rand() % 2;

}

}

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

{

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

{

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

}

printf("\n");

}

printf("\n");

}

void SumMatrix(int size)

{

int i, j;

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

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

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

if (M1[i][j] == 2) { M1[i][j] = 1; }

cout << M1[i][j] << " ";

}

cout << endl;

}

}

void CrossMatrix(int size)

{

int i, j;

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

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

if (M1[i][j] != M2[i][j]) { M1[i][j] = 0; }

cout << M1[i][j] << " ";

}

cout << endl;

}

}

void RingSum(int size)

{

int i, j;

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

{

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

{

if (M1[i][j] == M2[i][j]) { M1[i][j] = 0; }

else { M1[i][j] = 1; }

cout << M1[i][j] << " ";

}

cout << endl;

}

}

void decart();

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int i, j, Number, pos, size, first, second, type;

cout << " " << endl;

cin >> type;

switch (type) {

case 1:

cout << "Введите размер графа:" << endl;

cin >> size;

create(size);

cout << "Введите тип операции:\n1-отождествление\n2-стягивание\n3-расщепление\n4-объединение\n5-пересечение\n6-кольц. сумма\n";

cin >> Number;

switch (Number)

{

case 1:

cout << "Введите отождествляемые вершины" << endl;

cin >> first >> second;

identify(size, first - 1, second - 1);

for (i = 0; i < size - 1; i++)

{

for (j = 0; j < size - 1; j++)

{

cout << M1[i][j] << ' ';

}

cout << endl;

}

break;

case 2:

cout << "Введите стягивающиеся вершины" << endl;

cin >> first >> second;

if (M2[first][second] == 1)

{

constrict(size, first - 1, second - 1);

for (i = 0; i < size - 1; i++)

{

for (j = 0; j < size - 1; j++)

{

cout << M1[i][j] << ' ';

}

cout << endl;

}

}

else cout << "Невозможно стянуть данные вершины";

break;

case 3:

cout << "Введите вершину для расщепления" << endl;

cin >> pos;

InitM3(size + 1);

split(size, pos - 1);

for (i = 0; i < size + 1; i++)

{

for (j = 0; j < size + 1; j++)

{

cout << M3[i][j] << ' ';

}

cout << endl;

}

break;

case 4:

printf("Вторая матрица\n");

InitM1(size);

printf("Итоговая матрица\n");

SumMatrix(size);

break;

case 5:

printf("Вторая матриц\n");

InitM1(size);

printf("Итоговая матриц\n");

CrossMatrix(size);

break;

case 6:

printf("Вторая матрица\n");

InitM1(size);

printf("Итоговая матрица\n");

RingSum(size);

break;

}

break;

case 2:

fill();

break;

case 3:

decart();

break;

}

}

**Struct.cpp**

#include <time.h>

#include <malloc.h>

#include <locale.h>

#include <iostream>

using namespace std;

struct Node {

int items;

Node\* next;

};

typedef Node\* PNode;

PNode CreateNode(int items) {

PNode NewNode = new Node;

NewNode->items = items;

NewNode->next = NULL;

return NewNode;

}

void AddFirst(PNode NewNode, int i, PNode\* Head) {

NewNode->next = Head[i];

Head[i] = NewNode;

}

void InitializHead(int n, PNode\* Head, PNode NewNode) {

int i;

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

Head[i] = NULL;

}

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

NewNode = CreateNode(i);

AddFirst(NewNode, i, Head);

}

}

void gener(PNode NewNode, int i) {}

void AddAfter(PNode p, PNode NewNode) {

NewNode->next = p->next;

p->next = NewNode;

}

void AddLast(PNode NewNode, int i, PNode\* Head) {

PNode q = Head[i];

while (q->next) q = q->next;

AddAfter(q, NewNode);

}

void InitStruct(int n, PNode NewNode, PNode\* Head, int\*\* matrix) {

int i, j;

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

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

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

NewNode = CreateNode(j);

AddLast(NewNode, i, Head);

}

}

}

}

void DeleteNode(PNode OldNode, PNode\* Head, int i) {

PNode q = Head[i];

while (q && q->next != OldNode) q = q->next;

if (q == NULL) return;

q->next = OldNode->next;

delete OldNode;

}

void PrintStruct(int n, PNode\* Head) {

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

PNode q = Head[i];

while (q) {

cout << q->items << " ";

q = q->next;

