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

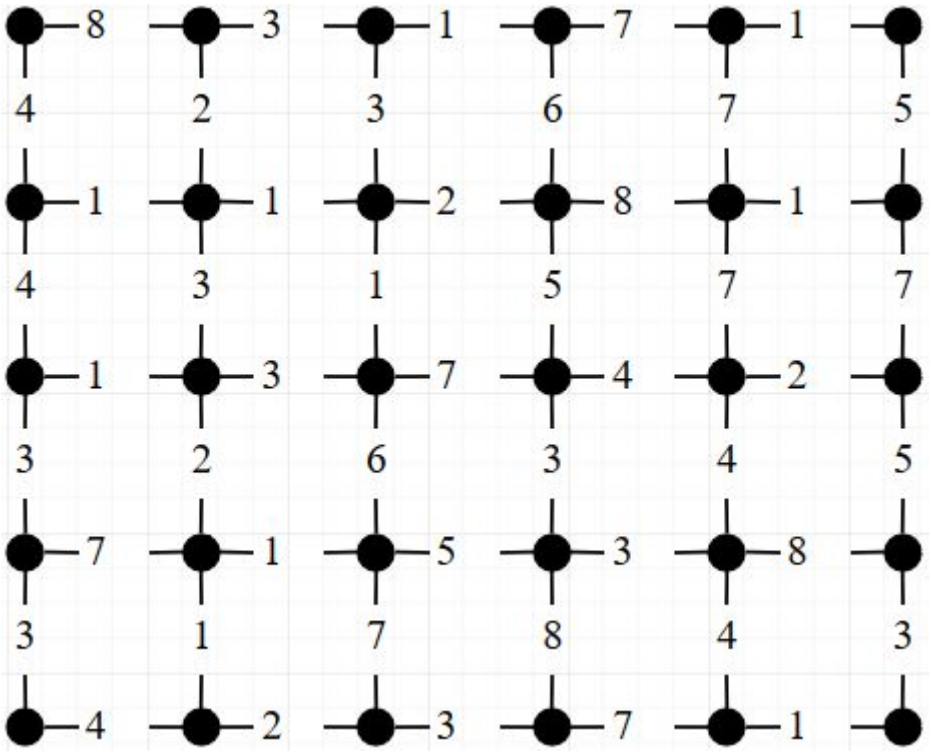
Плоским графом називається граф, вершини якого є точками площини, а ребра – безперервними лініями без самоперетинань, що з'єднують відповідні вершини так, що ніякі два ребра не мають спільних точок крім інцидентної їм обом вершини. Граф називається планарним, якщо він є ізоморфним плоскому графу.

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 γ 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 планарним.

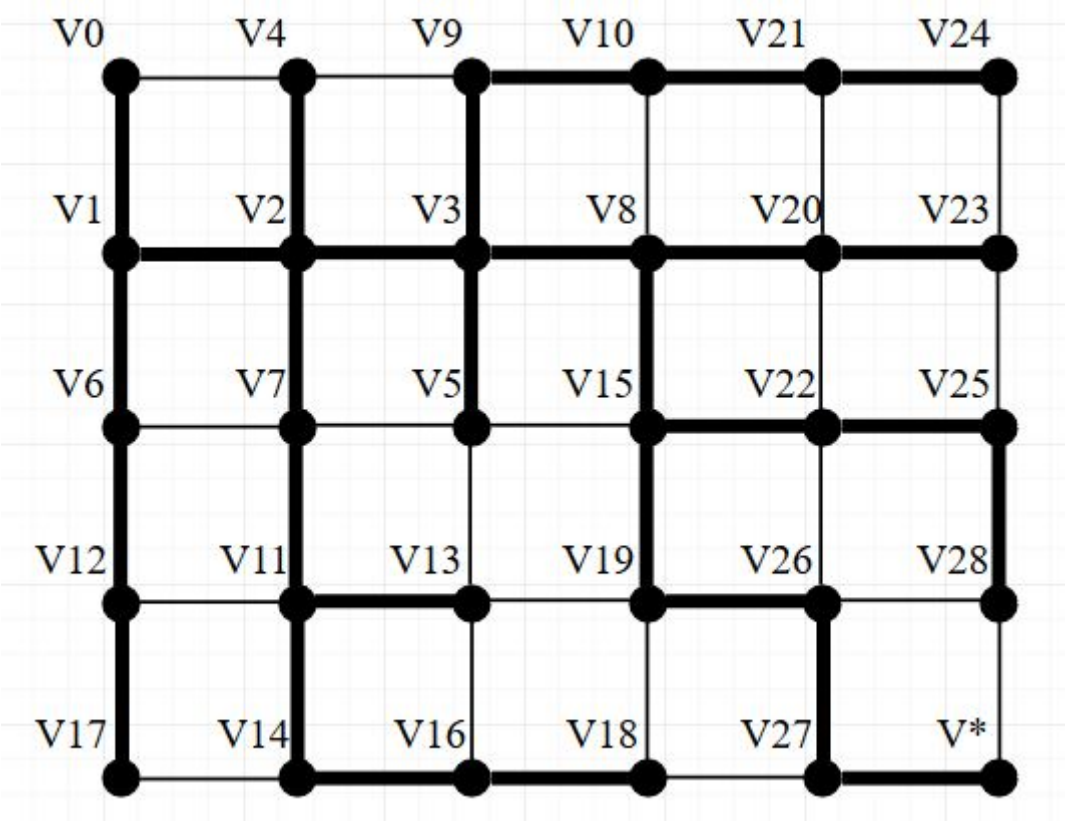
Завдання:

