Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему « Унарные и бинарные операции над графами»

Выполнил:

студент группы 20ВВ2

Кочергин В.П.

Принял:

к.т.н. доцент Юрова О.В.

д.т.н. профессор Митрохин М. А.

Пенза 2021

**Цель работы:** Изучить и написать алгоритм к унарным и бинарным операциям.

**Задание 1** Формирование матрицы и списка смежности.

Листинг:

#include <time.h>

#include <malloc.h>

#include <iostream>

#include <windows.h>

using namespace std;

struct Node

{

int items;

Node\* next;

};

typedef Node\* PNode;

PNode\* Head;

void InitializHead(int size)

{

int i;

Head = (PNode\*)malloc(size \* sizeof(PNode));

for (i = 0; i < size; i++) Head[i] = NULL;

}

PNode CreateNode(int items)

{

PNode NewNode = (Node\*)malloc(sizeof(Node));

NewNode->items = items;

NewNode->next = NULL;

return NewNode;

}

void AddFirst(PNode NewNode, int i)

{

NewNode->next = Head[i];

Head[i] = NewNode;

}

void AddAfter(PNode q, PNode NewNode)

{

NewNode->next = q->next;

q->next = NewNode;

}

void AddLast(PNode NewNode, int i)

{

PNode q = Head[i];

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

AddAfter(q, NewNode);

}

void print(int i)

{

PNode q = Head[i];

while (q)

{

printf("%d ", q->items);

q = q->next;

}

printf("\n");

}

int main()

{

int size, i, j, ran, \*\*a;

PNode NewNode = NULL;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("Укажите размер массива: ");

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

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

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

{

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

}

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

{

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

{

a[i][j] = 0;

}

}

srand(time(NULL));

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

{

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

{

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

}

}

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

{

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

{

a[j][i] = a[i][j];

}

}

InitializHead(size);

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

{

NewNode = CreateNode(i);

AddFirst(NewNode, i);

}

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

{

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

{

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

{

NewNode = CreateNode(j);

AddLast(NewNode, i);

}

}

}

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

{

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

{

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

}

printf("\n");

}

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

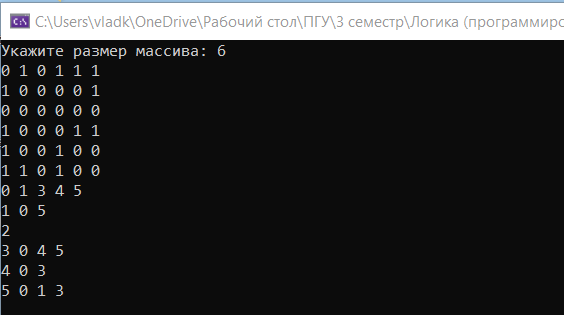
{

print(i);

}

}

Результаты работы программы:



**Задание 2** Унарные операции для матриц

Листинг:

#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;

}

}

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 > 50) {

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, PNode\* Head2) {

int i, \*vis;

PNode NewNode;

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

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

vis[i] = 0;

}

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

if (i < max) {

NewNode = CreateNode(i);

AddFirst(NewNode, i, Head2);

}

else {

NewNode = CreateNode(i - 1);

AddFirst(NewNode, i, Head2);

}

}

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) {

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

else {

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

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

AddLast(NewNode, min, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

AddLast(NewNode, min, Head2);

}

}

}

else {

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

else {

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

NewNode = CreateNode(min);

AddLast(NewNode, i, Head2);

}

else {

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

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

AddLast(NewNode, i, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

AddLast(NewNode, i, Head2);

}

}

}

}

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

}

}

}\*/

void otogstruct(int n, int min, int max, PNode\* Head1, PNode\* Head2) {

int i, \* vis;

PNode NewNode;

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

if (i < max) {

NewNode = CreateNode(i);

AddFirst(NewNode, i, Head2);

}

else {

NewNode = CreateNode(i - 1);

AddFirst(NewNode, i, Head2);

}

}

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) {

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

else {

while (Head1[min]) {

if (Head1[min]->items + 1 == Head1[i]->items) {}

else {

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

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

AddLast(NewNode, min, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

AddLast(NewNode, min, Head2);

}

}

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

}

}

}

else {

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

else {

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

NewNode = CreateNode(min);

AddLast(NewNode, i, Head2);