}

cout << endl;

}

}

void InitializMatrix(int\*\* matrix, int n) {

int i, j, ran;

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

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

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

matrix[i][j] = 0;

}

}

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

for (j = i + 1; j < n; j++) {

ran = rand() % 101;

if (ran > 30) {

matrix[i][j] = 1;

matrix[j][i] = matrix[i][j];

}

else {

matrix[i][j] = 0;

matrix[j][i] = matrix[i][j];

}

}

}

}

void PrintMatrix(int n, int\*\* matrix) {

int i, j;

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

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

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

void otogstruct(int n, int min, int max, PNode\* Head1) {

int i;

PNode NewNode;

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

Head1[i] = Head1[i]->next;

int flag = 0;

while (Head1[i]) {

if (Head1[i]->items == min) flag = 1;

if (i == max) {

NewNode = CreateNode(Head1[i]->items);

AddLast(NewNode, min, Head1);

}

else {

if (Head1[i]->items == max && flag == 1) {

DeleteNode(Head1[i], Head1, i);

}

else {

if (Head1[i]->items == max) {

Head1[i]->items = min;

/\*NewNode = CreateNode(min);

AddLast(NewNode, i, Head2);\*/

}

}

}

Head1[i] = Head1[i]->next;

}

}

}

void otog(int\*\* G, int\*\* A, int n, int min, int max) {

int i, j;

int razn = max - min;

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

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

if ((i < max) && (j < max)) { A[i][j] = G[i][j]; }

if ((i <= min) && (j == max)) { A[i][min] += G[i][j]; }

if ((i == max) && (j < min)) { A[min][j] += G[i][j]; }

if ((i < max) && (j > max)) { A[i][j - 1] = G[i][j]; }

if ((j < max) && (i > max)) { A[i - 1][j] = G[i][j]; }

if ((i == min + 1) && (j == max)) { A[i - 1][j - razn + 1] += G[i][j]; }

if ((j == min + 1) && (i == max)) { A[i - razn + 1][j - 1] += G[i][j]; }

}

}

for (i = max + 1; i < n; i++) {

for (j = max + 1; j < n; j++) {

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

}

}

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

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

if (A[i][j] > 1) { A[i][j] = 1; }

}

}

}

void stiag(int\*\* G, int\*\* A, int n, int min, int max) {

int i, j, razn;

razn = max - min;

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

{

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

{

if ((i < max) && (j < max)) A[i][j] = G[i][j];

if ((i < min) && (j == max)) A[i][min] += G[i][j];

if ((i == max) && (j < min)) A[min][j] += G[i][j];

if ((i < max) && (j > max)) A[i][j - 1] = G[i][j];

if ((j < max) && (i > max)) A[i - 1][j] = G[i][j];

if ((i == min + 1) && (j == max)) A[i - 1][j - razn + 1] += G[i][j];

if ((j == min + 1) && (i == max)) A[i - razn + 1][j - 1] += G[i][j];

}

}

for (i = max + 1; i < n; i++) {

for (j = max + 1; j < n; j++) {

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

}

}

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

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

if (A[i][j] > 1) { A[i][j] = 1; }

}

}

}

void raschepstruct(int n, int v, PNode\* Head1, PNode\* Head2) {

int i;

PNode NewNode;

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

Head1[i] = Head1[i]->next;

while (Head1[i]) {

if (Head1[i]->items == v) {

NewNode = CreateNode(n);

AddLast(NewNode, i, Head2);

NewNode = CreateNode(i);

AddLast(NewNode, n, Head2);

}

NewNode = CreateNode(Head1[i]->items);

AddLast(NewNode, i, Head2);

Head1[i] = Head1[i]->next;

}

}

NewNode = CreateNode(v);

AddLast(NewNode, n, Head2);

NewNode = CreateNode(n);

AddLast(NewNode, v, Head2);

}

void raschep(int\*\* G, int\*\* A, int n, int v) {

int i, j;

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

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

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

if (j == v) {

if (i == v) { A[i][n] = 1; A[n][i] = A[i][n]; }

else { A[i][n] = G[i][v]; A[n][i] = G[i][v]; }

}

}

}

}

void fill() {

int\*\* G, \*\* A;

int n, i, j, min, max, flag, v;

PNode NewNode = NULL, \* Head1, \* Head2;

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

cout << "Введите размер матрицы" << endl;

cin >> n;

