Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №6  
по дисциплине: «Унарные и бинарные операции над графами»

Выполнили студенты группы 22ВВВ2:  
Демин М. С.

Захаров А. С.

Амиров И. Р.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Унарные и бинарные операции над графами

**Цель работы**

Разобраться в графах и матрицах. Научиться осуществлять унарные и бинарные операции над графами.

**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.

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 https://lh7-us.googleusercontent.com/2GJ1d7C4kS7FEiZHzYtxAom8GVb2yuY6jxdLTpbWuV21z54rnvhuXLkuG2MwdOrVMQo1KIEpN6L7NHXi2hl6hyL6pClG6F5Q7csponUAsVBFVXXEKMAyefj2MKDocNFhtBn6aHDfbeoUKND0OxMjSA *G*2

б) пересечения  *G* = *G*1 https://lh7-us.googleusercontent.com/HmAJSI_LyXWJJGsXbKjXwYk6_et-OtGHSUryYGFBam0j9e8wtaOVLcL2fkXOVgeY3om_Y-TswXAf7tCNfsqOoOXEGfVw8JlRSfzIOUbOPlF_0QJwNM7Hu67v7RB1kQ3I8VJunvHLrQ7Pvl5j-OS9pQ *G*2

в) кольцевой суммы *G* = *G*1 https://lh7-us.googleusercontent.com/0oA3NKkKTWMoV9tWHzNAgamIvibwZ3NxT-iUhJzyDmVM_Dcec1D-GgHvPpXnTLQ7feIG78DM_YaMgPhJoQGBD-owmHxnZj_NMn4WcSKDZwGfF4R2MiJV_6tezlifKj65lXCxWzMfXH0YEiLVP6HkTw *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.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <locale.h>

#include <windows.h>

typedef struct node {

int vertex;

struct node\* next;

} node;

struct node\* createNode(int v) {

struct node\* newNode = (node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

typedef struct Graph {

int numVertices;

struct node\*\* adjLists;

} Graph;

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = (Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = (node\*\*)malloc(vertices \* sizeof(struct node\*));

int i;

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

graph->adjLists[i] = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph) {

#ifdef NO\_PRINT

return;

#endif

int v;

for (v = 0; v < graph->numVertices; v++) {

struct node\* temp = graph->adjLists[v];

printf("\nvertex: %d\n ", v + 1);

while (temp) {

printf("%d ", temp->vertex + 1);

temp = temp->next;

}

}

}

int\*\* identity(int\*\* matrix, int vertexA, int vertexB, int count) {

if (count == 1) {

printf("deistviye propuscheno\n");

return matrix;

}

// if (!matrix[vertexA][vertexB]) {

// printf("deistviye propuscheno\n");

// return matrix;

// }

int\*\* matrixNew = (int\*\*)calloc(count - 1, sizeof(int\*));

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

matrixNew[i] = (int\*)calloc(count - 1, sizeof(int));

}

int row = 0;

int col = 0;

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

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

if (i != vertexA && i != vertexB && j != vertexA && j != vertexB) {

matrixNew[row][col] = matrix[i][j];

col++;

}

}

if (i != vertexA && i != vertexB) {

col = 0;

row++;

}

}

row = 0;

col = 0;

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

if (i != vertexA && i != vertexB) {

matrixNew[row][count - 2] = matrix[vertexA][i]

|| matrix[vertexB][i];

matrixNew[count - 2][col] = matrix[i][vertexA]

|| matrix[i][vertexB];

row++;

col++;

}

else {

matrixNew[count - 2][count - 2] = matrix[vertexA][vertexB];

}

}

printf("\n");

count--;

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

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

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

}

printf("\n");

}

return matrixNew;

}

int\*\* tightening(int\*\* matrix, int vertexA, int vertexB, int count) {

if (!matrix[vertexA][vertexB]) {

printf("deistviye propuscheno\n");

return matrix;

}

if (matrix[vertexA][vertexA] == 0 && matrix[vertexB][vertexB] == 0

&& matrix[vertexA][vertexB] == 1) {

matrix[vertexA][vertexB] = 0;

matrix[vertexB][vertexA] = 0;

}

matrix = identity(matrix, vertexA, vertexB, count);

return matrix;

}

int\*\* splitting(int\*\* matrix, int vertex, int count) {

count++;

int\*\* matrixNew = (int\*\*)calloc(count, sizeof(int\*));

