Міністерство освіти і науки, молоді та спорту України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 2**

**«Основні поняття та застосування теорії графів»**

**(до тем «Теорія неорієнтованих графів», «Теорія орієнтованих графів»**

**Лектор:**

проф. кафедри ПЗ

Журавчак Л.М.

**Виконав:**

студ. групи ПЗ-14

Губик А. С.

**Прийняв:**

асистент кафедри ПЗ

Курапов П. P.

«\_\_\_» \_\_\_\_\_\_ 2022 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2022

**ТЕМА РОБОТИ**: Основні поняття та застосування теорії графів

**МЕТА РОБОТИ**: Ознайомитись на практиці із основними поняттями теорії неорієнтованих таорієнтованих графів, навчитись виконувати операції над графами, будувати матрицідосяжності, знаходити у зваженому графі субоптимальний гамільтоновий цикл,засвоїти алгоритми пошуку вглиб і вшир та алгоритм Дейкстри.

#### Додаток

**graph.c:**

#include <stdio.h>

#include "graph.h"

void AddMatrix(int res[], int a[], int b[], int size)

{

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

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

res[i \* size + j] = a[i \* size + j] || b[i \* size + j];

}

void MultMatrix(int res[], int a[], int b[], int size)

{

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

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

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

res[i \* size + j] = res[i \* size + j] ||

a[i \* size + k] &&

b[k \* size + j];

}

void PrintMatrix(int g[], int size)

{

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

printf("\n");

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

printf("%d ", g[i \* size + j]);

}

printf("\n");

}

void CopyMatrix(int res[], int g[], int size)

{

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

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

res[i \* size + j] = g[i \* size + j];

}

void ConstructComplementary(int GC[], int G[], int size)

{

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

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

if (i == j)

continue;

GC[i \* size + j] = !G[i \* size + j];

}

}

void ConstructUnion(int U[],int G[],int H[], int size)

{

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

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

if (i == j)

continue;

U[i \* size + j] = G[i \* size + j] || H[i \* size + j];

}

}

void ConstructCrossing(int U[], int G[], int H[], int size)

{

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

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

if (i == j)

continue;

U[i \* size + j] = G[i \* size + j] && H[i \* size + j];

}

}

int isUndirected(int G[], int size)

{

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

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

if(G[i \* size + j] != G[j \* size + i])

return 0;

return 1;

}

int CountEdges(int G[], int size)

{

int res = 0;

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

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

if(G[i \* size + j])

res++;

if(isUndirected(G, size))

return res / 2;

return res;

}

**graph.h:**

void AddMatrix(int res[], int a[], int b[], int size);

void MultMatrix(int res[], int a[], int b[], int size);

void PrintMatrix(int g[], int size);

void CopyMatrix(int res[], int g[], int n);

void ConstructComplementary(int GC[], int G[], int size);

void ConstructUnion(int U[], int G[], int H[], int size);

void ConstructCrossing(int X[], int G[], int H[], int size);

int CountEdges(int G[], int size);

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 1

Граф G=(V,E) задано матрицею суміжності (таблиця 2.1).

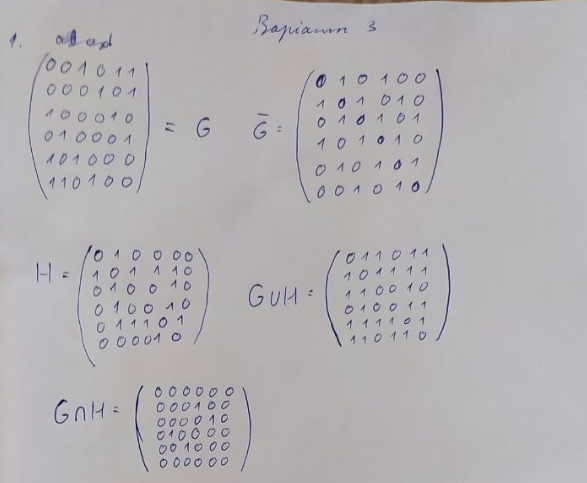
Вказати кількість вершин і ребер у графі-доповненні до графа G, у графі-перетині G∩H та

у графі-об’єднанні GUH. Граф H – варіант k+1.