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

InitializMatrix(G, n);

PrintMatrix(n, G);

cout << endl;

Head1 = (PNode\*)malloc(n \* sizeof(PNode));

InitializHead(n, Head1, NewNode);

InitStruct(n, NewNode, Head1, G);

PrintStruct(n, Head1);

cout << endl;

cout << "Введите тип операции:\n1-отождествление\n2-стягивание\n3-расщепление" << endl;

cin >> flag;

switch (flag)

{

case 1:

cout << "Введите отождествляемые вершины" << endl;

cin >> min >> max;

A = (int\*\*)malloc((n - 1) \* sizeof(int));

InitializMatrix(A, n - 1);

otog(G, A, n, min, max);

PrintMatrix(n - 1, A);

Head2 = (PNode\*)malloc((n + 1) \* sizeof(PNode));

//InitializHead(n - 1, Head2, NewNode);

otogstruct(n, min, max, Head1);

PrintStruct(n - 1, Head1);

break;

case 2:

cout << "Введите стягиваемые вершины" << endl;

cin >> min >> max;

A = (int\*\*)malloc((n - 1) \* sizeof(int));

InitializMatrix(A, n - 1);

if (G[min][max] == 1) {

stiag(G, A, n, min, max);

PrintMatrix(n - 1, A);

}

else { cout << "Невозможно стянуть данные вершины"; }

break;

case 3:

cout << "Введите расщепляемую вершину" << endl;

cin >> v;

A = (int\*\*)malloc((n + 1) \* sizeof(int));

InitializMatrix(A, n + 1);

raschep(G, A, n, v);

PrintMatrix(n + 1, A);

cout << endl;

Head2 = (PNode\*)malloc((n + 1) \* sizeof(PNode));

InitializHead(n + 1, Head2, NewNode);

raschepstruct(n, v, Head1, Head2);

PrintStruct(n + 1, Head2);

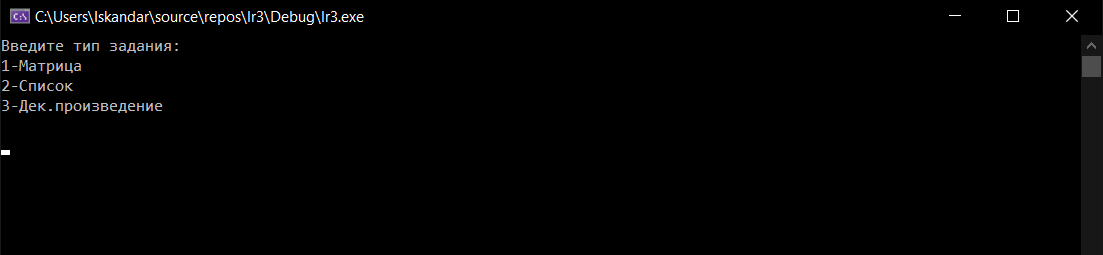
break;