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



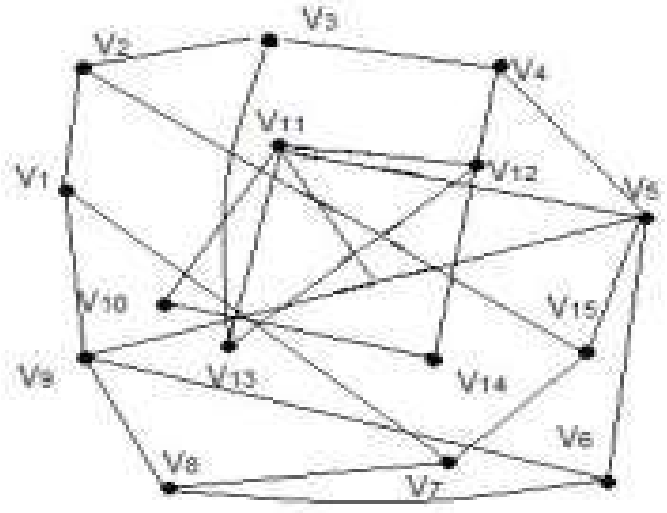
Будемо позначати найближчі вершини $v_1,\ v_2,\ v_3,\ ...$ у порядку їхньої появи.

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

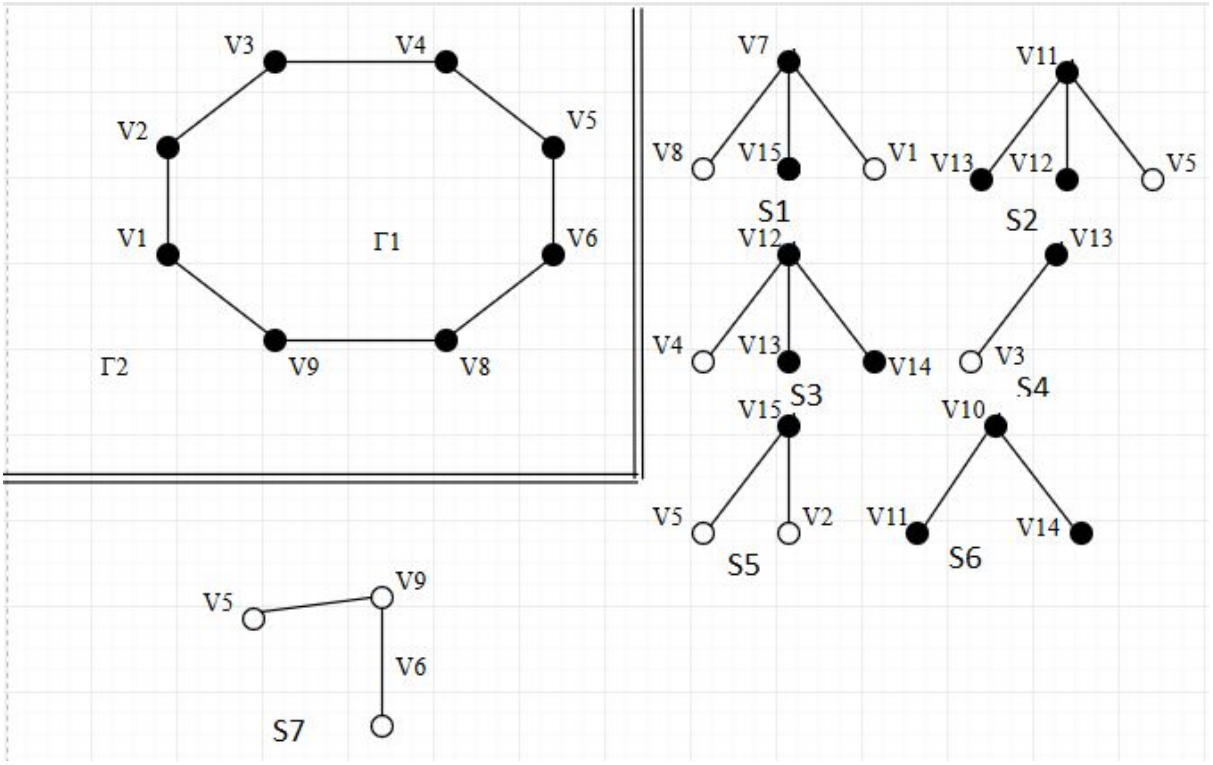


Дерево найближчих вершин виділено на рисунку жирними