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 графом»

Выполнили студенты группы 22ВВВ2:  
Беляев Д.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Унарные и бинарные операции над графом

**Цель работы**

Научиться выполнять унарные и бинарные операции над графом

**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://lh5.googleusercontent.com/S2M9w3d26l5Kn6EZbmbotVu7eny7UcuqYr3MHL05okoEwyqBfT7TMPsxuq9XgRsrD8MHW3L6Ab-pncNRhAiDVqqC6-wa8a_DL9X5_YjjeMVues6aj3KcuzX7QIdHHMYS0Gu91nQRrsmfiqyEFgbFNQ *G*2

б) пересечения  *G* = *G*1 https://lh4.googleusercontent.com/WrEjS_XnAIV-B6ckYQdyOZuJjwzsUVdOvcThfXlWuB5j9pkVqYOUuFpvZQXDuMnXdcJfbp98w5YEDkdXgRutRx0nZx2gtWPmVoW-k1sGHtGY5lpvK5u81DwBg0LcqQFW1NZlxw6lKcNFxF2oiBixNQ *G*2

в) кольцевой суммы *G* = *G*1 https://lh3.googleusercontent.com/5gdefpNOXvGYnUlD-lQx_cci1td2ZCCfzfuCMmUaAmAhTC1SjlPDjYmgfUkyI-4U_hZoYAPrtI5EuwQNA28p0mtxJgjJNyh07MKAdcsWMJ90WnAhlcH27IdLWYbNcqCpdQ7-abtQKFa0CTIwFcqugQ *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**

* **main.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "list.h"

int\*\* GenerateMatrix(int size\_x, int size\_y);

void ClearMatrix(int\*\* matrix, int size);

void PrintMatrix(int\*\* matrix, int size\_x, int size\_y);

void PrintListMatrix(DataType\*\* matrix, int size);

int\*\* GenerateAdjacencyMatrix(int size);

DataType\*\* GenerateListMatrix(int\*\* matrix, int size);

int\*\* Identification(int\*\* matrix, int\* size, int x, int y);

int\*\* Tightening(int\*\* matrix, int\* size, int x, int y);

int\*\* Splitting(int\*\* matrix, int\* size, int x, int y);

DataType\*\* IdentificationL(DataType\*\* matrix, int\* size, int x, int y);

DataType\*\* TighteningL(DataType\*\* matrix, int\* size, int x, int y);

DataType\*\* SplittingL(DataType\*\* matrix, int\* size, int x, int y);

int\*\* Union(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* outSize);

int\*\* Intersection(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* outSize);

int\*\* RoundSum(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* outSize);

int\*\* Multiplication(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* sizeOut);

int main()

{

srand(time(NULL));

int matrixSize1 = 0;

printf("Insert size of first matrix: ");

scanf("%d", &matrixSize1);

int\*\* matrix1 = GenerateAdjacencyMatrix(matrixSize1);

printf("--- Matrix 1 ---\n");

PrintMatrix(matrix1, matrixSize1, matrixSize1);

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

int matrixSize2 = 0;

printf("Insert size of second matrix: ");

scanf("%d", &matrixSize2);

int\*\* matrix2 = GenerateAdjacencyMatrix(matrixSize2);

printf("--- Matrix 2 ---\n");

PrintMatrix(matrix2, matrixSize2, matrixSize2);

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

int matrixSizeL1 = matrixSize1;

int matrixSizeL2 = matrixSize2;

DataType\*\* matrixL1 = GenerateListMatrix(matrix1, matrixSizeL1);

DataType\*\* matrixL2 = GenerateListMatrix(matrix2, matrixSizeL2);

printf("--- List Matrix 1---\n");

PrintListMatrix(matrixL1, matrixSizeL1);

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

printf("--- List Matrix 2---\n");

PrintListMatrix(matrixL2, matrixSizeL2);

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

int answ = -1;

while (answ != 0)

{

printf("Choose Action\n1 - Identification\n2 - Tightening\n3 - Splitting\n");

printf("4 - Identification list\n5 - Tightening list\n6 - Splitting list\n7 - Union\n");

printf("8 - Intersection\n9 - RoundSum\n10 - Multiplication\n0 - exit\n");

scanf("%d", &answ);

system("cls");

switch (answ)