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

matrixNew[i] = (int\*)calloc(count, sizeof(int));

// скопировать верхний левый угол до расщепленной части

for (int y = 0; y < vertex; y++)

for (int x = 0; x < vertex; x++)

matrixNew[y][x] = matrix[y][x];

// скопировать верхний правый угол после расщепленной части

for (int y = 0; y < vertex; y++)

for (int x = vertex + 1; x < count - 1; x++) // пропуская эту часть + старая матрица меньше

matrixNew[y][x - 1] = matrix[y][x]; // смещение чтоб не было пробела

// скопировать нижний левый угол после расщепленной части

for (int y = vertex + 1; y < count - 1; y++) // пропуская эту часть + старая матрица меньше

for (int x = 0; x < vertex; x++)

matrixNew[y - 1][x] = matrix[y][x]; // смещение чтоб не было пробела

// скопировать нижний правый угол после расщепленной части

for (int y = vertex + 1; y < count - 1; y++) // пропуская эту часть + старая матрица меньше

for (int x = vertex + 1; x < count - 1; x++) // пропуская эту часть + старая матрица меньше

matrixNew[y - 1][x - 1] = matrix[y][x]; // смещение чтоб не было пробела

// скопировать нижний правый угол после расщепленной части

for (int y = vertex + 1; y < count - 1; y++) // пропуская эту часть + старая матрица меньше

for (int x = vertex + 1; x < count - 1; x++) // пропуская эту часть + старая матрица меньше

matrixNew[y - 1][x - 1] = matrix[y][x]; // смещение чтоб не было пробела

// скопировать вертикальную часть в последние два столбца

for (int y = 0; y < count - 1; y++) {

matrixNew[y][count - 1] = matrix[y][vertex];

matrixNew[y][count - 2] = matrix[y][vertex];

}

// скопировать горизонтальную часть в последние две строки

for (int x = 0; x < count - 1; x++) {

matrixNew[count - 1][x] = matrix[vertex][x];

matrixNew[count - 1][x] = matrix[vertex][x];

}

if (matrixNew[0][0] && count == 2)

matrixNew[1][1] = 1;

else {

matrixNew[count - 1][count - 1] = 0;

matrixNew[count - 2][count - 2] = 0;

}

matrixNew[count - 1][count - 2] = 1;

matrixNew[count - 2][count - 1] = 1;

return matrixNew;

}

int\*\* unione(int\*\* matrix1, int\*\* matrix2, int size1, int size2, int\* sizeNew) {

int maxx = size1 > size2 ? size1 : size2;

int minn = maxx == size2 ? size1 : size2;

\*sizeNew = maxx;

bool m1smol = minn == size1;

int\*\* matrixNew = (int\*\*)calloc(maxx, sizeof(int\*));

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

matrixNew[i] = (int\*)calloc(maxx, sizeof(int));

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

for (int i2 = 0; i2 < maxx; i2++)

if (i >= minn || i2 >= minn)

matrixNew[i][i2] = m1smol ? matrix2[i][i2] : matrix1[i][i2];

else

matrixNew[i][i2] = matrix1[i][i2] + matrix2[i][i2] > 0;

return matrixNew;

}

int\*\* intersection(int\*\* matrix1, int\*\* matrix2, int size1, int size2,

int\* sizeNew) {

int maxx = size1 > size2 ? size1 : size2;

int minn = maxx == size2 ? size1 : size2;

\*sizeNew = minn;

int\*\* matrixNew = (int\*\*)calloc(minn, sizeof(int\*));

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

matrixNew[i] = (int\*)calloc(minn, sizeof(int));

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

for (int i2 = 0; i2 < minn; i2++)

matrixNew[i][i2] = matrix1[i][i2] \* matrix2[i][i2] > 0;

return matrixNew;

}

int\*\* rmv(int\*\* matrix, int\* size, int r) {

int s = \*size;

int\*\* nMatrix = (int\*\*)calloc(s, sizeof(int\*));

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

nMatrix[i] = (int\*)calloc(s, sizeof(int));

int di = 0, dj = 0;

for (size\_t i = 0; i < s; i++) {

if (i == r)

i++;

if (i >= s)

break;

for (size\_t j = 0; j < s; j++) {

if (j == r)

j++;

if (j >= s)

break;

nMatrix[di][dj] = matrix[i][j];

dj++;

