МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту

Лабораторна робота №5
з курсу "Дискретна математика "


Виконав:
ст. гр. КН-110
Холод Ігор

Викладач:
Мельникова Н.І.


Львів – 2018

**Тема: Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи**

**Мета роботи: набуття практичних вмінь та навичок з використання алгоритму Дейкстри.**

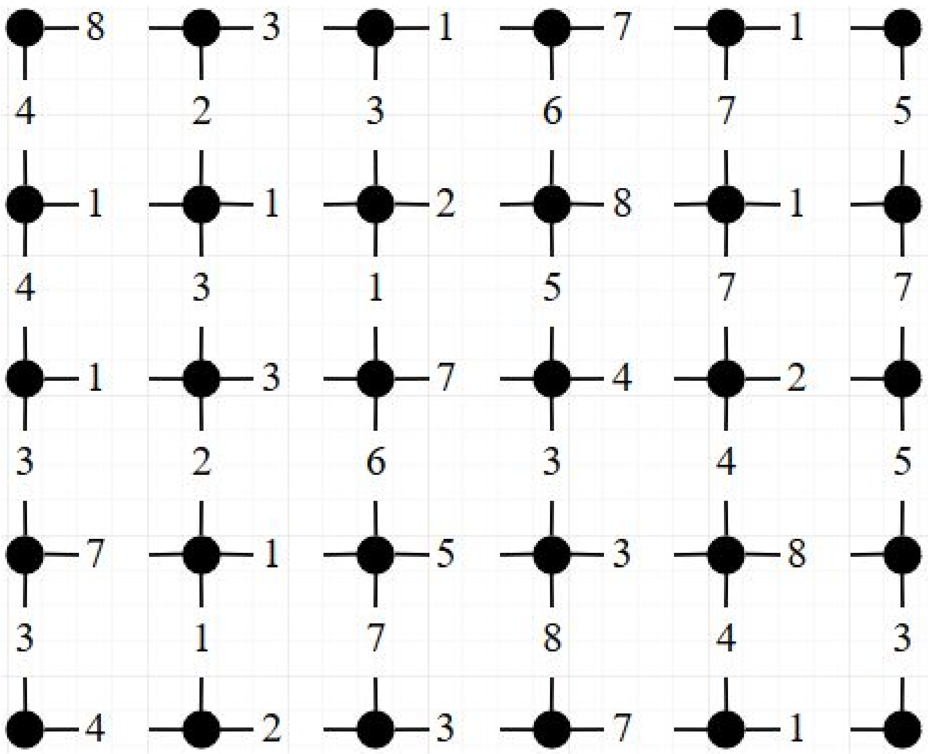
**Теоретичні відомості:**
Задача знаходження найкоротшого шляху з одним джерелом полягає у знаходженні найкоротших(мається на увазі найоптимальніших за вагою) шляхів від деякої вершини(джерела) до всіх вершин графа G . Для розв’язку цієї задачі використовується «жадібний» алгоритм, який називається алгоритмом Дейкстри.
**«Жадібними»** називаються алгоритми, які на кожному кроці вибирають оптимальний із можливих варіантів.

**Плоскі і планарні графи**
**Плоским** графом називається граф, вершини якого є точками площини, а ребра – безперервними лініями без самоперетинань, що з'єднують відповідні вершини так, що ніякі два ребра не мають спільних точок крім інцидентної їм обом вершини. Граф називається планарним, якщо він є ізоморфним плоскому графу.
**Гранню** плоского графа називається максимальна по включенню множина точок площини, кожна пара яких може бути з'єднана жордановою кривою, що не перетинає ребра графа. Границею грані будемо вважати множину вершин і ребер, що належать цій грані.

**Алгоритм $\gamma$ укладання графа** $G$ являє собою процес послідовного приєднання до деякого укладеного підграфа $\overline{G}$ графа G нового ланцюга, обидва кінці якого належать $\overline{G}$. При цьому в якості початкового плоского графа $\overline{G}$ вибирається будь-який простий цикл графа G . Процес продовжується доти, поки не буде побудовано плоский граф, ізоморфний графові G , або приєднання деякого ланцюга виявиться неможливим. В останньому випадку граф G не є планарным.

