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

**ОТЧËТ**  
по лабораторной работе №6  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Унарные и бинарные операции над графами»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

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  *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 \***

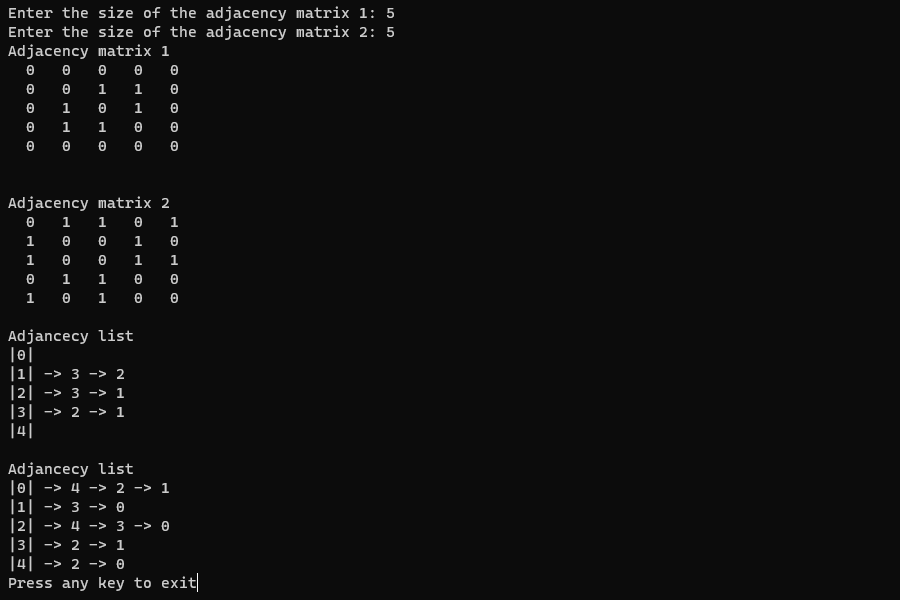
1. Для матричной формы представления графов выполните операцию декартова произведения графов *G = G*1X *G*2.

Результат выполнения операции выведите на экран.

**Ход работы**

**Задание 1**

1. Сгенерировал две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Вывел сгенерированные матрицы на экран.
2. \* Для указанных графов преобразовал представление матриц смежности в списки смежности. Вывел полученные списки на экран.

****

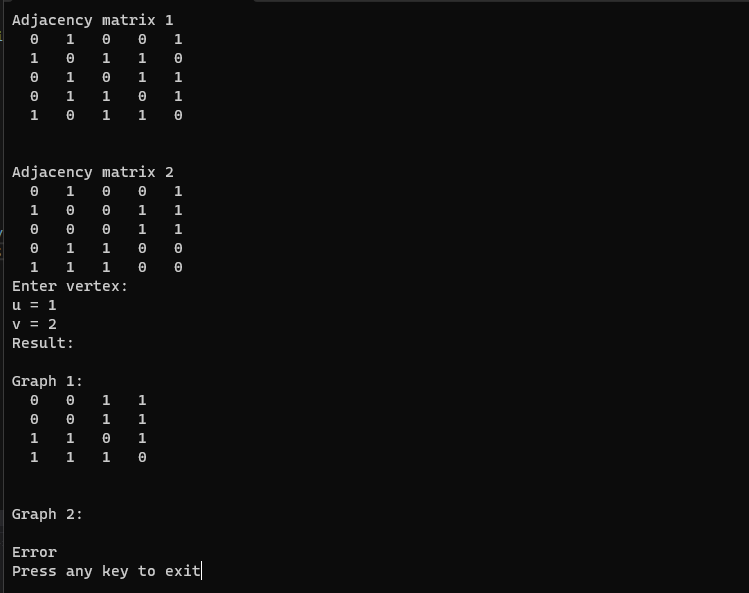
**Задание 2**

1. Для матричной формы представления графов выполнил операции:

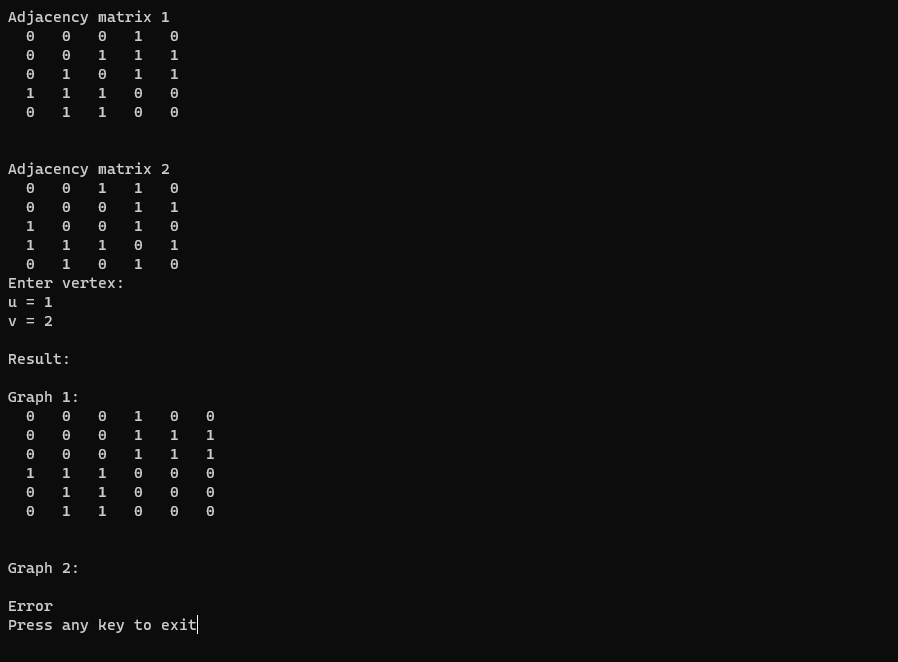
а) отождествления вершин



б) стягивания ребра

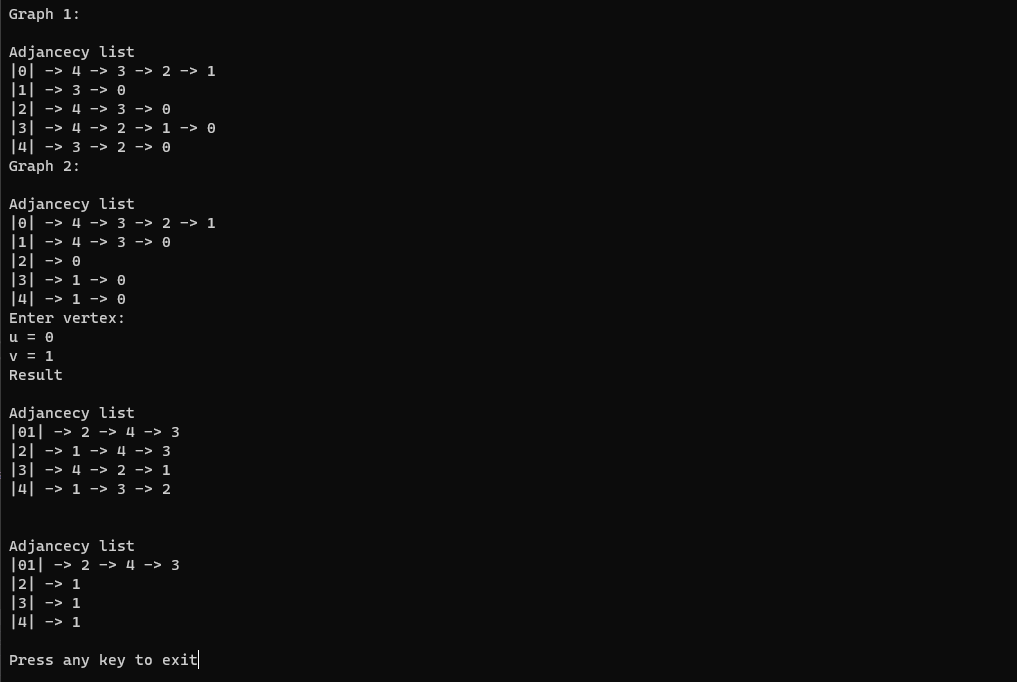


в) расщепления вершины



2\* Для представления графов в виде списков смежности выполнил операции:

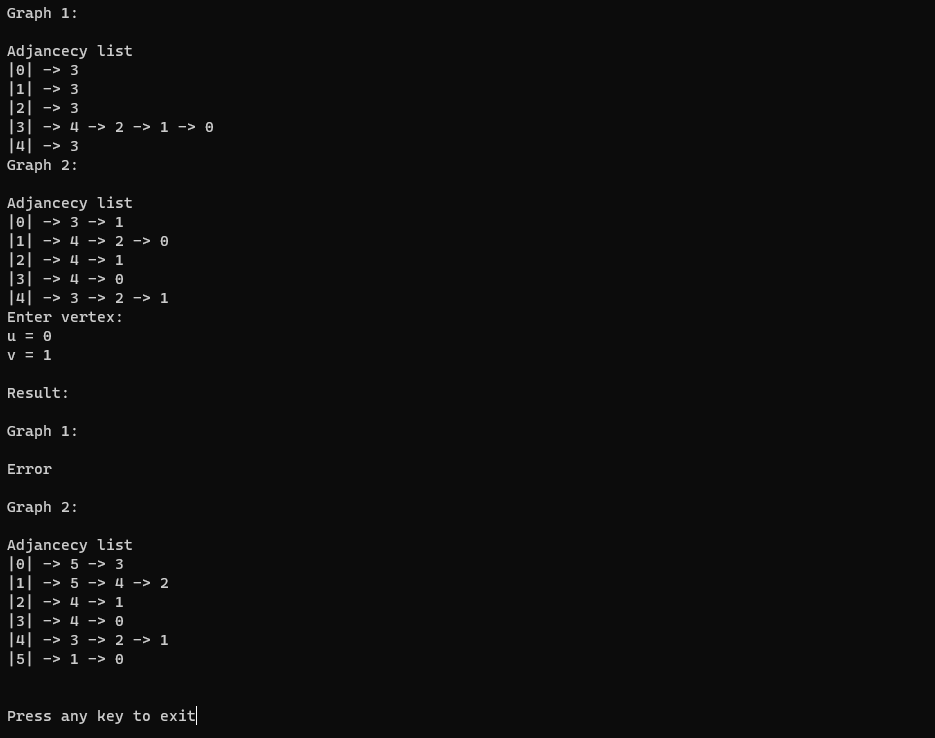
а) отождествления вершин



б) стягивания ребра



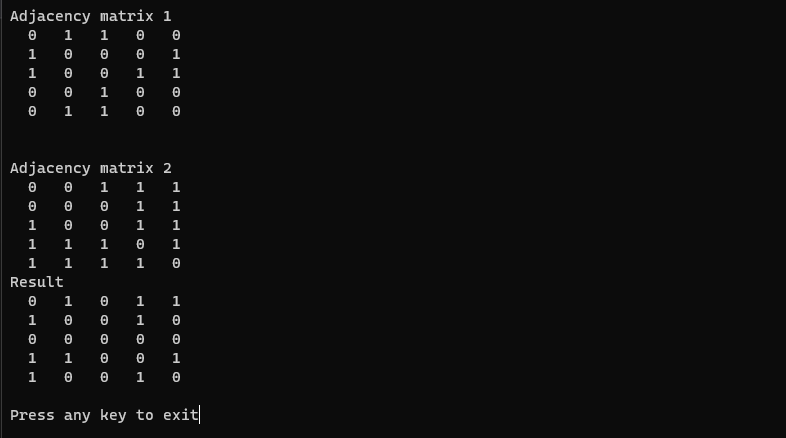
в) расщепления вершины



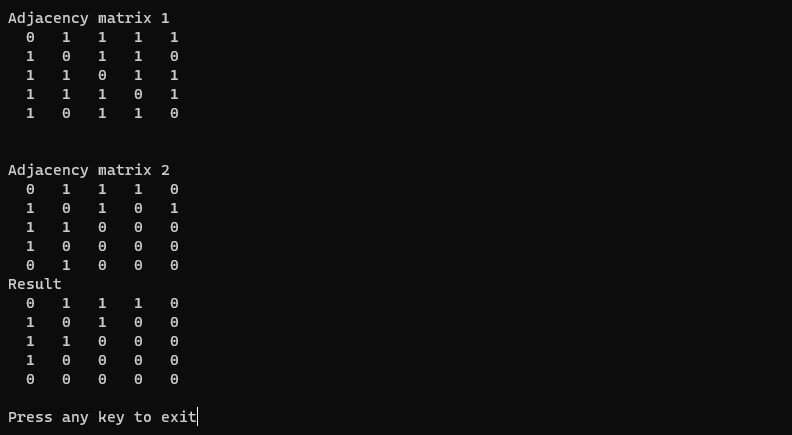
**Задание 3**

1. Для матричной формы представления графов выполнил операции:

а) объединения *G* = *G*1  *G*2



б) пересечения *G* = *G*1  *G*2



в) кольцевой суммы *G* = *G*1  *G*2



**Задание 4 \***

1. Для матричной формы представления графов выполнил операцию декартова произведения графов *G = G*1X *G*2.