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує

обчислення матриць суміжності графа-доповнення, графа-перетину та графа-об’єднання.

ОПИС ВИКОНАННЯ РОБОТИ



ТЕКСТ ПРОГРАМИ

#include <stdio.h>

#include "graph.h"

enum{

GraphSize = 6

};

int main(){

int G[GraphSize][GraphSize] = {{0, 0, 1, 0, 1, 1},

{0, 0, 0, 1, 0, 1},

{1, 0, 0, 0, 1, 0},

{0, 1, 0, 0, 0, 1},

{1, 0, 1, 0, 0, 0},

{1, 1, 0, 1, 0, 0}

};

int H[GraphSize][GraphSize] = {{0, 1, 0, 0, 0, 0},

{1, 0, 1, 1, 1, 0},

{0, 1, 0, 0, 1, 0},

{0, 1, 0, 0, 1, 0},

{0, 1, 1, 1, 0, 1},

{0, 0, 0, 0, 1, 0}

};

int GC [GraphSize][GraphSize] = {0};

int U [GraphSize][GraphSize] = {0};

int X [GraphSize][GraphSize] = {0};

ConstructComplementary(GC, G, GraphSize);

ConstructUnion(U, G, H, GraphSize);

ConstructCrossing(X, G, H, GraphSize);

printf("\_\n");

printf("G:\n");

printf("Verticies: %d, edges: %d",

GraphSize, CountEdges(GC, GraphSize));

PrintGraph(GC, GraphSize);

printf("\nG ∪ H:\n");

printf("Verticies: %d, edges: %d",

GraphSize, CountEdges(U, GraphSize));

PrintGraph(U, GraphSize);

printf("\nG ⋂ H:\n");

printf("Verticies: %d, edges: %d",

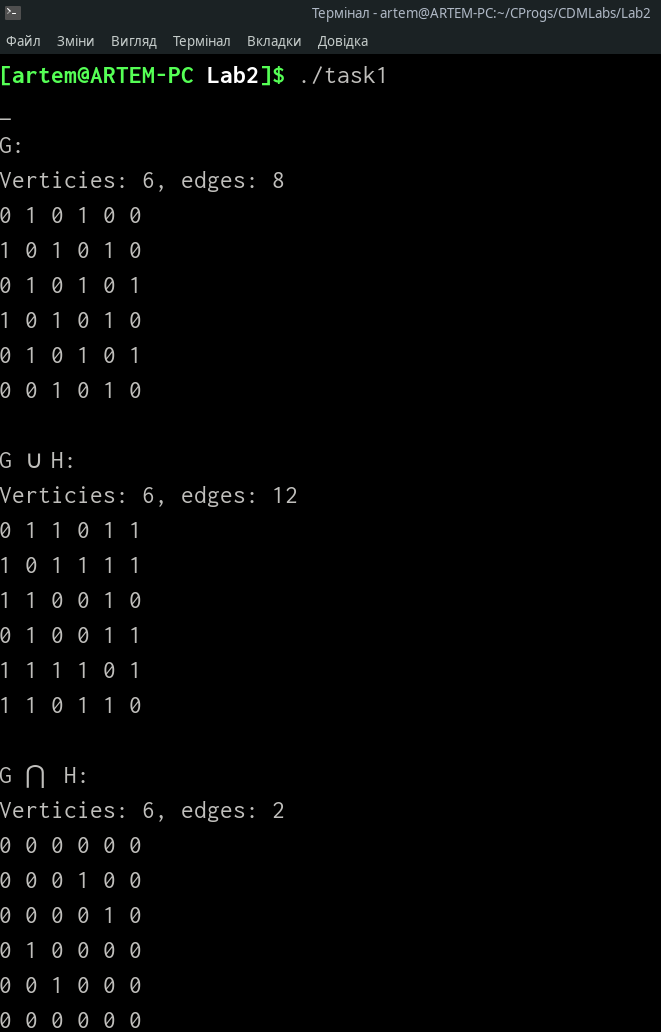
GraphSize, CountEdges(X, GraphSize));

PrintGraph(X, GraphSize);

return 0;