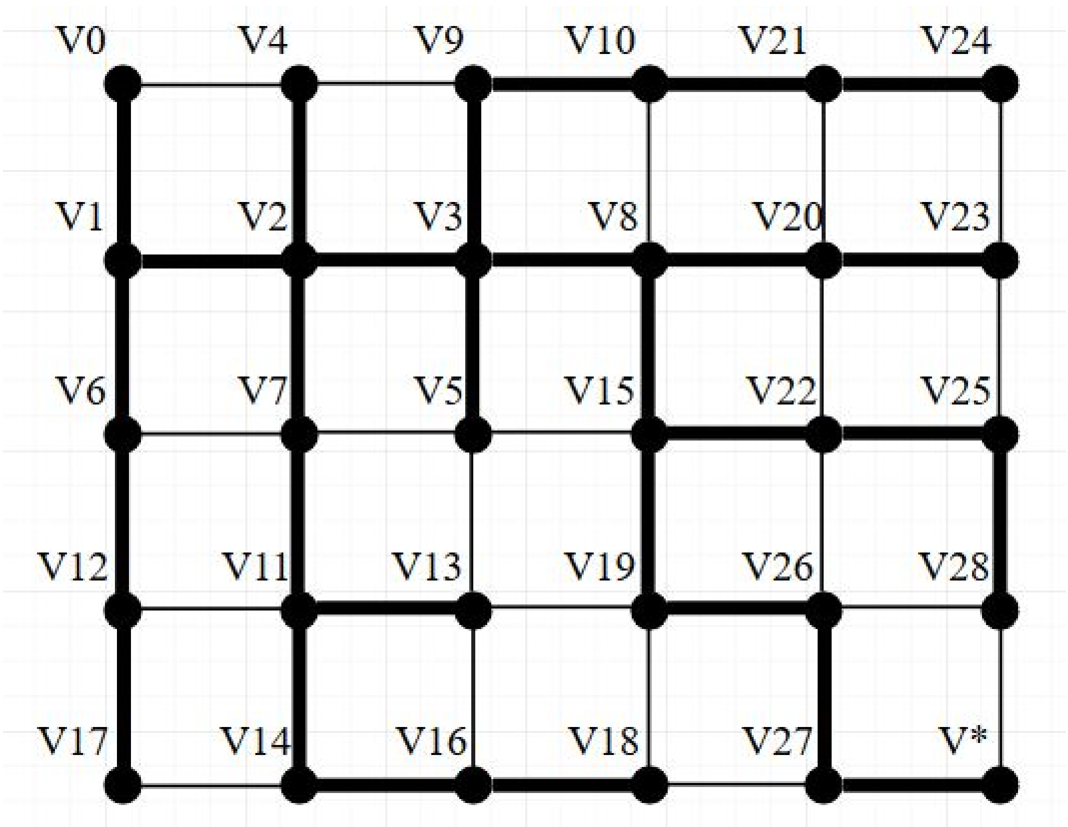
**Завдання:**

**1)За допомогою алгоритму Дейкстра знайти найкоротший шлях у графі поміж парою вершин $V_0$ і $V^*$ .**



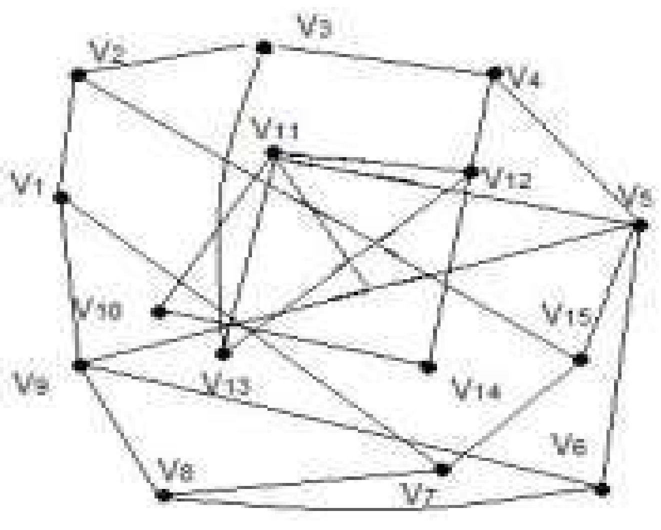

Будемо позначати найближчі вершини $v_1$, $v_2$, $v_3$, ... у порядку їхньої появи.