}

else {

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

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

AddLast(NewNode, i, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

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]; } // верно

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

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

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

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

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

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

}

}

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 stiagstruct(int n, int min, int max, PNode\* Head1, PNode\* Head2) {

int i, \* vis;

PNode NewNode;

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

if (i < max) {

NewNode = CreateNode(i);

AddFirst(NewNode, i, Head2);

}

else {

NewNode = CreateNode(i - 1);

AddFirst(NewNode, i, Head2);

}

}

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) {

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

else {

while (Head1[min]) {

if (Head1[min]->items + 1 == Head1[i]->items) {}

else {

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

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

AddLast(NewNode, min, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

AddLast(NewNode, min, Head2);

}

}

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

}

}

}

else {

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

else {

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

else {

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

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

AddLast(NewNode, i, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

AddLast(NewNode, i, Head2);

}

}

}

}

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

}

}

}

/\*void stiagstruct(int n, int min, int max, PNode\* Head1, PNode\* Head2) {

int i, \* vis;

PNode NewNode;

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

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

vis[i] = 0;

}

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

if (i < max) {

NewNode = CreateNode(i);

AddFirst(NewNode, i, Head2);

}

else {

NewNode = CreateNode(i - 1);

AddFirst(NewNode, i, Head2);

}

}

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) {

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

else {

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

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

AddLast(NewNode, min, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

AddLast(NewNode, min, Head2);

}

}

}

else {

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

else {

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

else {

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

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

AddLast(NewNode, i, Head2);

}

else {

NewNode = CreateNode(Head1[i]->items - 1);

AddLast(NewNode, i, Head2);

}

}

}

}

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

}

}

}\*/

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]; //копирка

if ((i < min) && (j == max)) A[i][min] += G[i][j]; //перемещаение из большего в меньшее

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

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

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];//исключение

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 + 1; i++) {

NewNode = CreateNode(i);

AddFirst(NewNode, i, Head2);

}

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]; }

}

}

}

}

int main() {

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 массива" << 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);

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

NewNode = CreateNode(i);

AddFirst(NewNode, i, Head1);

}

InitStruct(n, NewNode, Head1, G);

PrintStruct(n, Head1);

cout << endl;

cout << "Введите нужно вам действие (1-отождествление, 2-стягивание, 3-расщепление)" << endl;

cin >> flag;

switch (flag)

{

case 1:

cout << "Введите нужные вам 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);

cout << endl;

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

InitializHead(n, Head2, NewNode);

otogstruct(n, min, max, Head1, Head2);

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

if (i == max) {}

else {

while (Head2[i]) {

cout << Head2[i]->items << " ";

Head2[i] = Head2[i]->next;

}

cout << endl;

}

}

break;

case 2:

cout << "Введите нужные вам 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 << "Вы ввели не правильные вершины"; }

cout << endl;

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

InitializHead(n, Head2, NewNode);

stiagstruct(n, min, max, Head1, Head2);

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

if (i == max) {}

else {

while (Head2[i]) {

cout << Head2[i]->items << " ";

Head2[i] = Head2[i]->next;

}

cout << endl;

}

}

break;

case 3:

cout << "Введите нужную вам 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;

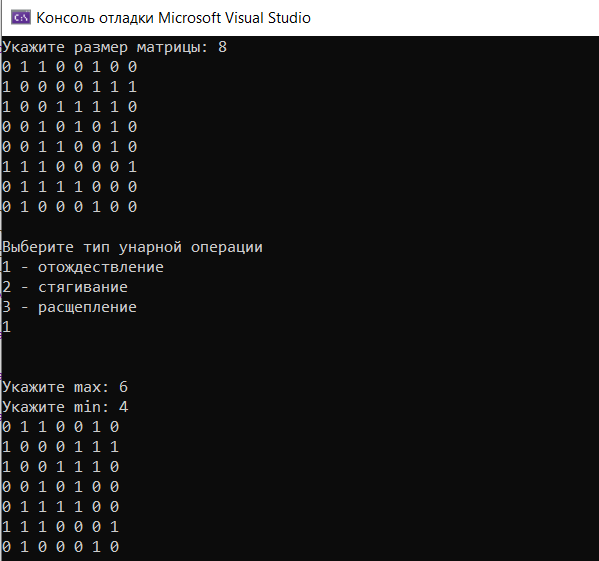
}

return 0;