}

di++;

dj = 0;

}

\*size = s - 1;

return nMatrix;

}

int\*\* roundsum(int\*\* matrix1, int\*\* matrix2, int size1, int size2, int\* outSize) {

int\*\* bMatrix = (size1 > size2) ? matrix1 : matrix2;

int lSize = (size1 > size2) ? size2 : size1;

int\*\* lMatrix = (size1 > size2) ? matrix2 : matrix1;

int\*\* matrixNew = (int\*\*)calloc(lSize, sizeof(int\*));

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

matrixNew[i] = (int\*)calloc(lSize, sizeof(int));

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

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

int res = lMatrix[i][j] == bMatrix[i][j] ? 0 : 1;

matrixNew[i][j] = res;

}

}

int rsSize = lSize;

int\*\* rsMatrix = matrixNew;

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

int c = 0;

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

if (matrixNew[i][j] == 0)

c++;

}

if (c == lSize)

rsMatrix = rmv(rsMatrix, &rsSize, i);

}

\*outSize = rsSize;

return rsMatrix;

}

void cleanstdin() {

fseek(stdin, 0, SEEK\_END);

}

Graph\* fillgraph(int\*\* matrix, int count) {

struct Graph\* graph2 = createGraph(count);

bool edge;

srand(time(NULL));

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

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

edge = matrix[i][j];

if (edge == true) {

addEdge(graph2, i, j);

}

}

}

return graph2;

}

int\*\* mult(int\*\* matrix1, int\*\* matrix2, int size1, int size2, int\* sizeOut) {

\*sizeOut = size1 \* size2;

int s = \*sizeOut;

int\*\* matrixNew = (int\*\*)calloc(s, sizeof(int\*));

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

matrixNew[i] = (int\*)calloc(s, sizeof(int));

int bSize = (size1 > size2) ? size1 : size2;

int\*\* bMatrix = (size1 > size2) ? matrix1 : matrix2;

int lSize = (size1 > size2) ? size2 : size1;

int\*\* lMatrix = (size1 > size2) ? matrix2 : matrix1;

int z1 = 0, v1 = 0;

int z2 = 0, v2 = 0;

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

if (i / bSize != z1)

z1++;

if (z1 >= lSize)

z1 = 0;

if (v1 >= bSize)

v1 = 0;

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

if (j / bSize != z2)

z2++;

if (v2 >= bSize)

v2 = 0;

if (z1 == z2 && bMatrix[v1][v2] == 1)

matrixNew[i][j] = 1;

else if (v1 == v2 && lMatrix[z1][z2] == 1)

matrixNew[i][j] = 1;

v2 += 1;

}

z2 = 0;

v2 = 0;

v1 += 1;

}

return matrixNew;

}

int main() {

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setbuf(stdout, NULL);

int count1 = 0;

while (count1 < 1 || count1 > 5000) {

printf("Vvedite razmer pervogo grafa:");

cleanstdin();

scanf("%d", &count1);

}

int count2 = 0;

while (count2 < 1 || count2 > 5000) {

printf("Vvedite razmer vtorogo grafa: ");

cleanstdin();

scanf("%d", &count2);

}

srand(time(NULL));

int\*\* matrix1 = (int\*\*)malloc(count1 \* sizeof(int\*));

for (int i = 0; i < count1; i++) {

matrix1[i] = (int\*)malloc(count1 \* sizeof(int));

}

int\*\* matrix2 = (int\*\*)malloc(count2 \* sizeof(int\*));

for (int i = 0; i < count2; i++) {

matrix2[i] = (int\*)malloc(count2 \* sizeof(int));

}

for (int i = 0; i < count1; i++) {

for (int j = 0; j < count1; j++) {

matrix1[j][i] = 0;

}

}

for (int i = 0; i < count2; i++) {

for (int j = 0; j < count2; j++) {

matrix2[j][i] = 0;

}

}

for (int i = 0; i < count1; i++) {

for (int j = i + 1; j < count1; j++) {

int random = rand() % 2; // Генерация случайного числа 0 или 1

matrix1[i][j] = random;

matrix1[j][i] = random; // Заполнение элементов и их симметричных пар

}

}

for (int i = 0; i < count2; i++) {

for (int j = i + 1; j < count2; j++) {

int random = rand() % 2; // Генерация случайного числа 0 или 1

matrix2[i][j] = random;

matrix2[j][i] = random; // Заполнение элементов и их симметричных пар

}

}

struct Graph\* graph1 = NULL;

printf("\n");