}

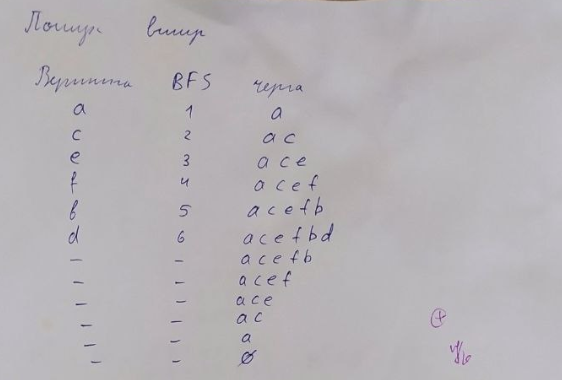
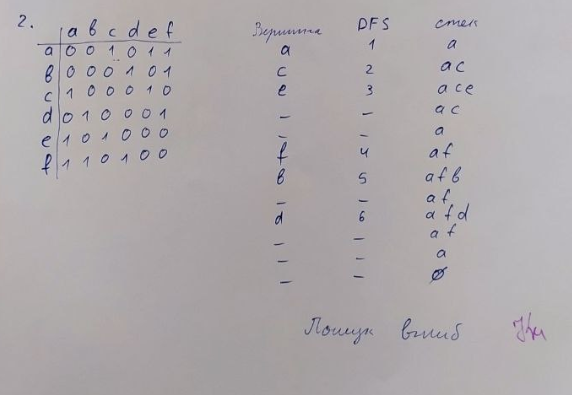
РЕЗУЛЬТАТИ



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 2

Граф G=(V,E) задано матрицею суміжності (таблиця 2.1). Здійснити пошук углиб та пошук ушир у графі G, починаючи з першої вершини. Записати протоколи обходу. Під час обходу при виборі наступної вершини обов’язково брати найменшу відповідно до лексикографічного порядку.

ОПИС ВИКОНАННЯ РОБОТИ



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 3

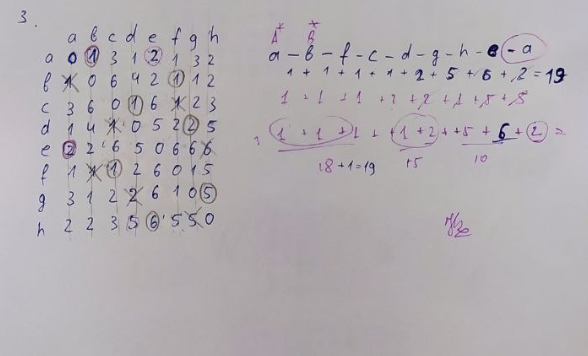
Знайти гамільтоновий цикл (ГЦ) у зваженому графі, який заданий ваговою матрицею

(таблиця 2.2), взявши за вихідну вершину 1. Використайте алгоритм найближчого сусіда.

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує

алгоритм найближчого сусіда для пошуку ГЦ у зваженому графі (таблиця 2.2), взявши за вихідну вершину 1.

ОПИС ВИКОНАННЯ РОБОТИ



ТЕКСТ ПРОГРАМИ

#include <stdio.h>

enum{

GraphSize = 8

};

int notPassed(int v[])

{

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

if (!v[i])

return 1;

return 0;

}

int ShortestRoute(int G[][GraphSize])

{

int v[GraphSize] = {0};

v[0] = 1;

int minId = 0, i = 0, res = 0;

printf("a-");

while (notPassed(v)){

int min = (unsigned short)(-1) / 2;

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

if (i == j || v[j])

continue;

if (G[i][j] < min){

min = G[i][j];

minId = j;

}

}

res += min;

v[minId] = 1;

i = minId;

printf("%c-", 'a' + minId);

}

printf("a\n");

return res + G[i][0];

}

int main()

{

int G[GraphSize][GraphSize] = { {0, 1, 3, 1, 2, 1, 3, 2},

{1, 0, 6, 4, 2, 1, 1, 2},

{3, 6, 0, 1, 6, 1, 2, 3},

{1, 4, 1, 0, 5, 2, 2, 5},

{2, 2, 6, 5, 0, 6, 6, 6},

{1, 1, 1, 2, 6, 0, 1, 5},

{3, 1, 2, 2, 6, 1, 0, 5},

{2, 2, 3, 5, 6, 5, 5, 0},

};