**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void fill\_arr(int\*\* Arr, int size);

void print\_arr(int\*\* Arr, int size);

int\*\* vertexIdentification2(int\*\* Arr, int size, int u, int v);

int\*\* vertexSplitting(int\*\* Arr, int size, int u, int v);

int find\_arr(int\* Arr, int size, int element);

int\*\* vertexIdentification1(int\*\* matrix, int size, int u, int v);

int\*\* workGraph(int\*\* Arr, int sizeMatrix1, int\*\* Arrey, int sizeMatrix2, int work);

int\*\* Multiplication(int\*\* Arr, int sizeMatrix1, int\*\* Arrey, int sizeMatrix2);

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

struct Graph\* vertexSplittingMatrix(struct Graph\* graph, int\*\* Arrey, int size, int u, int v);

struct node\* createNode(int v);

struct Graph\* createGraph(int vertices);

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

void printGraph(struct Graph\* graph);

struct Graph\* transformGraph(int\*\* Arr, int size);

struct Graph\* delVertexGraph(int\*\* Arr, int size, int u);

struct Graph\* vertexIndentificationMatrix(struct Graph\* graph, int\*\* Arrey, int size, int u, int v);

void printGraphDel(struct Graph\* graph, int u, int k);

int main(void)

{

srand(time(NULL));

int sizeMatrix1, sizeMatrix2;

int\*\* Arr, \*\*Arrey, \*\*ArrResult;

char ch;

int ext = 0, size = 0;;

int u, v;

printf("Enter the size of the adjacency matrix 1: ");

scanf("%d", &sizeMatrix1);

Arr = (int\*\*)malloc(sizeMatrix1 \* sizeof(int\*));

for (int i = 0; i < sizeMatrix1; ++i)

{

Arr[i] = (int\*)malloc(sizeMatrix1 \* sizeof(int));

}

printf("Enter the size of the adjacency matrix 2: ");

scanf("%d", &sizeMatrix2);

Arrey = (int\*\*)malloc(sizeMatrix2 \* sizeof(int\*));

for (int i = 0; i < sizeMatrix2; ++i)

{

Arrey[i] = (int\*)malloc(sizeMatrix2 \* sizeof(int));

}

printf("Adjacency matrix 1\n");

fill\_arr(Arr, sizeMatrix1);

print\_arr(Arr, sizeMatrix1);

printf("\n\nAdjacency matrix 2\n");

fill\_arr(Arrey, sizeMatrix2);

print\_arr(Arrey, sizeMatrix2);

if (sizeMatrix1 >= sizeMatrix2)

size = sizeMatrix1;

else

size = sizeMatrix2;

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

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

{

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

}

struct Graph\* graph1 = transformGraph(Arr, sizeMatrix1);

struct Graph\* graph2 = transformGraph(Arrey, sizeMatrix2);

printGraph(graph1);

printGraph(graph2);

printf("Press any key to exit");

getchar();

getchar();

do

{

system("cls");

printf("0. Vertex identification (Matrix)\n1. Vertex contraction (Matrix)\n2. Vertex splitting (Matrix)\n3. Vertex identification (list)\n\

4. Vertex contraction (list)\n5. Vertex splitting (list)\n6. Unification graph\n7. Intersection graph\n8. Ring sum graph\n9. Multiplication graph\n");

printf("Select item: ");

scanf("%c", &ch);

system("cls");

if (ch != '\n')

{

scanf("%\*[^\n]");

getchar();

}

switch (ch)

{

case '0':

printf("Adjacency matrix 1\n");

print\_arr(Arr, sizeMatrix1);

printf("\n\nAdjacency matrix 2\n");

print\_arr(Arrey, sizeMatrix2);

printf("Enter vertex: \n");

printf("u = ");

scanf("%d", &u);

printf("v = ");

scanf("%d", &v);