$l(v_1) = 4$, $l(v_2) = 5$, $l(v_3) = 6$, $l(v_4) = 7$, $l(v_5) = 7$, $l(v_6) = 8$, $l(v_7) = 8$, $l(v_8) = 8$, $l(v_9) = 9$, $l(v_{10}) = 10$, $l(v_{11}) = 10$, $l(v_{12}) = 11$, $l(v_{13}) = 11$, $l(v_{14}) = 11$, $l(v_{15}) = 13$, $l(v_{16}) = 13$, $l(v_{17}) = 14$, $l(v_{18}) = 16$, $l(v_{19}) = 16$, $l(v_{20}) = 16$, $l(v_{21}) = 17$, $l(v_{22}) = 17$, $l(v_{23}) = 17$, $l(v_{24}) = 18$, $l(v_{25}) = 19$, $l(v_{26}) = 19$, $l(v_{27}) = 23$, $l(v_{28}) = 24$, $l(v^*) = 24$.

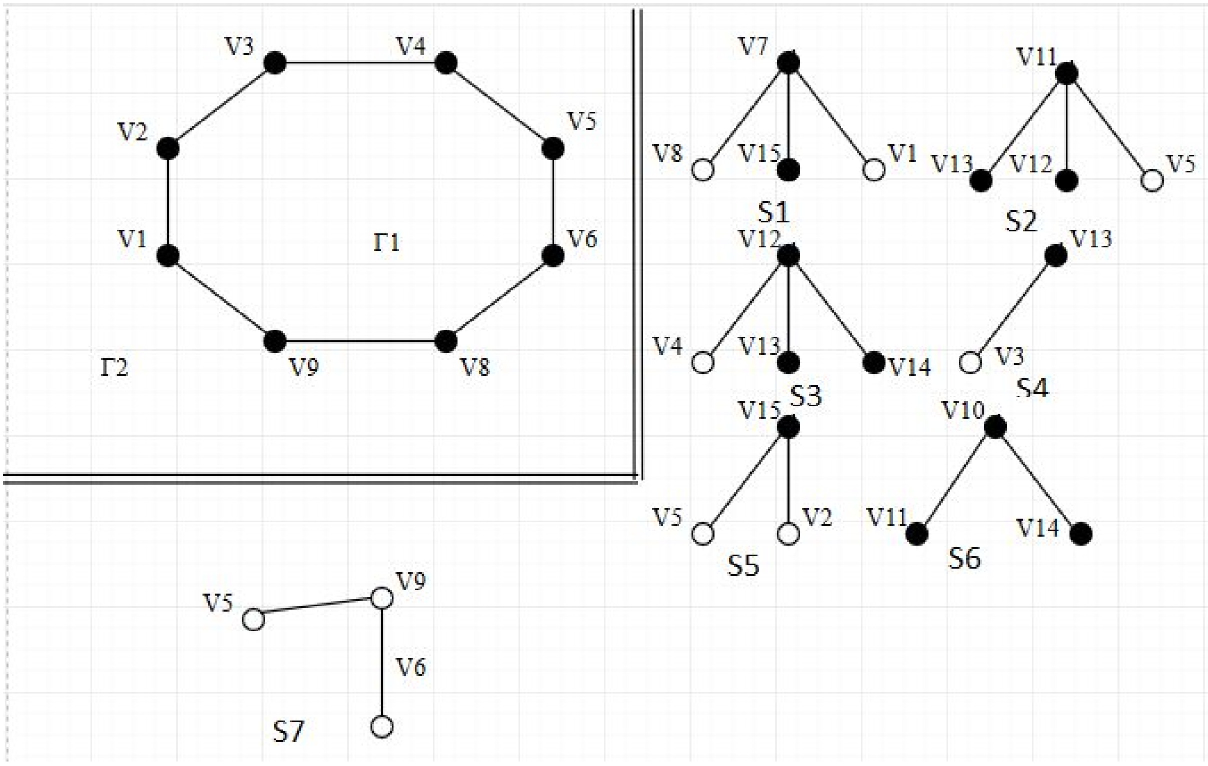



Дерево найближчих вершин виділено на рисунку жирними лініями і є кістяковим деревом, тому що містить усі вершини графа. Шуканий найкоротший ланцюг: $[v_0,\ v_1,\ v_2,\ v_3,\ v_8,\ v_{15},\ v_{19},\ v_{26},\ v_{27},\ v^*]$, довжина ланцюга $l = l(v^*) = 24$.