//Задание 2.1

int vertexA, vertexB, vertexSpl;

bool ready = true;

while (ready) {

cleanstdin();

graph1 = fillgraph(matrix1, count1);

printGraph(graph1);

printf("\n");

printf("\n\n");

for (int i = 0; i < count1; i++) {

for (int j = 0; j < count1; j++) {

printf("%d ", matrix1[i][j]); /\*52\*/

}

printf("\n");

}

printf("\n");

for (int i = 0; i < count2; i++) {

for (int j = 0; j < count2; j++) {

printf("%d ", matrix2[i][j]);

}

printf("\n");

}

printf("\nVyberete kakuyu operaciyu nuzhno sdelat':\n");

printf(

"0. Vyhod.\n1. Otozhdestvlenie vershin.\n2. Styagivanie rebra.\n3. Rasshcheplenie vershin.\n4. Objedineniya grafof.\n5. Peresecheniya grafof.\n6. Kol'cevaya summa grafof.\n7. Dekartovo proizvedeniye grafof.\n\n");

printf("\nDlya vybora vvedite cifru ot 1 do 7 sootvetstvenno\n");

int input;

cleanstdin();

scanf("%d", &input);

int\*\* new\_ = NULL;

int maxx = 0;

int\*\* matrixnew = NULL;

int sizenew = 0;

switch (input) {

case 0:

ready = false;

break;

case 1:

// maxx = count1 < count2 ? count1 : count2;

printf("\nVvedite nuzhnye vershiny [1-%d] [1-%d]: ", count1,

count2);

cleanstdin();

scanf("%d %d", &vertexA, &vertexB);

if (vertexA < 1 || vertexA > count1 || vertexB < 1

|| vertexB > count1) {

printf("Vvedite eshche raz");

break;

}

vertexA--;

vertexB--;

new\_ = identity(matrix1, vertexA, vertexB, count1);

if (new\_ != matrix1) {

free(matrix1);

matrix1 = new\_;

count1--;

}

// new\_ = identity(matrix2, vertexA, vertexB, count2);

// if (new\_ != matrix2) {

// free(matrix2);

// matrix2 = new\_;

// count2--;

// }

break;

case 2:

// maxx = count1 < count2 ? count1 : count2;

printf(

"\nVvedite vershiny mezhdu kotorymi styanut' rebro [1-%d] [1-%d]: ",

count1, count1);

cleanstdin();

scanf("%d %d", &vertexA, &vertexB);

if (vertexA < 1 || vertexA > count1 || vertexB < 1

|| vertexB > count1) {

printf("Vvedite eshche raz");

break;

}

vertexA--;

vertexB--;

new\_ = tightening(matrix1, vertexA, vertexB, count1);

if (new\_ != matrix1) {

free(matrix1);

count1--;

matrix1 = new\_;

}

// new\_ = tightening(matrix2, vertexA, vertexB, count2);

// if (new\_ != matrix2) {

// free(matrix2);

// count2--;

// matrix2 = new\_;

// }

break;

case 3:

printf("\nVvedite vershine kotoruyu hotite rasshchepit' [1-%d]: ",

count1);

cleanstdin();

scanf("%d", &vertexSpl);

if (vertexSpl < 1 || vertexSpl > count1) {

printf("Vvedite eshche raz");

break;

}

vertexSpl--;

new\_ = splitting(matrix1, vertexSpl, count1);

if (new\_ != matrix1) {

free(matrix1);

matrix1 = new\_;

count1++;

}

// new\_ = splitting(matrix2, vertexSpl, count2);

// if (new\_ != matrix2) {

// free(matrix2);

// matrix2 = new\_;

// count2++;

// }

break;

case 4:

printf("\nMatrix Union: ");

sizenew = 0;

matrixnew = unione(matrix1, matrix2, count1, count2, &sizenew);

printf("union ok\n----\n");

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

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

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

}

printf("\n");

}

printf("-----");

break;

case 5:

printf("\nMatrix intersection: ");

sizenew = 0;

matrixnew = intersection(matrix1, matrix2, count1, count2,

&sizenew);

printf("intersection ok\n----\n");