Arr = vertexIdentification1(Arr, sizeMatrix1, u, v);

Arrey = vertexIdentification1(Arrey, sizeMatrix2, u, v);

printf("Result:\n");

print\_arr(Arr, sizeMatrix1 - 1);

printf("\n");

print\_arr(Arrey, sizeMatrix1 - 1);

printf("Press any key to exit");

getchar();

getchar();

break;

case '1':

printf("Adjacency matrix 1\n");

print\_arr(Arr, sizeMatrix1);

printf("\n\nAdjacency matrix 2\n");

print\_arr(Arrey, sizeMatrix2);

printf("Enter vertex: \n");

printf("u = ");

scanf("%d", &u);

printf("v = ");

scanf("%d", &v);

printf("Result:\n");

printf("\nGraph 1:\n");

if (Arr[u][v] == 0)

{

printf("\nError\n");

}

else

{

Arr = vertexIdentification1(Arr, sizeMatrix1, u, v);

print\_arr(Arr, sizeMatrix1 - 1);

printf("\n");

}

printf("\nGraph 2:\n");

if (Arrey[u][v] == 0)

{

printf("\nError\n");

}

else

{

Arrey = vertexIdentification1(Arrey, sizeMatrix2, u, v);

print\_arr(Arrey, sizeMatrix2 - 1);

printf("\n");

}

printf("Press any key to exit");

getchar();

getchar();

break;

case '2':

printf("Adjacency matrix 1\n");

print\_arr(Arr, sizeMatrix1);

printf("\n\nAdjacency matrix 2\n");

print\_arr(Arrey, sizeMatrix2);

printf("Enter vertex: \n");

printf("u = ");

scanf("%d", &u);

printf("v = ");

scanf("%d", &v);

printf("\nResult:\n");

printf("\nGraph 1:\n");

if (Arr[u][v] == 0)

{

printf("\nError\n");

}

else

{

Arr = vertexSplitting(Arr, sizeMatrix1, u, v);

print\_arr(Arr, sizeMatrix1 + 1);

printf("\n");

}

printf("\nGraph 2:\n");

if (Arrey[u][v] == 0)

{

printf("\nError\n");

}

else

{

Arrey = vertexSplitting(Arrey, sizeMatrix2, u, v);

print\_arr(Arrey, sizeMatrix2 + 1);

printf("\n");

}

printf("Press any key to exit");

getchar();

getchar();

break;

case '3':

printf("Graph 1:\n");

printGraph(graph1);

printf("Graph 2:\n");

printGraph(graph2);

printf("Enter vertex: \n");

printf("u = ");

scanf("%d", &u);

printf("v = ");

scanf("%d", &v);

printf("Result\n");

graph1 = vertexIndentificationMatrix(graph1, Arr, sizeMatrix1, u, v);

graph2 = vertexIndentificationMatrix(graph2, Arrey, sizeMatrix2, u, v);

printGraphDel(graph1, u , v);

printf("\n");

printGraphDel(graph2, u, v);

printf("\nPress any key to exit");

getchar();

getchar();

break;

case '4':

printf("Graph 1:\n");

printGraph(graph1);

printf("Graph 2:\n");

printGraph(graph2);

printf("Enter vertex: \n");

printf("u = ");

scanf("%d", &u);

printf("v = ");

scanf("%d", &v);

printf("\nResult:\n");

printf("\nGraph 1:\n");

if (Arr[u][v] == 0)

{

printf("\nError\n");

}

else

{

graph1 = vertexIndentificationMatrix(graph1, Arr, sizeMatrix1, u, v);

printGraphDel(graph1, u, v);

printf("\n");

}

printf("\nGraph 2:\n");

if (Arrey[u][v] == 0)

{

printf("\nError\n");

}

else

{

graph2 = vertexIndentificationMatrix(graph2, Arrey, sizeMatrix2, u, v);

printGraphDel(graph2, u, v);

printf("\n");

}

printf("\nPress any key to exit");

getchar();

getchar();

break;

case '5':

printf("Graph 1:\n");

printGraph(graph1);

printf("Graph 2:\n");

printGraph(graph2);

printf("Enter vertex: \n");

printf("u = ");

scanf("%d", &u);

printf("v = ");

scanf("%d", &v);

printf("\nResult:\n");

printf("\nGraph 1:\n");

if (Arr[u][v] == 0)