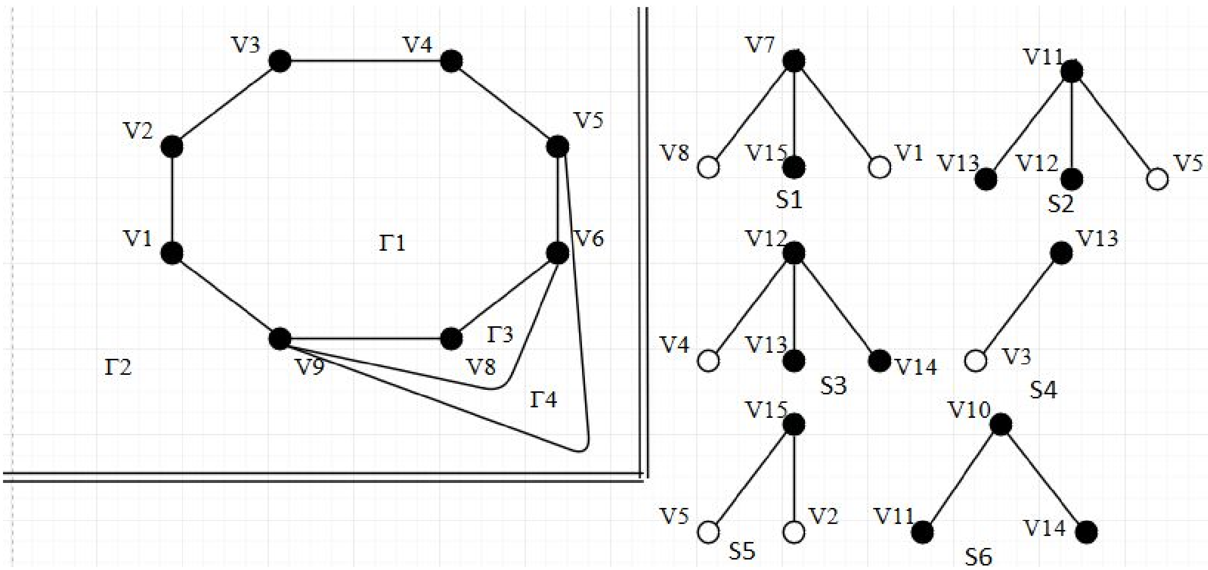
За допомогою $\gamma$-алгоритма зробити укладку графа у площині, або довести що вона неможлива.



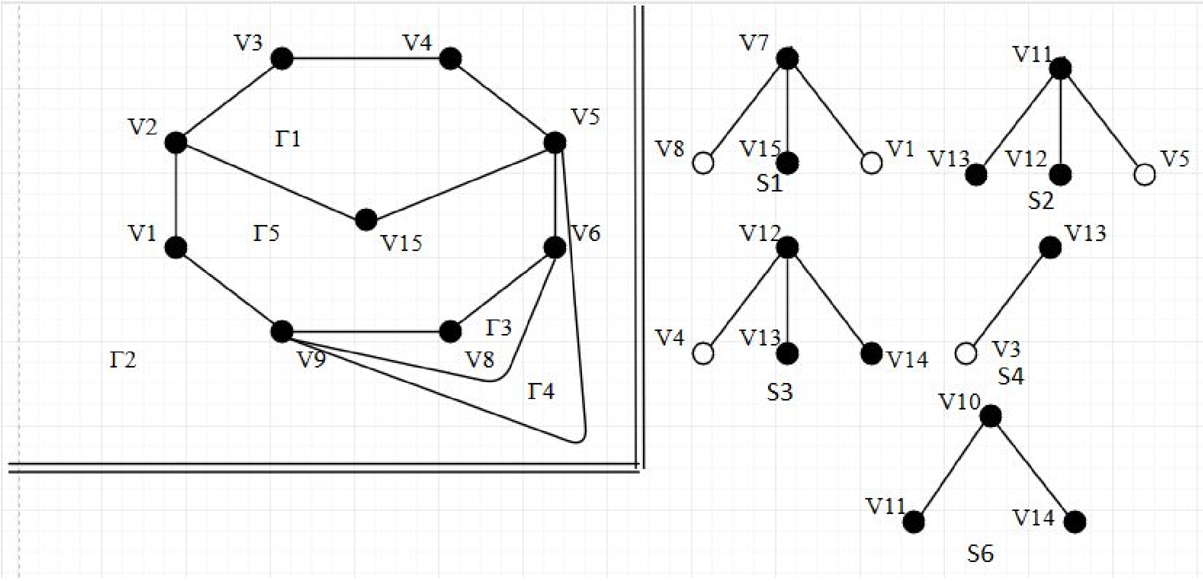
Розв’язання: Укладемо спочатку цикл [1, 2, 3, 4, 5, 6, 8, 9] що розіб’є площину на дві грані Г1 та Г2. Запишемо сегменти S1, S2, S3, S4, S5, S6 та S7.