{

case 1:

printf("Identification matrix2. Row = 0, col = 3\n");

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

matrix2 = Identification(matrix2, &matrixSize2, 0, 3);

PrintMatrix(matrix2, matrixSize2, matrixSize2);

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

break;

case 2:

printf("Tightening matrix2. Row = 0, col = 1\n");

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

matrix2 = Tightening(matrix2, &matrixSize2, 0, 1);

PrintMatrix(matrix2, matrixSize2, matrixSize2);

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

break;

case 3:

printf("Splitting matrix2. Row = 0, col = 1\n");

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

matrix2 = Splitting(matrix2, &matrixSize2, 0, 1);

PrintMatrix(matrix2, matrixSize2, matrixSize2);

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

break;

case 4:

printf("Identification matrixList2. Row = 0, col = 3\n");

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

matrixL2 = IdentificationL(matrixL2, &matrixSize2, 0, 3);

PrintListMatrix(matrixL2, matrixSize2);

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

break;

case 5:

printf("Tightening matrixList2. Row = 0, col = 1\n");

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

matrixL2 = TighteningL(matrixL2, &matrixSize2, 0, 1);

PrintListMatrix(matrixL2, matrixSize2);

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

break;

case 6:

printf("Splitting matrixList2. Row = 0, col = 1\n");

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

matrixL2 = SplittingL(matrixL2, &matrixSize2, 0, 1);

PrintListMatrix(matrixL2, matrixSize2);

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

break;

case 7:

{

printf("Union matrix1 and matrix2\n");

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

int uSize = 0;

int\*\* uMatrix = Union(matrix1, matrixSize1, matrix2, matrixSize2, &uSize);

PrintMatrix(uMatrix, uSize, uSize);

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

break;

}

case 8:

{

printf("Intersection matrix1 and matrix2\n");

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

int iSize = 0;

int\*\* iMatrix = Intersection(matrix1, matrixSize1, matrix2, matrixSize2, &iSize);

PrintMatrix(iMatrix, iSize, iSize);

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

break;

}

case 9:

{

printf("Round sum matrix1 and matrix2\n");

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

int rsSize = 0;

int\*\* rsMatrix = RoundSum(matrix1, matrixSize1, matrix2, matrixSize2, &rsSize);

PrintMatrix(rsMatrix, rsSize, rsSize);

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

break;

}

case 10:

{

printf("Multiplication matrix1 and matrix2\n");

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

int mSize = 0;

int\*\* mMatrix = Multiplication(matrix1, matrixSize1, matrix2, matrixSize2, &mSize);

PrintMatrix(mMatrix, mSize, mSize);

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

break;

}

default:

break;

}

}

return 0;

}

#pragma region Matrix work

int\*\* GenerateMatrix(int size\_x, int size\_y)

{

int\*\* matrix = (int\*\*)malloc(size\_x \* sizeof(int\*));

if (matrix == NULL)

return NULL;

for (int i = 0; i < size\_x; i++)

{

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

if (matrix[i] == NULL)

return NULL;

}

for (int i = 0; i < size\_x; i++)

{

for (int j = 0; j < size\_y; j++)

{

matrix[i][j] = 0;

}

}

return matrix;

}

void ClearMatrix(int\*\* matrix, int size)

{

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

{

free(matrix[i]);

}

free(matrix);

}

void PrintMatrix(int\*\* matrix, int size\_x, int size\_y)

{

for (int i = 0; i < size\_x; i++)

{

for (int j = 0; j < size\_y; j++)

{

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

}

printf("\n");

}

}

void PrintListMatrix(DataType\*\* matrix, int size)

{

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

{

PrintDataType(matrix[i]);

}

}

int\*\* RemoveVertex(int\*\* matrix, int\* size, int r)

{

int s = \*size;

int\*\* nMatrix = GenerateMatrix(s - 1, s - 1);

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;

}

#pragma endregion

#pragma region First number

int\*\* GenerateAdjacencyMatrix(int size)