{

printf("\nError\n");

}

else

{

graph1 = vertexSplittingMatrix(graph1, Arr, sizeMatrix1, u, v);

printGraph(graph1);

printf("\n");

}

printf("\nGraph 2:\n");

if (Arrey[u][v] == 0)

{

printf("\nError\n");

}

else

{

graph2 = vertexSplittingMatrix(graph2, Arrey, sizeMatrix2, u, v);

printGraph(graph2);

printf("\n");

}

printf("\nPress any key to exit");

getchar();

getchar();

break;

case '6':

printf("Adjacency matrix 1\n");

print\_arr(Arr, sizeMatrix1);

printf("\n\nAdjacency matrix 2\n");

print\_arr(Arrey, sizeMatrix2);

printf("Result\n");

ArrResult = workGraph(Arr, sizeMatrix1, Arrey, sizeMatrix2, 0);

print\_arr(ArrResult, size);

printf("\nPress any key to exit");

getchar();

break;

case '7':

printf("Adjacency matrix 1\n");

print\_arr(Arr, sizeMatrix1);

printf("\n\nAdjacency matrix 2\n");

print\_arr(Arrey, sizeMatrix2);

printf("Result\n");

ArrResult = workGraph(Arr, sizeMatrix1, Arrey, sizeMatrix2, 1);

print\_arr(ArrResult, size);

printf("\nPress any key to exit");

getchar();

break;

case '8':

printf("Adjacency matrix 1\n");

print\_arr(Arr, sizeMatrix1);

printf("\n\nAdjacency matrix 2\n");

print\_arr(Arrey, sizeMatrix2);

printf("Result\n");

ArrResult = workGraph(Arr, sizeMatrix1, Arrey, sizeMatrix2, 2);

print\_arr(ArrResult, size);

printf("\nPress any key to exit");

getchar();

break;

case '9':

printf("Matrix 1:\n");

print\_arr(Arr, sizeMatrix1);

printf("Matrix 2:\n");

print\_arr(Arrey, sizeMatrix2);

printf("Result\n");

ArrResult = Multiplication(Arr, sizeMatrix1, Arrey, sizeMatrix2);

size = sizeMatrix1 \* sizeMatrix2;

print\_arr(ArrResult, size);

printf("\nPress any key to exit");

getchar();

break;

default:

break;

}

} while (ext == 0);

free(Arr);

free(Arrey);

return(0);

}

struct node

{

int vertex;

struct node\* next;

};

struct Graph

{

int numVertices;

struct node\*\* adjLists;

};

void print\_arr(int\*\* Arr, int size)

{

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

{

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

{

printf("%3d ", Arr[i][j]);

}

printf("\n");

}

}

int\*\* vertexIdentification2(int\*\* Arr, int size, int u, int v)

{

int\*\* ArreyIndentification;

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

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

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

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

{

ArrU[i] = Arr[u][i];

ArrV[i] = Arr[v][i];

}

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

{

ArrZ[i] = ArrU[i] || ArrV[i];

}

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

{

Arr[u][i] = 0;

Arr[i][u] = 0;

Arr[v][i] = 0;

Arr[i][v] = 0;

}

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

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

{

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

}

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

{

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

{

ArreyIndentification[i][j] = Arr[i][j];

}

}

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

{

if (u == i)

{

ArreyIndentification[u][i] = 0;

continue;

}

if (v == i)

{

ArreyIndentification[v][i] = 0;

ArreyIndentification[i][v] = 0;

continue;

}

ArreyIndentification[u][i] = ArrZ[i];

ArreyIndentification[i][u] = ArrZ[i];

}

return ArreyIndentification;

}

int\*\* vertexIdentification1(int\*\* Arr, int size, int u, int v)

{

int\*\* ArreyIndentification;

ArreyIndentification = (int\*\*)malloc((size - 1) \* sizeof(int\*));

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

{

ArreyIndentification[i] = (int\*)malloc((size - 1) \* sizeof(int));

}

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

{

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

{

ArreyIndentification[i][j] = 0;

}

}

if (v < u)

{

int t = u;

u = v;

v = t;

}

int di = 0, dj = 0;

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

{

if (i == u || i == v)

i++;

if (i == u || i == v)

i++;

if (i >= size)

break;

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

{

if (j == u || j == v)

j++;

if (j == u || j == v)

j++;

if (j >= size)

break;