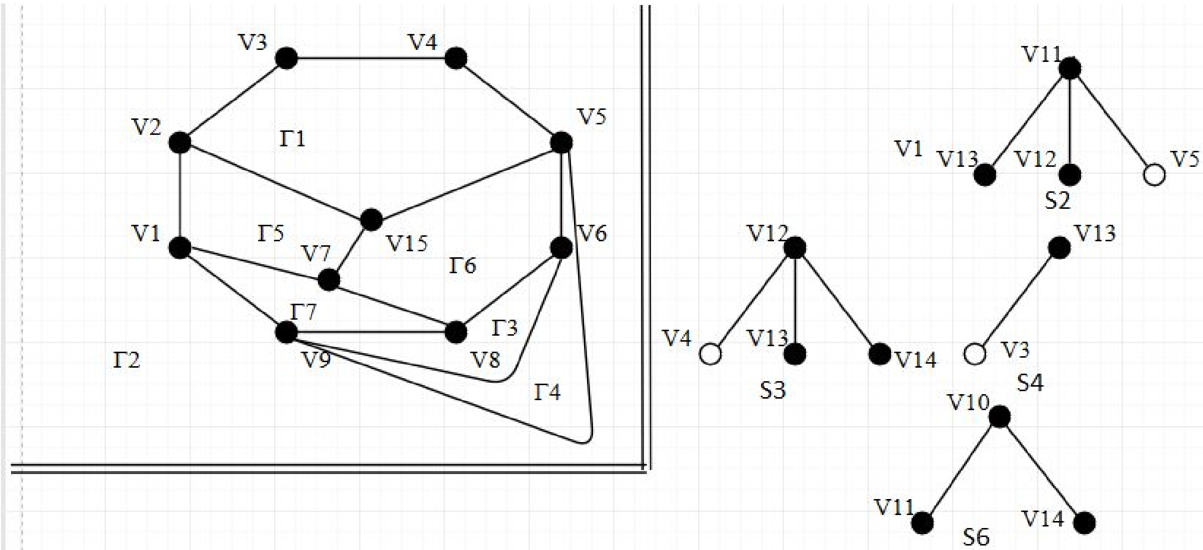

Укладемо сегмент S7 в грань Г2. Таким чином Г2 буде розбита на грані Г2, Г3 та Г4.