{

int\*\* matrix = GenerateMatrix(size, size);

if (matrix == NULL)

return NULL;

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

{

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

{

if (i == j)

continue;

int rnd = rand() % 2;

int izolate = rand() % 2;

matrix[i][j] = rnd;

if (rnd == 1)

matrix[j][i] = 1;

}

}

return matrix;

}

DataType\*\* GenerateListMatrix(int\*\* matrix, int size)

{

DataType\*\* matrixL = (DataType\*\*)malloc(sizeof(DataType\*) \* size);

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

matrixL[i] = CreateDataType(i);

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

{

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

{

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

continue;

Add(j, matrixL[i]);

}

}

return matrixL;

}

#pragma endregion

#pragma region Second Number

int\*\* Identification(int\*\* matrix, int\* size, int x, int y)

{

int s = \*size;

if (y < x)

{

int t = x;

x = y;

y = t;

}

int\*\* nMatrix = GenerateMatrix(s - 1, s - 1);

int di = 0, dj = 0;

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

{

if (i == x || i == y)

i++;

if (i == x || i == y)

i++;

if (i >= s)

break;

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

{

if (j == x || j == y)

j++;

if (j == x || j == y)

j++;

if (j >= s)

break;

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

dj++;

}

di++;

dj = 0;

}

int d = 0;

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

{

if (i == x || i == y)

i++;

if (i == x || i == y)

i++;

if (i >= s)

break;

if (matrix[x][i] == 1)

{

nMatrix[s - 2][d] = 1;

nMatrix[d][s - 2] = 1;

}

if (matrix[y][i] == 1)

{

nMatrix[s - 2][d] = 1;

nMatrix[d][s - 2] = 1;

}

d++;

}

\*size = \*size - 1;

return nMatrix;

}

int\*\* Tightening(int\*\* matrix, int\* size, int x, int y)

{

if (matrix[x][y] != 1)

return matrix;

return Identification(matrix, size, x, y);

}

int\*\* Splitting(int\*\* matrix, int\* size, int x, int y)

{

if (matrix[x][y] != 1)

return matrix;

int s = \*size;

matrix = (int\*\*)realloc(matrix, (s + 1) \* sizeof(int\*));

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

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

matrix[s][i] = 0;

for (size\_t i = 0; i < s + 1; i++)

{

matrix[i] = (int\*)realloc(matrix[i], (s + 1) \* sizeof(int));

matrix[i][s] = 0;

}

matrix[s][x] = 1;

matrix[x][s] = 1;

matrix[s][y] = 1;

matrix[y][s] = 1;

\*size = \*size + 1;

return matrix;

}

DataType\*\* IdentificationL(DataType\*\* matrix, int\* size, int x, int y)

{

int s = \*size;

if (y < x)

{

int t = x;

x = y;

y = t;

}

DataType\*\* matrixL = (DataType\*\*)malloc(sizeof(DataType\*) \* s - 1);

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

matrixL[i] = CreateDataType(i);

int d = 0;

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

{

if (i == x || i == y)

continue;

for (size\_t j = 1; j < s; j++)

{

int\* val = TryCheckElementAt(j, matrix[i]);

if (val == NULL)

break;

if (\*val == x || \*val == y)

continue;

if (\*val >= x)

\*val -= 1;

if (\*val >= y)

\*val -= 1;

if (val < 0)

continue;

Add(\*val, matrixL[d]);

}

d++;

}

for (size\_t i = 1; i < s; i++)

{

int\* val = TryCheckElementAt(i, matrix[x]);

if (val == NULL)

break;

if (\*val >= x)

\*val -= 1;

if (\*val >= y)

\*val -= 1;

if (\*val < 0)

continue;

int bool = 0;

for (size\_t j = 1; j < s - 1; j++)

{

int\* cVal = TryCheckElementAt(j, matrixL[s - 2]);

if (cVal == NULL)

{

break;

}

if (\*cVal == \*val)

{

bool = 1;

break;

}

}

if (bool == 1)

continue;

Add(\*val, matrixL[s - 2]);

}

for (size\_t i = 1; i < s; i++)