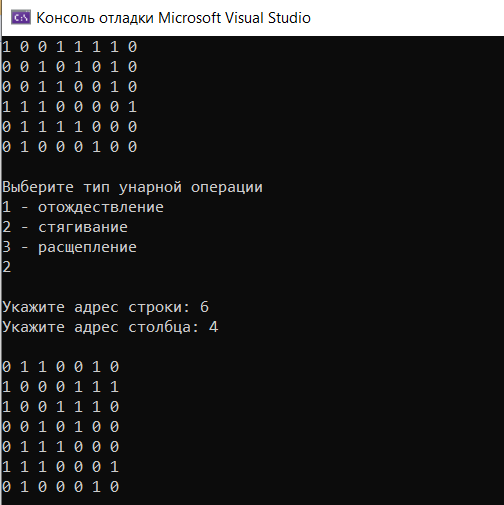
}

Результат работы программы

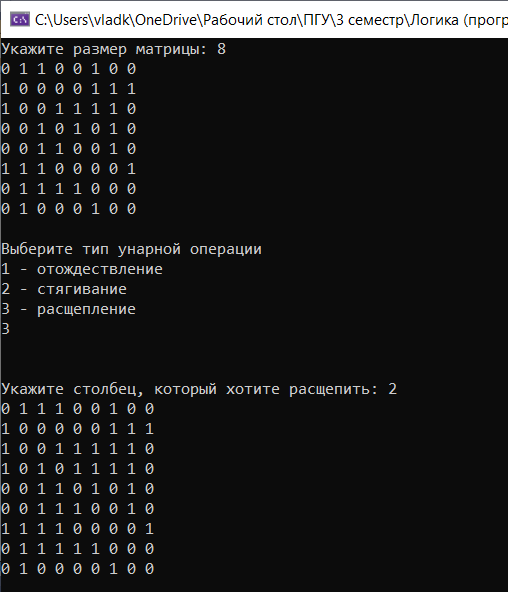
1) Отождествление



2) Стягивание



3) Расщепление



**Задание 3** Бинарные операции для матриц

Листинг:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

void objed(int size1, int size2, int size, int\*\* g, int\*\* a, int\*\* b)

{

int i, j;

for (i = 0; i < size1; i++)

{

for (j = 0; j < size1; j++)

{

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

}

}for (i = 0; i < size2; i++)

{

for (j = 0; j < size2; j++)

{

g[i][j] = g[i][j] + b[i][j];

}

}

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

{

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

{

if (g[i][j] > 1) g[i][j] = 1;

}

}

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

{

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

{

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

}

printf("\n");

}

}

void peres(int minsize, int size, int\*\* g, int\*\* a, int\*\* b)

{

int i, j;

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

{

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

{

g[i][j] = a[i][j] \* b[i][j];

}

}

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

{

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

{

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

}

printf("\n");

}

}

void sum(int minsize, int size, int size1, int size2, int\*\* g, int\*\* a, int\*\* b)

{

int i, j;

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

{

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

{

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

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

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

}

}

if (size1 > size2)

{

for (i = 0; i < size1; i++)

{

for (j = 0; j < size1; j++)

{

if (j >= size2) if ((a[i][j] == 1)) g[i][j] = 1;

if (i >= size2) if ((a[i][j] == 1)) g[i][j] = 1;

}

}

}

else

{

for (i = 0; i < size2; i++)

{

for (j = 0; j < size2; j++)

{

if (j >= size1) if ((b[i][j] == 1)) g[i][j] = 1;

if (i >= size1) if ((b[i][j] == 1)) g[i][j] = 1;

}

}

}

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

{

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

{

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

}

printf("\n");

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setvbuf(stdin, NULL, \_IONBF, 0);

setvbuf(stdout, NULL, \_IONBF, 0);

int i, j, \*\* a, \*\*b, max, \*\* g, size1, size2, size, minsize, min, razn, type;

printf("Укажите размер 1-ой матрицы: ");

scanf\_s("%d", &size1);

printf("Укажите размер 2-ой матрицы: ");

scanf\_s("%d", &size2);

a = (int\*\*)malloc(size1 \* sizeof(int));

b = (int\*\*)malloc(size2 \* sizeof(int));

if (size1 > size2)

{

size = size1;

minsize = size2;

}

else

{

size = size2;

minsize = size1;