ArreyIndentification[di][dj] = Arr[i][j];

dj++;

}

di++;

dj = 0;

}

int k = 0;

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

{

if (i == u || i == v)

i++;

if (i == u || i == v)

i++;

if (i >= size)

break;

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

{

ArreyIndentification[size - 2][k] = 1;

ArreyIndentification[k][size - 2] = 1;

}

if (Arr[u][i] == 1)

{

ArreyIndentification[size - 2][k] = 1;

ArreyIndentification[k][size - 2] = 1;

}

k++;

}

return ArreyIndentification;

}

int\*\* vertexSplitting(int\*\* Arr, int size, int u, int v)

{

int\*\* ArreyIndentification;

ArreyIndentification = (int\*\*)malloc((size + 1) \* sizeof(int\*));

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

{

ArreyIndentification[i] = (int\*)malloc((size + 1) \* sizeof(int));

}

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

{

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

{

if (i == size || j == size) {

ArreyIndentification[i][j] = 0;

continue;

}

ArreyIndentification[i][j] = Arr[i][j];

}

}

ArreyIndentification[u][v] = 0;

ArreyIndentification[v][u] = 0;

if (Arr[u][v] == 1)

{

ArreyIndentification[u][size] = 1;

ArreyIndentification[size][u] = 1;

ArreyIndentification[v][size] = 1;

ArreyIndentification[size][v] = 1;

}

return ArreyIndentification;

}

int\*\* workGraph(int\*\* Arr, int sizeMatrix1, int\*\* Arrey, int sizeMatrix2, int work)

{

int size = 0;

if (sizeMatrix1 >= sizeMatrix2)

size = sizeMatrix1;

else

size = sizeMatrix2;

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

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

{

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

}

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

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

{

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

}

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

{

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

{

ArrU[i][j] = 0;

}

}

if (sizeMatrix1 >= sizeMatrix2)

{

for (int i = 0; i < sizeMatrix2; i++)

{

for (int j = 0; j < sizeMatrix2; j++)

{

ArrU[i][j] = Arrey[i][j];

}

}

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

{

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

{

if(work == 0)

ArrResult[i][j] = ArrU[i][j] || Arr[i][j];

if(work == 1)

ArrResult[i][j] = ArrU[i][j] && Arr[i][j];

else

ArrResult[i][j] = ArrU[i][j] ^ Arr[i][j];

}

}

}

else

{

for (int i = 0; i < sizeMatrix1; i++)

{

for (int j = 0; j < sizeMatrix1; j++)

{

ArrU[i][j] = Arr[i][j];

}

}

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

{

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

{

if (work == 0)

ArrResult[i][j] = ArrU[i][j] || Arrey[i][j];

if (work == 1)

ArrResult[i][j] = ArrU[i][j] && Arrey[i][j];

else

ArrResult[i][j] = ArrU[i][j] ^ Arrey[i][j];

}

}

}

int s = size;

if (work == 2)

{

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

{

int count = 0;

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

{

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

count++;

}

if (count == size)

ArrResult = RemoveVertex(ArrResult, &s, i);

}

}

return ArrResult;

}

struct Graph\* vertexIndentificationMatrix(struct Graph\* graph, int\*\* Arrey, int size, int u, int v)

{

int count = 0, count2 = 0;

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

int\* Arr = (int\*)malloc(count \* sizeof(int));

int\* Arr2 = (int\*)malloc(count2 \* sizeof(int));

while (temp)

{

count++;

Arr = (int\*)realloc(Arr, count \* sizeof(int));

Arr[count - 1] = temp->vertex;

temp = temp->next;

}

temp = graph->adjLists[u];

while (temp)

{

if (temp->vertex == v)

{

temp = temp->next;

continue;

}

if (find\_arr(Arr, count, temp -> vertex) == 1)

{

count2++;

Arr2 = (int\*)realloc(Arr2, count2 \* sizeof(int));

Arr2[count2 - 1] = temp->vertex;

}

temp = temp->next;

}

graph = delVertexGraph(Arrey, size, u);

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

{

addEdge(graph, v, Arr2[i]);

}

struct Graph\* graph2 = createGraph(size - 1);

int k = 0;

for (int i = 0; i < graph2->numVertices + 1; i++)

{

if (i == u)

{

continue;

}

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

graph2->adjLists[k] = temp;

k++;