{

int\* val = TryCheckElementAt(i, matrix[y]);

if (val == NULL)

break;

if (\*val >= x)

\*val -= 1;

if (\*val >= y)

\*val -= 1;

if (\*val < 0)

continue;

int bool = 0;

for (size\_t j = 1; j < s - 1; j++)

{

int\* cVal = TryCheckElementAt(j, matrixL[s - 2]);

if (cVal == NULL)

{

break;

}

if (\*cVal == \*val)

{

bool = 1;

break;

}

}

if (bool == 1)

continue;

Add(\*val, matrixL[s - 2]);

}

for (int i = 1; i < s - 1; i++)

{

int\* vert = TryCheckElementAt(i, matrixL[s - 2]);

if (vert == NULL)

return;

Add(s - 2, matrixL[\*vert]);

}

\*size = \*size - 1;

return matrixL;

}

DataType\*\* TighteningL(DataType\*\* matrix, int\* size, int x, int y)

{

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

{

int\* val = TryCheckElementAt(i, matrix[x]);

if (val == NULL)

break;

if (\*val == y)

return IdentificationL(matrix, size, x, y);

}

}

DataType\*\* SplittingL(DataType\*\* matrix, int\* size, int x, int y)

{

int bool = 1;

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

{

int\* val = TryCheckElementAt(i, matrix[x]);

if (val == NULL)

break;

if (\*val == y)

{

bool = 0;

break;

}

}

if (bool == 1)

return matrix;

matrix = (DataType\*\*)realloc(matrix, (\*size + 1)\* sizeof(DataType\*));

matrix[\*size] = CreateDataType(\*size);

Add(x, matrix[\*size]);

Add(y, matrix[\*size]);

Add(\*size, matrix[x]);

Add(\*size, matrix[y]);

\*size = \*size + 1;

return matrix;

}

#pragma endregion

#pragma region Third Number

int\*\* Union(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* outSize)

{

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\*\* uMatrix = GenerateMatrix(bSize, bSize);

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

{

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

{

int res = 0;

if (i >= lSize || j >= lSize)

res = bMatrix[i][j];

else

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

uMatrix[i][j] = res;

}

}

\*outSize = bSize;

return uMatrix;

}

int\*\* Intersection(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* outSize)

{

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\*\* iMatrix = GenerateMatrix(bSize, bSize);

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

{

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

{

int res = lMatrix[i][j] \* bMatrix[i][j];

iMatrix[i][j] = res;

}

}

\*outSize = bSize;

return iMatrix;

}

int\*\* RoundSum(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* outSize)

{

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\*\* dMatrix = GenerateMatrix(lSize, lSize);

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

{

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

{

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

dMatrix[i][j] = res;

}

}

int rsSize = lSize;

int\*\* rsMatrix = dMatrix;

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

{

int c = 0;

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

{

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

c++;

}

if (c == lSize)

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

}

\*outSize = rsSize;

return rsMatrix;

}

#pragma endregion

#pragma region Fourth Number

int\*\* Multiplication(int\*\* matrix1, int size1, int\*\* matrix2, int size2, int\* sizeOut)

{

\*sizeOut = size1 \* size2;

int s = \*sizeOut;

int\*\* mMatrix = GenerateMatrix(s, s);

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 n = 0, m = 0;

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

{

for (size\_t k = 0; k < bSize; k++)

{

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

{

for (size\_t l = 0; l < bSize; l++)

{

if (i == j)

{

if (bMatrix[k][l] == 1)

mMatrix[n][m] = 1;

}

else if (k == l)

{

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

mMatrix[n][m] = 1;

}

m++;

}

}

n++;

m = 0;

}

}

return mMatrix;

}

#pragma endregion

* **list.h**

#ifndef LIST

#define LIST

typedef struct Node {

int value;

struct Node\* next;

} DataType;

int IsDataTypeNull(DataType\* head);

DataType\* CreateDataType(int data);

void Add(int data, DataType\* head);

DataType\* TryAddElementAt(int data, int pos, DataType\* head);