}

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

//srand(time(NULL));

for (i = 0; i < size1; i++)

{

a[i] = (int\*)malloc(size1 \* sizeof(int));

}

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

{

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

}

for (i = 0; i < size1; i++)

{

for (j = 0; j < size1; j++)

{

a[i][j] = 0;

}

}

for (i = 0; i < size1; i++)

{

for (j = 1 + i; j < size1; j++)

{

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

}

}

for (i = 0; i < size1; i++)

{

for (j = 0; j < size1; j++)

{

a[j][i] = a[i][j];

}

}

for (i = 0; i < size1; i++)

{

for (j = 0; j < size1; j++)

{

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

}

printf("\n");

}

printf("\n");

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

{

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

{

g[i][j] = 0;

}

}

//srand(time(NULL));

for (i = 0; i < size2; i++)

{

b[i] = (int\*)malloc(size2 \* sizeof(int));

}

for (i = 0; i < size2; i++)

{

for (j = 0; j < size2; j++)

{

b[i][j] = 0;

}

}

for (i = 0; i < size2; i++)

{

for (j = 1 + i; j < size2; j++)

{

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

}

}

for (i = 0; i < size2; i++)

{

for (j = 0; j < size2; j++)

{

b[j][i] = b[i][j];

}

}

for (i = 0; i < size2; i++)

{

for (j = 0; j < size2; j++)

{

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

}

printf("\n");

}

printf("\n");

printf("Выберите тип бинарной операции \n1 - объединение \n2 - пересечение \n3 - кольцевая сумма\n");

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

printf("\n");

switch (type)

{

case 1: objed(size1, size2, size, g, a, b);

break;

case 2: peres(minsize, size, g, a, b);

break;

case 3: sum(minsize, size, size1, size2, g, a, b);

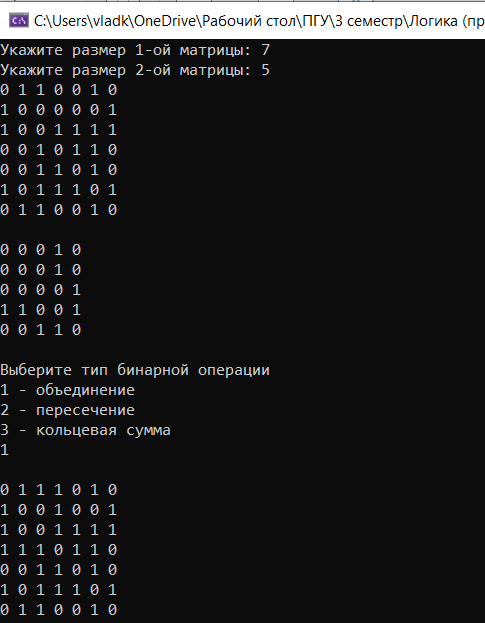
break;