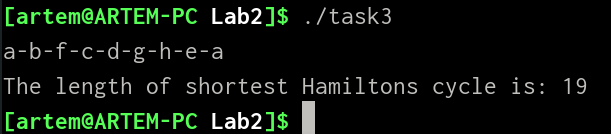
printf("The length of shortest Hamiltons cycle is: %d\n",

ShortestRoute(G));

return 0;

}

**РЕЗУЛЬТАТИ**



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 4

Орграф G задано геометрично (таблиця 2.3). Обчисліть матрицю досяжності орграфа G

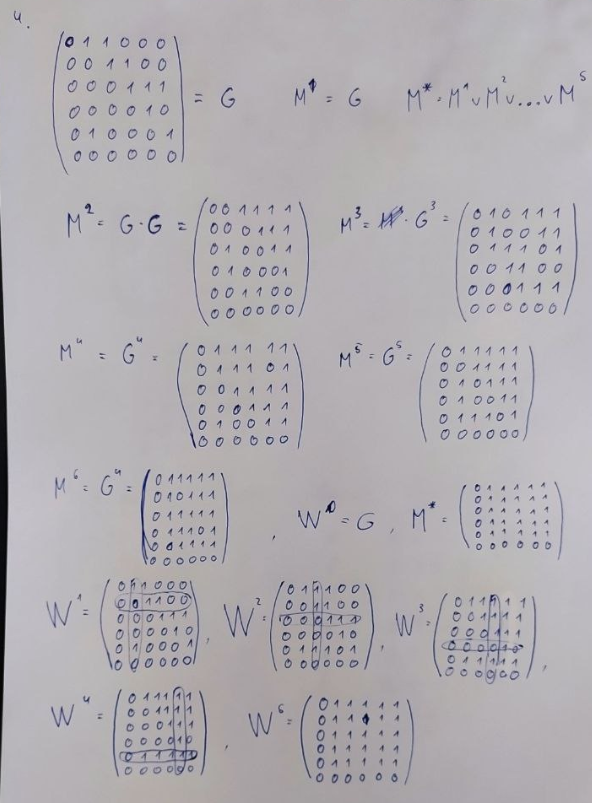
(таблиця 2.3, ваги дуг не брати до уваги) двома способами: за допомогою множення матриці суміжності та алгоритму Воршелла.

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує

обчислення матриці досяжності орграфа G (таблиця 2.3, ваги дуг не брати до уваги) двома

способами: за допомогою піднесення до степеня матриці суміжності та алгоритму Воршелла згідно зі своїм варіантом.

ВИКОНАННЯ РОБОТИ



ТЕКСТ ПРОГРАМИ

#include <stdio.h>

#include "graph.h"

enum{

GraphSize = 6

};

void Warshall(int res[][GraphSize][GraphSize])

{

for(int k = 0; k < GraphSize; k++)

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

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

res[k + 1][i][j] = res[k][i][j] ||

res[k][i][k] &&

res[k][k][j];

}

int main(int argc, char \*argv[])

{

int G[GraphSize][GraphSize] ={ {0, 1, 1, 0, 0, 0},

{0, 0, 1, 1, 0, 0},

{0, 0, 0, 1, 1, 1},

{0, 0, 0, 0, 1, 0},

{0, 1, 0, 0, 0, 1},

{0, 0, 0, 0, 0, 0},

};

int m[GraphSize + 1][GraphSize][GraphSize] = {0};

int adj[GraphSize][GraphSize] = {0};

CopyMatrix(m[0], G, GraphSize);

for(int i = 1; i < GraphSize; ++i){

MultMatrix(m[i], m[i - 1], G, GraphSize);

AddMatrix(adj, adj, m[i], GraphSize);

}

//PrintMatrix(m[argv[1][0] - '0'], GraphSize);

PrintMatrix(adj, GraphSize);

int n[GraphSize][GraphSize][GraphSize] = {0};

CopyMatrix(n[0], G, GraphSize);

Warshall(n);