DataType\* TryRemoveElementAt(int pos, DataType\* head);

int\* TryCheckElementAt(int pos, DataType\* head);

void PrintDataType(DataType\* head);

int\* GetLength(DataType\* head);

DataType\* Clear(DataType\* head);

#endif

* **list.c**

#include <stdio.h>

#include <stdlib.h>

#include "list.h"

DataType\* CreateElement(int data)

{

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

if (tmp == NULL)

{

printf("ERROR: No memory");

return NULL;

}

tmp->value = data;

tmp->next = NULL;

return tmp;

}

int IsDataTypeNull(DataType\* head)

{

if (head == NULL)

return 1;

else

return 0;

}

DataType\* CreateDataType(int data)

{

DataType\* tmp = CreateElement(data);

return tmp;

}

void Add(int data, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return;

}

DataType\* tmp = CreateElement(data);

if (tmp == NULL)

return;

DataType\* l = head;

while (l->next != NULL)

{

l = l->next;

}

l->next = tmp;

}

DataType\* TryAddElementAt(int data, int pos, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return head;

}

DataType\* tmp = CreateElement(data);

if (tmp == NULL)

return head;

if (pos == 0)

{

DataType\* headEl = head;

tmp->next = headEl;

head = tmp;

return head;

}

DataType\* l = head;

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

{

if (l == NULL)

{

printf("ERROR: Index out of list\n");

return head;

}

l = l->next;

}

tmp->next = l->next;

l->next = tmp;

return head;

}

DataType\* TryRemoveElementAt(int pos, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return head;

}

if (pos == 0)

{

DataType\* headEl = head;

head = headEl->next;

free(headEl);

return head;

}

DataType\* l = head;

DataType\* tmp = NULL;

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

{

if (l == NULL || l->next == NULL)

{

printf("ERROR: Index out of list\n");

return head;

}

tmp = l;

l = l->next;

}

tmp->next = l->next;

free(l);

return head;

}

int\* TryCheckElementAt(int pos, DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

DataType\* l = head;

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

{

if (l->next == NULL || l == NULL)

return NULL;

l = l->next;

}

return &(l->value);

}

void PrintDataType(DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return;

}

DataType\* l = head;

while (l->next != NULL)

{

printf("%d ", l->value);

l = l->next;

}

printf("%d\n", l->value);

}

int\* GetLength(DataType\* head)

{

if (IsDataTypeNull(head) == 1)

{

printf("ERROR: Null exception\n");

return NULL;

}

int count = 0;

DataType\* l = head;

while (l != NULL)

{

l = l->next;

count++;

}

return &count;

}

DataType\* Clear(DataType\* dataType)

{

int\* length = GetLength(dataType);

if (length == NULL)

return;

DataType\* h = dataType;

for (int i = 0; i < \*length; i++)

{

h = TryRemoveElementAt(0, h);

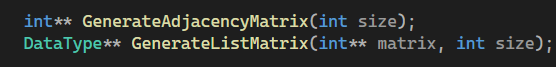
}

return h;