}

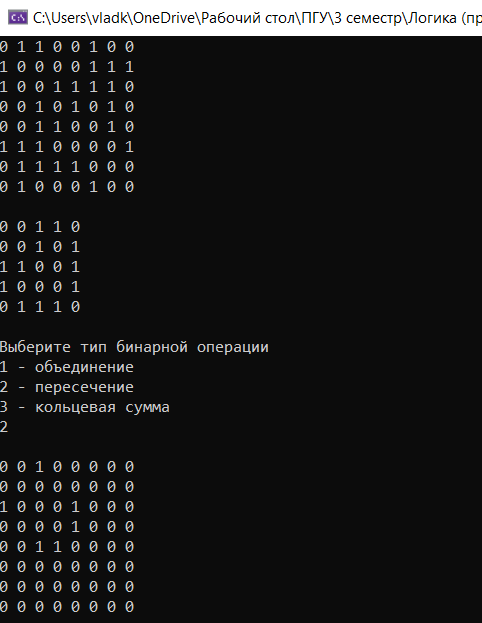
}

Результат работы программы

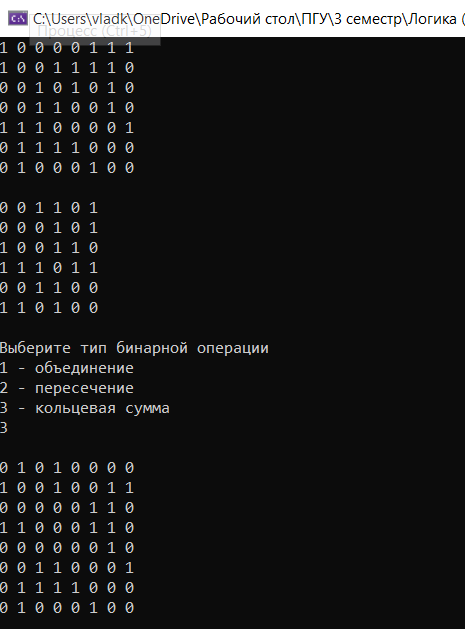
1) Объединение



2) Пересечение



3) Кольцевая сумма



**Задание 4** Декартово произведение.

Листинг:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

using namespace std;

int main()

{

int i, j, size, \*\* a, \*\* g, \*\*A, m, n, ran;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setvbuf(stdin, NULL, \_IONBF, 0);

setvbuf(stdout, NULL, \_IONBF, 0);

printf("Введите размеры массивов ");

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

printf("Матрица смежности:\n");

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

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

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

srand(time(NULL));

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

{

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

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

}

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

{

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

}

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

{

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

{

a[i][j] = 0;

g[i][j] = 0;

}

}

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

{

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

{

A[i][j] = 0;

}

}

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

{

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

{

ran = rand() % 101;

if (ran > 50) a[i][j] = 1;

else a[i][j] = 0;

}

}

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

{

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

{

ran = rand() % 101;

if (ran > 50) g[i][j] = 1;

else g[i][j] = 0;

}

}

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

{

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

{

a[j][i] = a[i][j];

g[j][i] = g[i][j];

}

}

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

{

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

{

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

}

printf("\n");

}

printf("\n");

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

{

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

{

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

}

printf("\n");

}

printf("\n");

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

{

for (i = size\*m; i < size + size\*m; i++)//строка

{

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

{

for (j = size\*n; j < size + size\*n; j++)//столбец

{

if ((m == n) && (i == j))

A[i][j] = 0;

if ((m == n) && (i != j))

A[i][j] = g[i - size \* m][j - size \* n];

if ((m != n) && (i - size \* m == j - size \* n))

A[i][j] = a[m][n];

if ((m != n) && (i - size \* m != j - size \* n))

A[i][j] = 0;

}

}

}

}

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

{

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

{

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

}

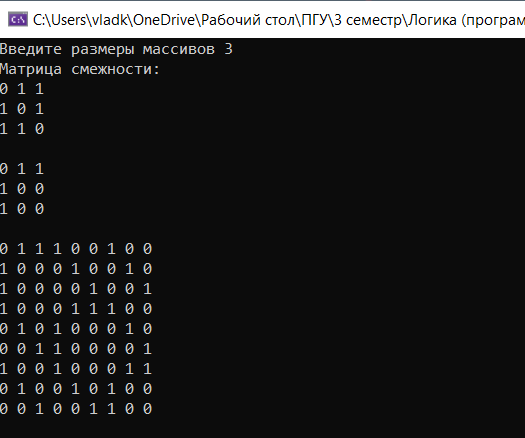
printf("\n");

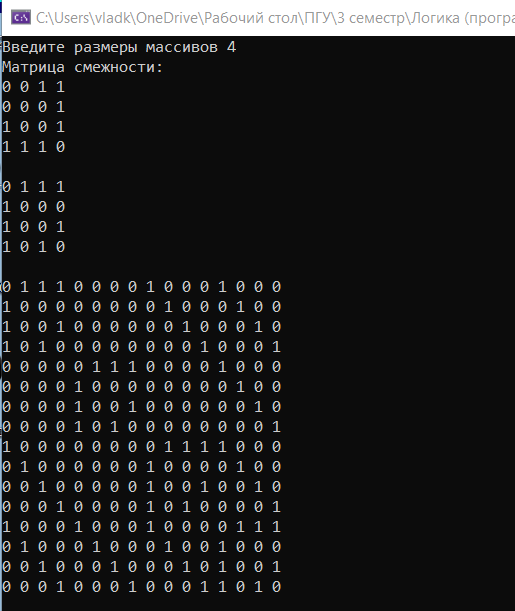
}

printf("\n");

}

Результаты работы программы:





Вывод: В конечном счёте удалось разобраться с данными операциями и написать к ним алгоритм.