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

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

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

}

printf("\n");

}

printf("-----");

break;

case 6:

printf("\nMatrix round sum: ");

sizenew = 0;

matrixnew = roundsum(matrix1, matrix2, count1, count2, &sizenew);

printf("ok\n----\n");

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

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

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

}

printf("\n");

}

printf("-----");

break;

case 7:

printf("\nMatrix multiply: ");

sizenew = 0;

matrixnew = roundsum(matrix1, matrix2, count1, count2, &sizenew);

printf("ok\n----\n");

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

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

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

}

printf("\n");

}

printf("-----");

break;

default:

printf("Vvedite eshche raz");

break;

}

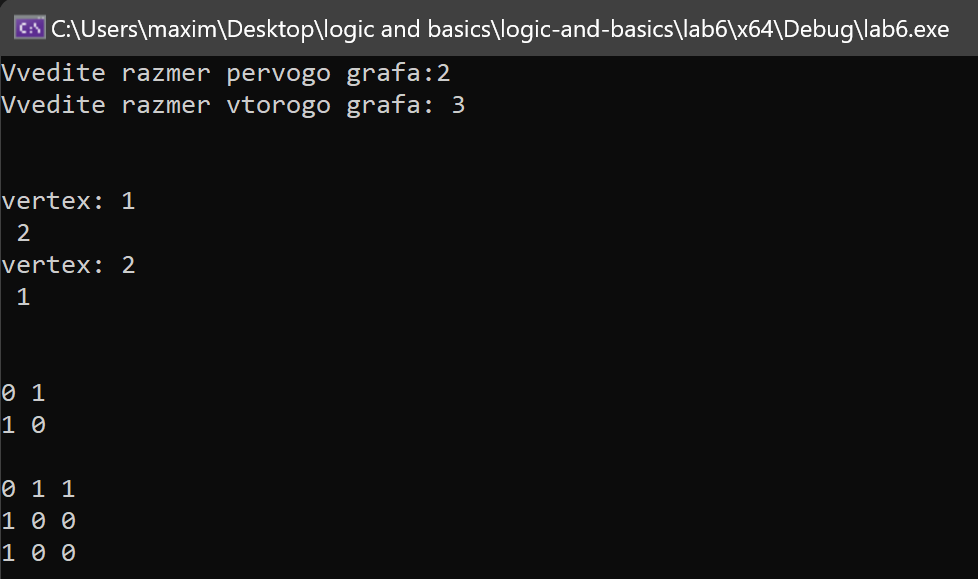
cleanstdin();