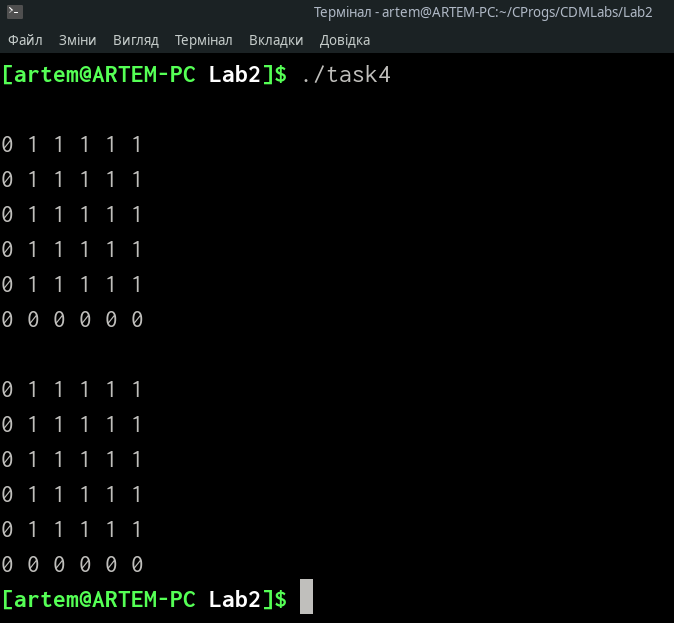
//PrintMatrix(n[argv[1][0] - '0'], GraphSize);

PrintMatrix(n[GraphSize], GraphSize);

return 0;

}

**РЕЗУЛЬТАТИ**



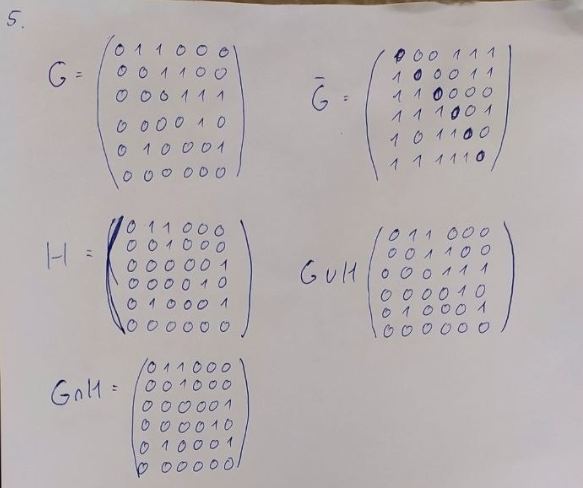
ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 5

Орграф G задано геометично (таблиця 2.3). Вкажіть кількість вузлів і дуг: у орграфі-доповненнідо орграфа G, у орграфі-перетині G∩H та у орграфі-об’єднанні GUH (таблиця 2.3, ваги дуг небрати до уваги). Орграф H – варіант k+1.

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує

обчислення матриць суміжності орграфа-доповнення, орграфа-перетину G∩H та орграфа-об’єднан-ня GUH (таблиця 2.3, ваги дуг не брати до уваги, орграф H – варіант k+1).

**ОПИС ВИКОНАННЯ РОБОТИ**



ТЕКСТ ПРОГРАМИ

#include <stdio.h>

#include "graph.h"

enum{

GraphSize = 6

};

int main()

{

int G[GraphSize][GraphSize] ={ {0, 1, 1, 0, 0, 0},

{0, 0, 1, 1, 0, 0},

{0, 0, 0, 1, 1, 1},

{0, 0, 0, 0, 1, 0},

{0, 1, 0, 0, 0, 1},

{0, 0, 0, 0, 0, 0},

};

int H[GraphSize][GraphSize] ={ {0, 1, 1, 0, 0, 0},

{0, 0, 1, 0, 0, 0},

{0, 0, 0, 0, 0, 1},

{0, 0, 0, 0, 1, 0},

{0, 1, 0, 0, 0, 1},

{0, 0, 0, 0, 0, 0},

};

int GC[GraphSize][GraphSize] ={0},

U[GraphSize][GraphSize] ={0},

X[GraphSize][GraphSize] ={0};