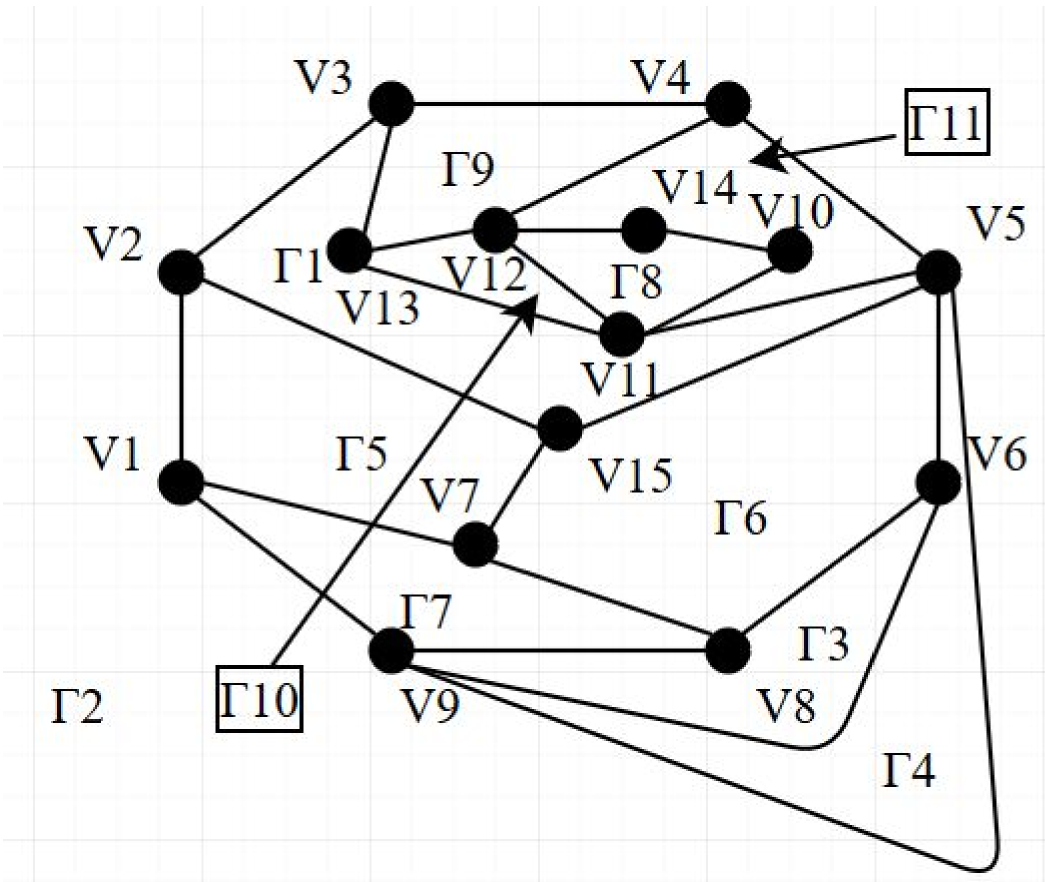

Тоді укладемо сегмент S5 у грань Г1, яка розіб’ється на грані Г1 та Г5.



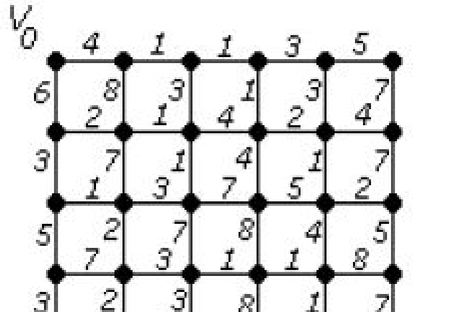

Укладаємо сегмент S1 у грань Г5. Тоді ця грань розіб’ється на грані Г5, Г6 та Г7.




Таким самим чином послідовно укладемо решту сегментів у грань Г1, які в кінцевому результаті розіб’ють цю грань на Г1, Г8, Г9, Г10 та Г11. Одержуємо укладання графа на площині.




**Завдання 2:**Написати програму, яка реалізує алгоритм Дейкстри знаходження найкоротшого шляху між парою вершин у графі. Протестувати розроблену програму на графі згідно свого варіанту.