}

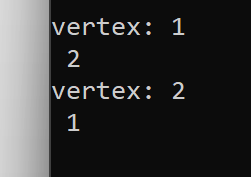
}

**Задания**

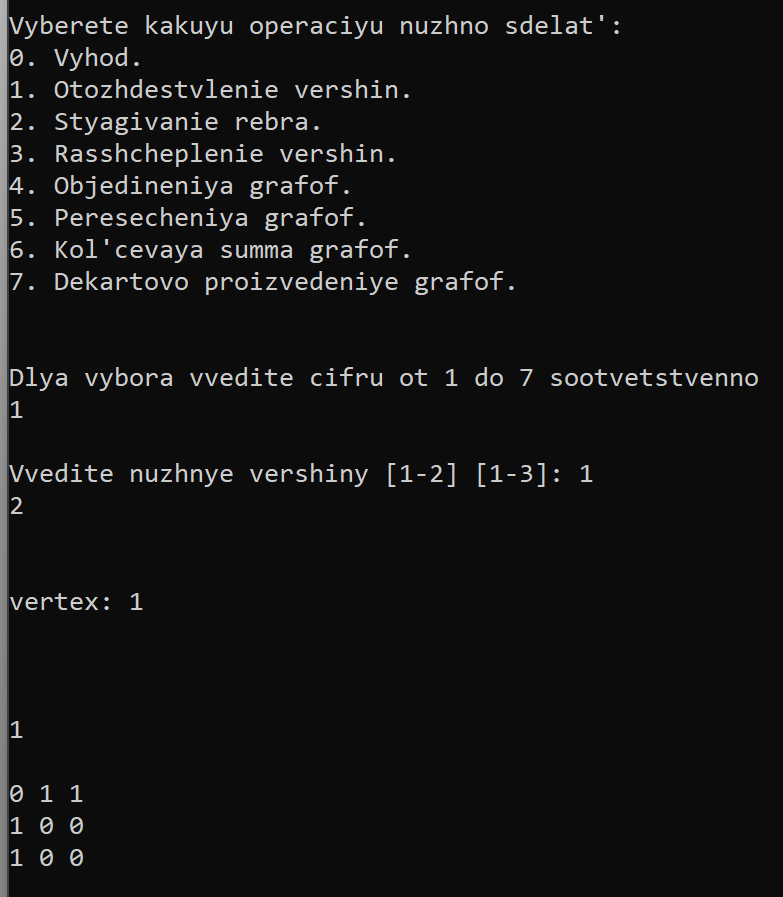
**1.1 Создания матриц**

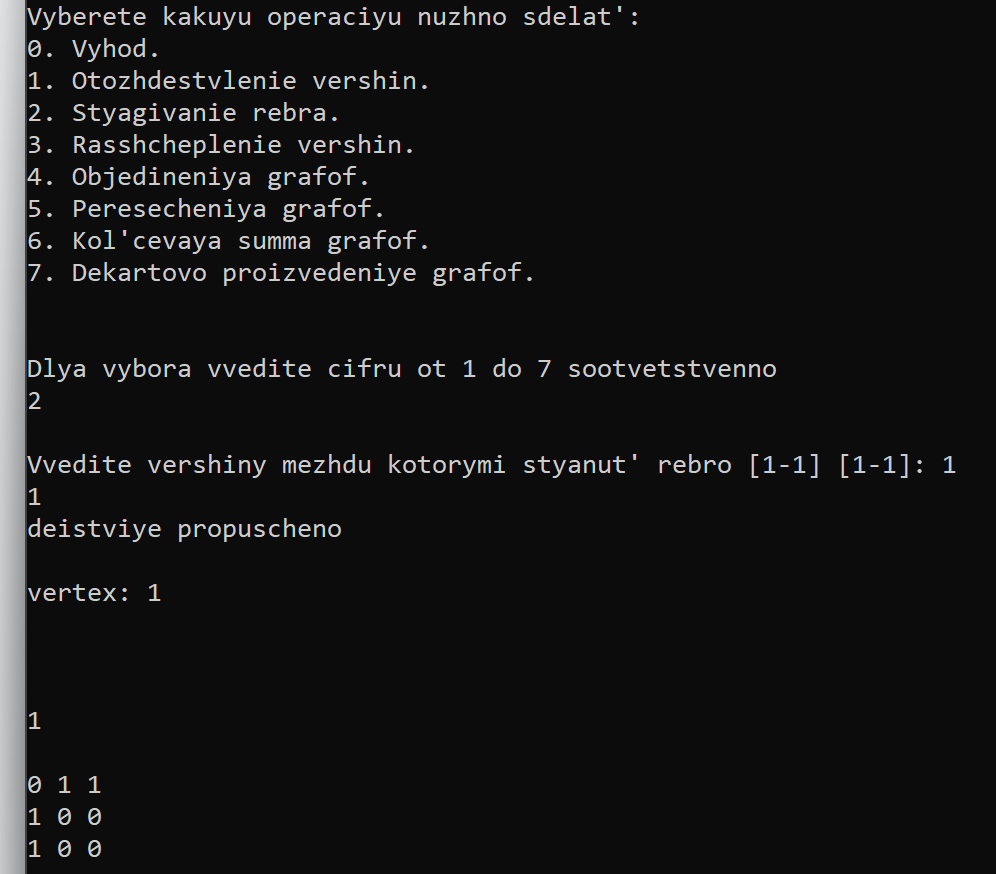
****

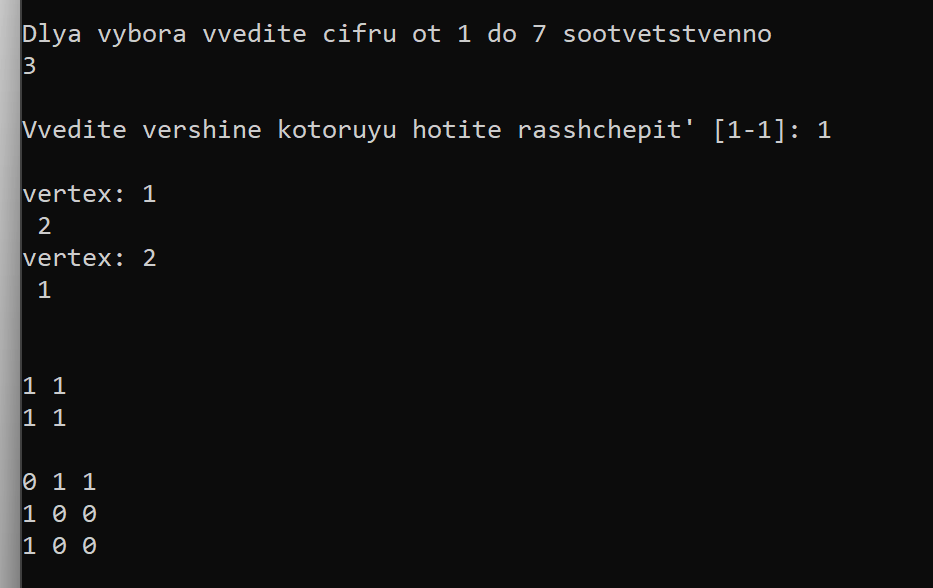