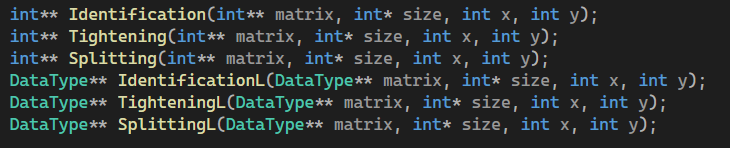
}

**Задания**

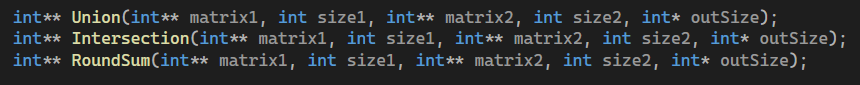
Задание 1



Задание 2

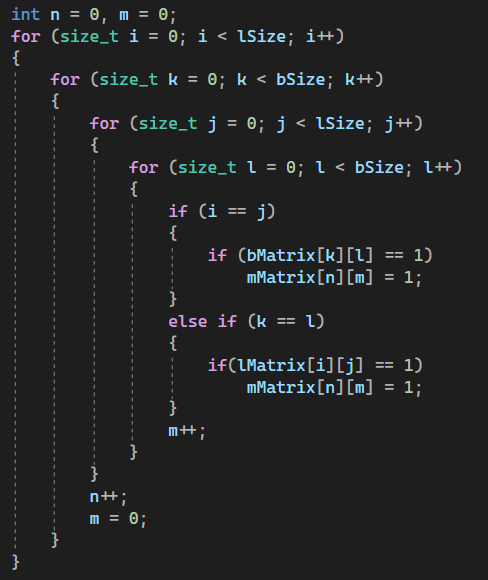


Задание 3

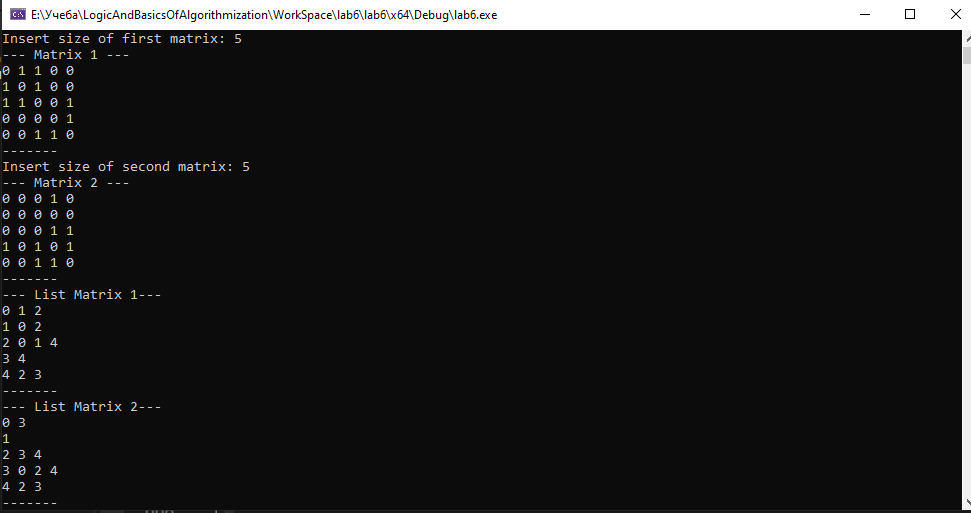


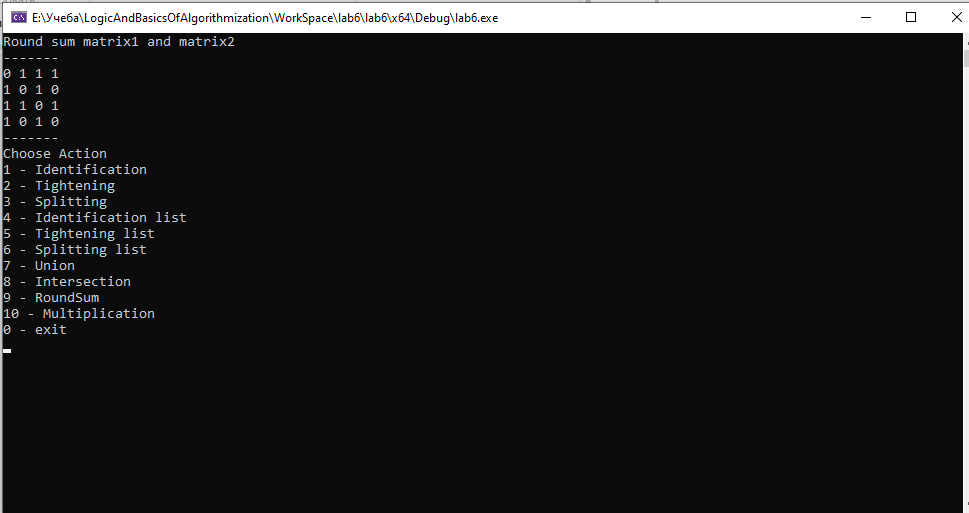
Задание 4





**Результат работы программы**

****

****

**Вывод**

Научился выполнять унарные и бинарные операции над графом