Код програми:

```
#include<stdio.h>

#define inf 10000000

int G[30][30];

void fill_graph(int); //функція вводу матриці відношення
void dejkstry(int, int); //алгоритм Дейкстри
```

```

int main()
{
	int start, n;

	printf("Enter amount of vertexes:");
	scanf("%d", &n);

	fill_graph(n);

	printf("\nEnter start vertex:");
	scanf("%d", &start);
	dejkstry(n, start);
	printf("\n");

	return 0;
}


void fill_graph(int n)
{
	//заповнюємо матрицю відношення нулями
	//маючи наувазі, що поки-що граф немає ребер
	for (int i = 0; i < n; i++)
	for (int j = 0; j < n; j++)
	G[i][j] = 0;

	//вводимо вагу кожного ребра
	//1-а вершина -- 2-а вершина -- вага ребра між ними
	printf("Enter graph (type 999 to stop entering):\n");
	int st, nd, weight;
	while(1)
	{
	//вводимо 1-у вершину
	printf("1st vertex: ");
	scanf("%d", &st);
	//умова для перевірки правильності введених даних
	//оскільки вершини пронумеровані від 0 до n - 1
	//а 999 - задана умова виходу з циклу
	while (((st < 0) || (st > n - 1)) && (st != 999))
	{
	printf("Retry: ");
	scanf ("%d", &st);
	}
	if (st == 999)
	break;
```

```
        //вводимо 2-у вершину
        printf("2nd vertex: ");
        scanf("%d", &nd);
        while (((nd < 0) || (nd > n - 1)) && (nd != 999))
        {
        printf("Retry: ");
        scanf("%d", &nd);
        }
        if (nd == 999)
        break;
        //вводимо вагу ребра між вершинами
        printf("Weight: ");
        scanf("%d", &weight);
        while (weight <= 0)
        {
        printf("Retry: ");
        scanf("%d", &weight);
        }
        if (weight == 999)
        break;

        //заповнюємо елемент G[1-а вершина][2-а вершина]
        //матриця симетрична, тому координати можна міняти місцями
        if ((st != 999) && (nd != 999) && (weight != 999))
        {
        G[st][nd] = weight;
        G[nd][st] = weight;
        }
        }
}

void dejkstry(int n, int starter)
{

        int weights[30][30], distance[30], prev[30];
        int visited[30], count, min_distance, next, i, j;

        //weights - ваги кожного ребра (немає ребра - відстань нескінченна)
        //prev[] - попередні вершини зі шляху
        //count - кількість вже пройдених вершин
        //visited - уже відвідані вершини

        //створюємо матрицю, що буде містити вагу ребер
        for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
                if (G[i][j] == 0)
```

```
            weights[i][j] = inf;
            else
            weights[i][j] = G[i][j];

    //заповнюємо prev, distance and visited
    for(i = 0; i < n; i++)
    {
    distance[i] = weights[starter][i];
    prev[i] = starter;
    visited[i] = 0;
    }

    //відстань від початкової вершини до неї ж - 0
    distance[starter] = 0;
    //початкова вершина автоматично стає відвідувана
    visited[starter] = 1;
    //одна вершина вже відвідана
    count = 1;

    while (count < n - 1) //поки не пройдені усі вершини
    {
    //на початку кожного циклу мінімальна відстань
    //до не відвіданих ребер, оскільки граф оновився
    //стає нескінченною для подальшого порівняння
    min_distance = inf;

    //next зберігає значення вершини на найменшій відстані від заданої
    for (i = 0; i < n; i++)
            //якщо відстань до невідвіданої вершини є меншою
            //ніж попередня мінімальна відстань
            if ((distance[i] < min_distance) && (visited[i] == 0))
            {
            //то зберігаємо відстань у min_sdistance
            //і цю вершину у next
            min_distance = distance[i];
            next = i;
            }
            //наступна вершина вже відвідана
            visited[next] = 1;

            //перерахунок найменших відстаней до кожної з невідвіданих вершин
            for (i = 0; i < n; i++)
            if (visited[i] == 0)
            //рахуємо відстані, починаючи з щойно доданої вершини
            //оскільки лише вони могли змігнитися
            if (min_distance + weights[next][i] < distance[i])
```

```
            {
                distance[i] = min_distance + weights[next][i];
                //і записуємо додану вершину як попередню до найближчої
                prev[i] = next;
            }
        count++;
        }
        //друкуємо відстань і шлях для кожної вершини...
        for (i = 0; i < n; i++)
        if(i != starter) //...окрім початкової
        {
            printf("\nDistance to vertex %d = %d;", i, distance[i]);
            printf("\nPath = %d;", i);

            j = i;
            do
            {
            //в ланцюговому порядку друкуємо кожну з вершин шляху
            j = prev[j];
            printf(" <- %d",j);
            } while (j != starter);
        }
    }
```

Результат роботи програми (користувач вводить граф: одна вершина - друга вершина - вага ребра між цими вершинами, а також початкову вершину), а програма виводить відстань до усіх інших вершин і шляхи до цих вершин.