**1.2 Создание списков смежности**

****

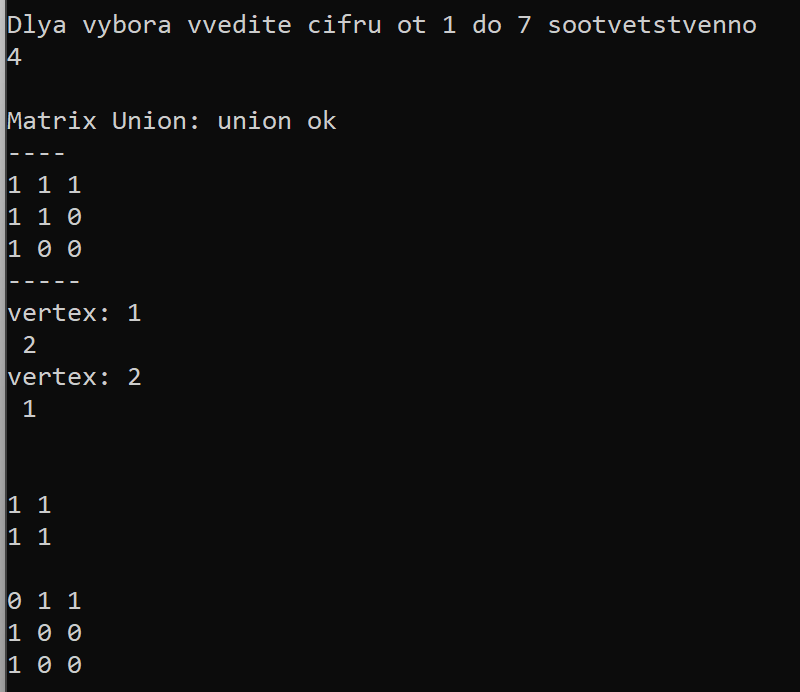
**2.1 Выполнение операций**

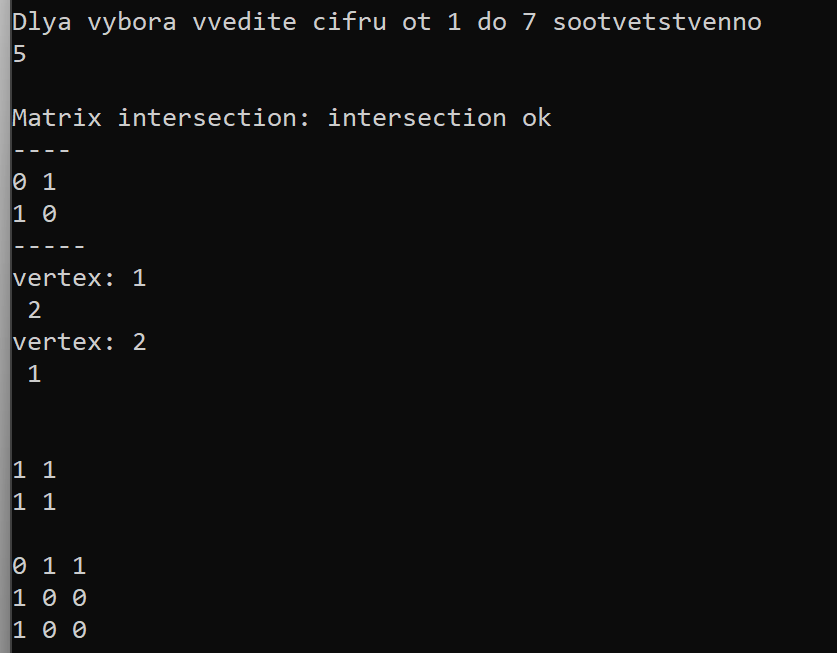
****

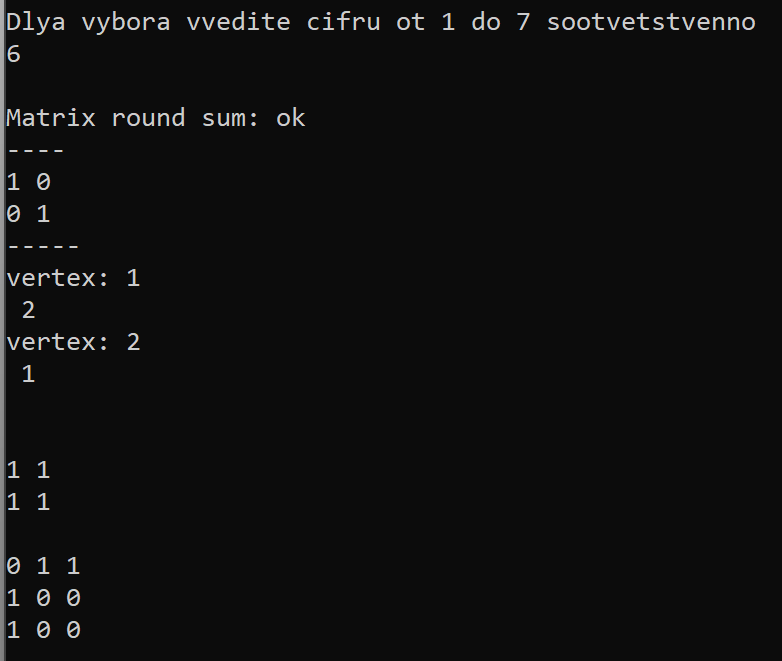