}

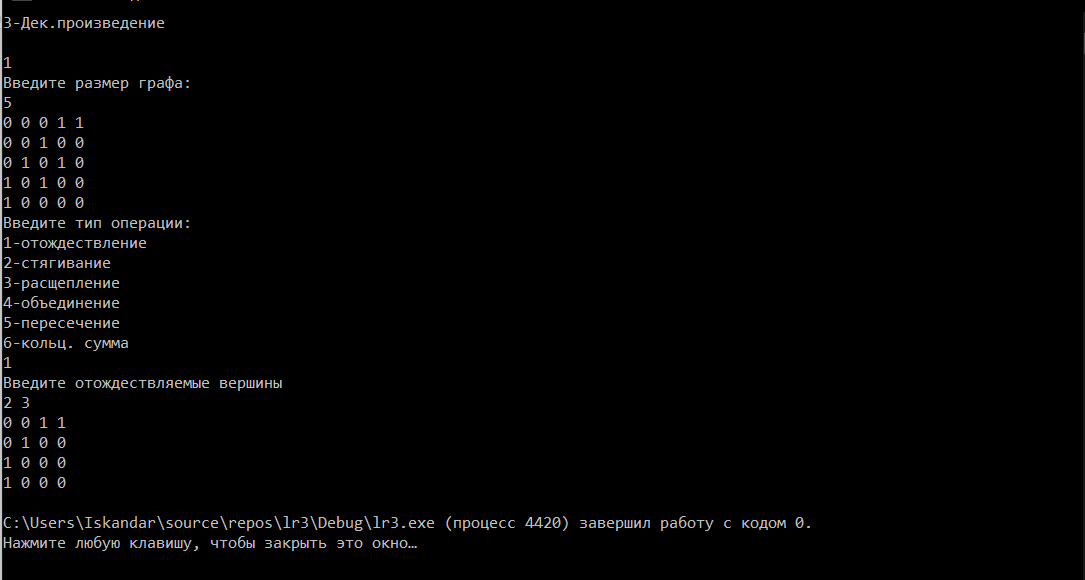
}

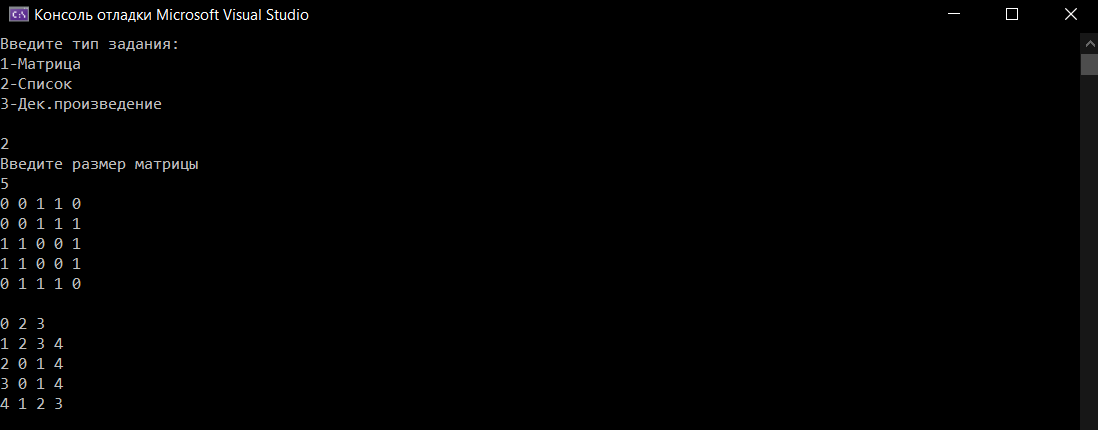
Результаты работы программы:

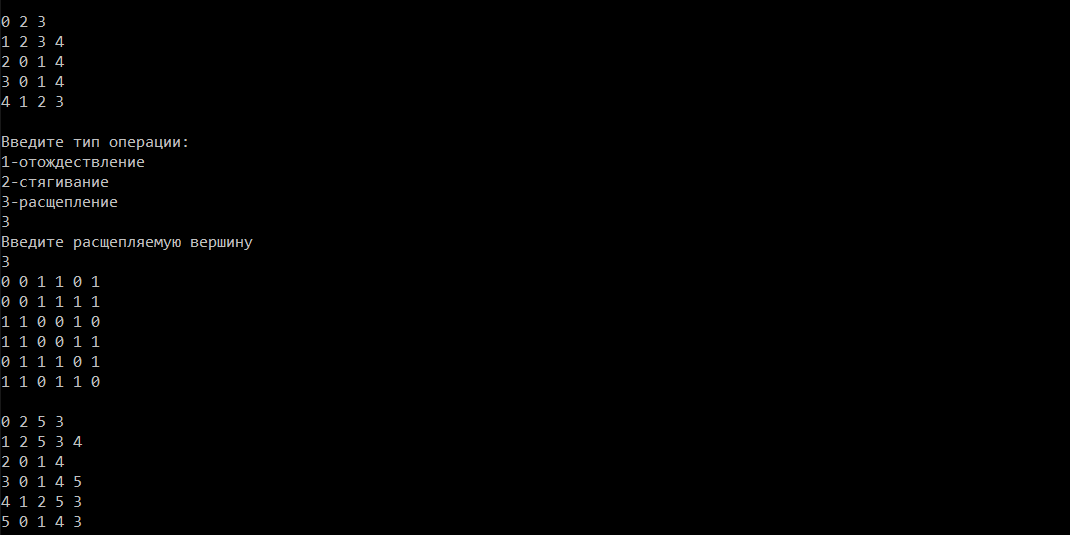
Работа программы с матрицами смежности







Со списками смежности на примере операции расщепления



**Вывод:** научился производить унарные и бинарные операции над графами.