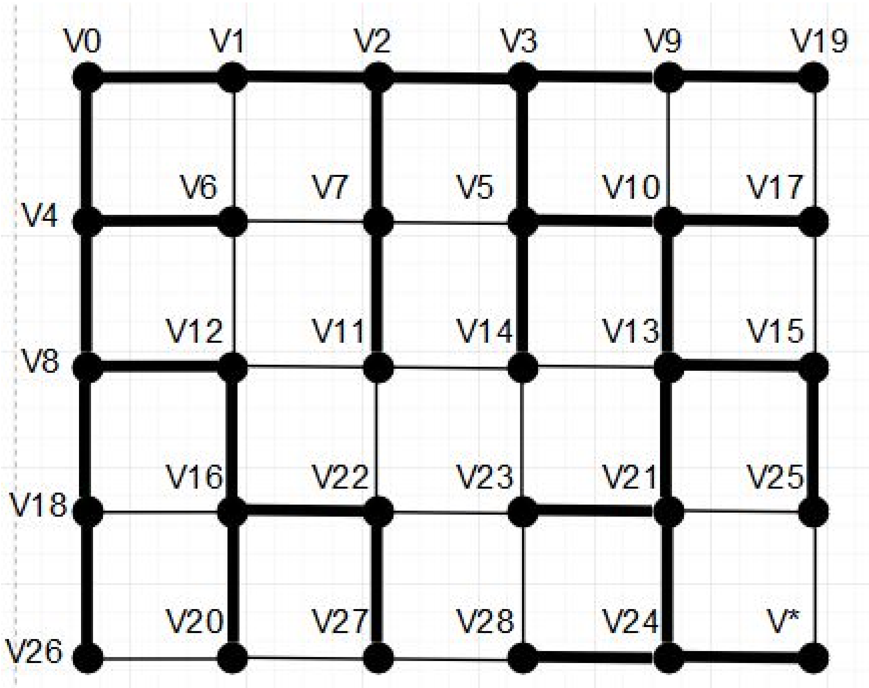
```
Enter start vertex:0

Distance to vertex 1 = 4;
Path = 1; <- 0
Distance to vertex 2 = 5;
Path = 2; <- 1 <- 0
Distance to vertex 3 = 6;
Path = 3; <- 2 <- 1 <- 0
Distance to vertex 4 = 9;
Path = 4; <- 3 <- 2 <- 1 <- 0
Distance to vertex 5 = 14;
Path = 5; <- 4 <- 3 <- 2 <- 1 <- 0
Distance to vertex 6 = 6;
Path = 6; <- 0
Distance to vertex 7 = 8;
Path = 7; <- 6 <- 0
Distance to vertex 8 = 8;
Path = 8; <- 2 <- 1 <- 0
Distance to vertex 9 = 7;
Path = 9; <- 3 <- 2 <- 1 <- 0
Distance to vertex 10 = 9;
Path = 10; <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 11 = 13;
Path = 11; <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
```

```
Distance to vertex 14 = 11;
Path = 14; <- 8 <- 2 <- 1 <- 0
Distance to vertex 15 = 11;
Path = 15; <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 16 = 10;
Path = 16; <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 17 = 12;
Path = 17; <- 16 <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 18 = 14;
Path = 18; <- 12 <- 6 <- 0
Distance to vertex 19 = 12;
Path = 19; <- 13 <- 12 <- 6 <- 0
Distance to vertex 20 = 15;
Path = 20; <- 19 <- 13 <- 12 <- 6 <- 0
Distance to vertex 21 = 15;
Path = 21; <- 22 <- 16 <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 22 = 14;
Path = 22; <- 16 <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 23 = 17;
Path = 23; <- 17 <- 16 <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 24 = 17;
Path = 24; <- 18 <- 12 <- 6 <- 0
Distance to vertex 25 = 14;
```

```
Distance to vertex 26 = 18;
Path = 26; <- 20 <- 19 <- 13 <- 12 <- 6 <- 0
Distance to vertex 27 = 18;
Path = 27; <- 28 <- 22 <- 16 <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 28 = 15;
Path = 28; <- 22 <- 16 <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
Distance to vertex 29 = 21;
Path = 29; <- 28 <- 22 <- 16 <- 10 <- 9 <- 3 <- 2 <- 1 <- 0
```

Найменше остове дерево для тестового графа має такий вигляд:




**Висновок:** я зрозумів і освоїв “жадібний” алгоритм Дейкстри для пошуку найкоротшого шляху між заданою вершиною і рештою вершин графа, навчився застосовувати його на практиці,  а також реалізовувати його програмно.