}

return graph2;

}

struct Graph\* delVertexGraph(int\*\* Arr, int size, int u)

{

struct Graph\* graph = createGraph(size);

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

{

if (i == u)

continue;

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

{

if (j == u)

continue;

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

{

addEdge(graph, i, j);

}

}

}

return graph;

}

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

{

int s = \*size;

int\*\* ArrResult = (int\*\*)malloc((s - 1) \* sizeof(int\*));

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

{

ArrResult[i] = (int\*)malloc((s - 1) \* sizeof(int));

}

int di = 0, dj = 0;

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

{

if (i == r)

i++;

if (i >= s)

break;

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

{

if (j == r)

j++;

if (j >= s)

break;

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

dj++;

}

di++;

dj = 0;

}

\*size = s - 1;

return ArrResult;

}

int find\_arr(int\* Arr, int size, int element)

{

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

{

if (element == Arr[j])

{

return 0;

}

}

return 1;

}

void fill\_arr(int\*\* Arr, int size)

{

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

{

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

{

if (i == j)

{

Arr[i][j] = 0;

continue;

}

if (rand() % 3 == 1)

{

Arr[i][j] = 0;

Arr[j][i] = 0;

}

else

{

Arr[i][j] = 1;

Arr[j][i] = 1;

}

}

}

}

struct Graph\* vertexSplittingMatrix(struct Graph\* graph, int\*\* Arrey, int size, int u, int v)

{

struct Graph\* graph2 = createGraph(size + 1);

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

{

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

{

if (i == u && j == v)

continue;

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

{

addEdge(graph2, i, j);

}

}

}

addEdge(graph2, size, u);

addEdge(graph2, size, v);

return graph2;

}

struct node\* createNode(int v)

{

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

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

graph->numVertices = vertices;

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

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

graph->adjLists[i] = NULL;

return graph;

}

struct Graph\* transformGraph(int\*\* Arr, int size)

{

struct Graph\* graph = createGraph(size);

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

{

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

{

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

{

addEdge(graph, i, j);

}

}

}

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)

{

printf("\nAdjancecy list");

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

{

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

printf("\n|%d|", v);

while (temp)

{

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

temp = temp->next;

}

}

printf("\n");

}

void printGraphDel(struct Graph\* graph, int u, int k)

{

printf("\nAdjancecy list");

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

{

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

if (v == k - 1)

{

printf("\n|%d%d|", u, k);

}

else if (v >= u)

{

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

}

else

{

printf("\n|%d|", v);

}

while (temp)

{

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

temp = temp->next;

}

}

printf("\n");

}

int\*\* Multiplication(int\*\* Arr, int sizeMatrix1, int\*\* Arrey, int sizeMatrix2)

{

int size = sizeMatrix1 \* sizeMatrix2;

int lSize, bSize;

int\*\* lMatrix, \*\* bMatrix;

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

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

{

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

}

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

{

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

{

ArrResult[i][j] = 0;

}

}

if (sizeMatrix1 >= sizeMatrix2)

{

bSize = sizeMatrix1;

lSize = sizeMatrix2;

bMatrix = Arr;

lMatrix = Arrey;

}

else

{

lSize = sizeMatrix1;

bSize = sizeMatrix2;

lMatrix = Arr;

bMatrix = Arrey;

}

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)

ArrResult[n][m] = 1;

}

else if (k == l)

{

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

ArrResult[n][m] = 1;

}

m++;

}

}

n++;

m = 0;

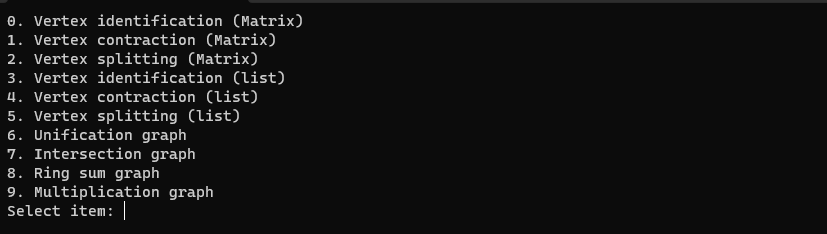
}

}

return ArrResult;

}

**Результат работы программы**

****

**Вывод**

В результате выполнения лабораторной работы я научился выполнять унарные и бинарные операции над графами.