****

****

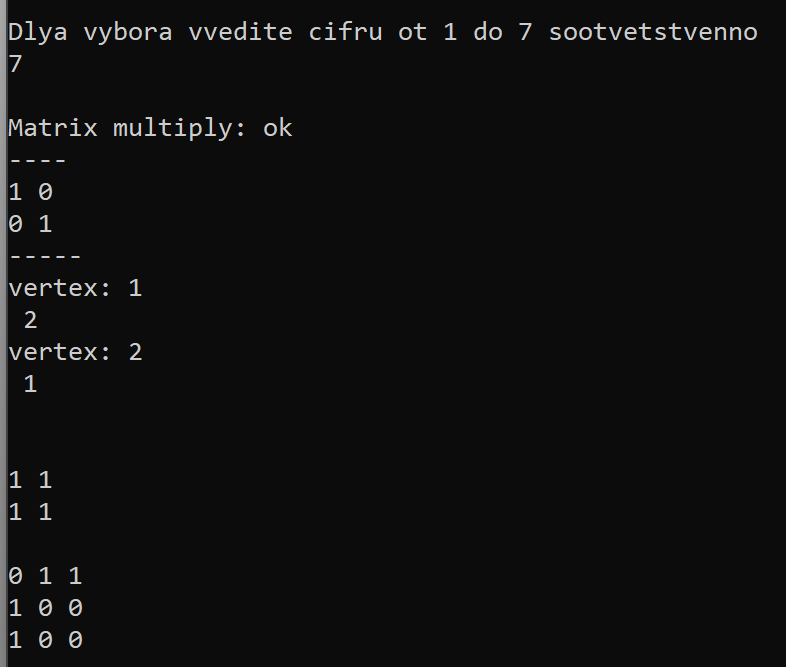
**3**

****

****

****

**4**

****

**Вывод**

Разобрались в графах и матрицах. Научились осуществлять унарные и бинарные операции над графами.