ConstructComplementary(GC, G, GraphSize);

ConstructUnion(U, G, H, GraphSize);

ConstructCrossing(X, G, H, GraphSize);

printf("\_\n");

printf("G:\n");

printf("Verticies: %d, edges: %d",

GraphSize, CountEdges(GC, GraphSize));

PrintGraph(GC, GraphSize);

printf("\nG ∪ H:\n");

printf("Verticies: %d, edges: %d",

GraphSize, CountEdges(U, GraphSize));

PrintGraph(U, GraphSize);

printf("\nG ⋂ H:\n");

printf("Verticies: %d, edges: %d",

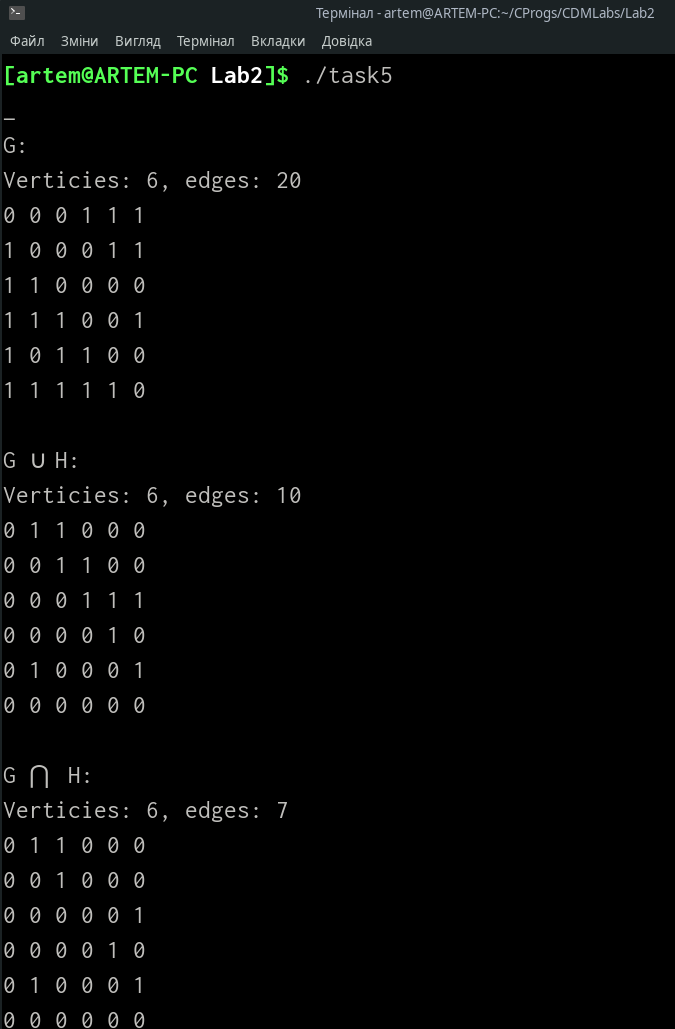
GraphSize, CountEdges(X, GraphSize));

PrintGraph(X, GraphSize);

return 0;

}

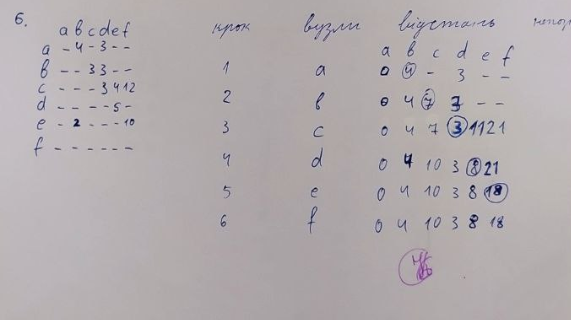
**РЕЗУЛЬТАТИ**



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 6

За допомогою алгоритму Дейкстри знайдіть найкоротші шляхи від вершини х1 зваженого орграфа G до будь-якої іншої (таблиця 2.3).

ОПИС ВИКОНАННЯ РОБОТИ



ВИСНОВКИ

Графи використовуються для подання звʼязків між точками. За допомогою різних алгоритмів можна визначати найменшу відстань між точками, можливі шляхи і т. д.