лініями і є кістяковим деревом, тому що містить усі вершини графа. Шуканий найкоротший ланцюг: $[v_0,\ v_1,\ v_2,\ v_3,\ v_8,\ v_{15},\ v_{19},\ v_{26},\ v_{27},\ v^*]$, довжина ланцюга $l = l(v^*) = 24$.

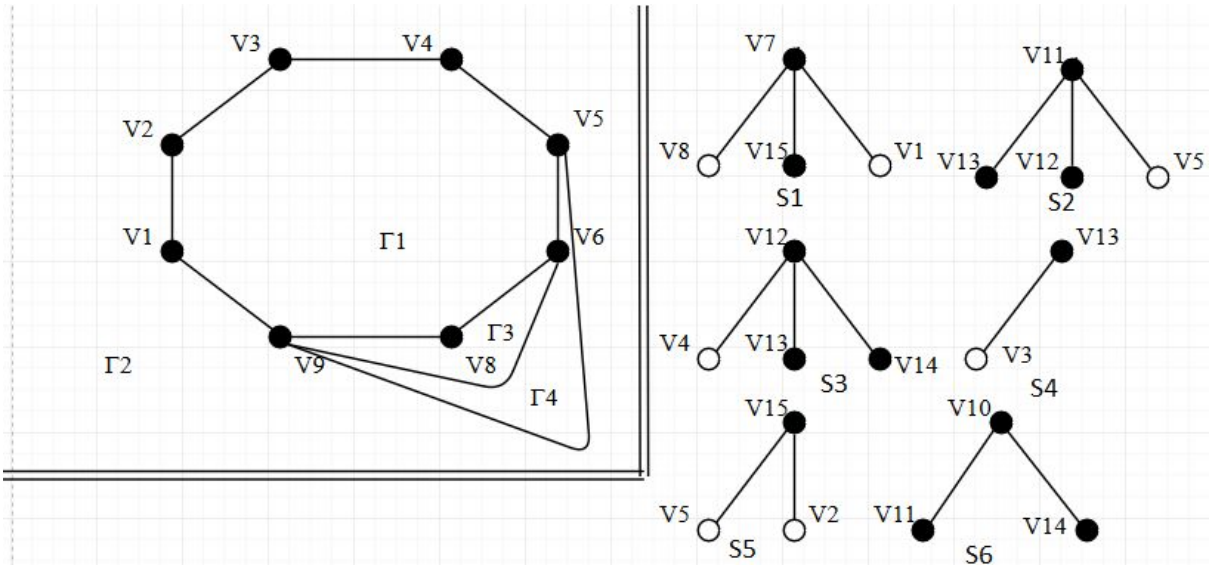
За допомогою γ -алгоритма зробити укладку графа у площині, або довести що вона неможлива.



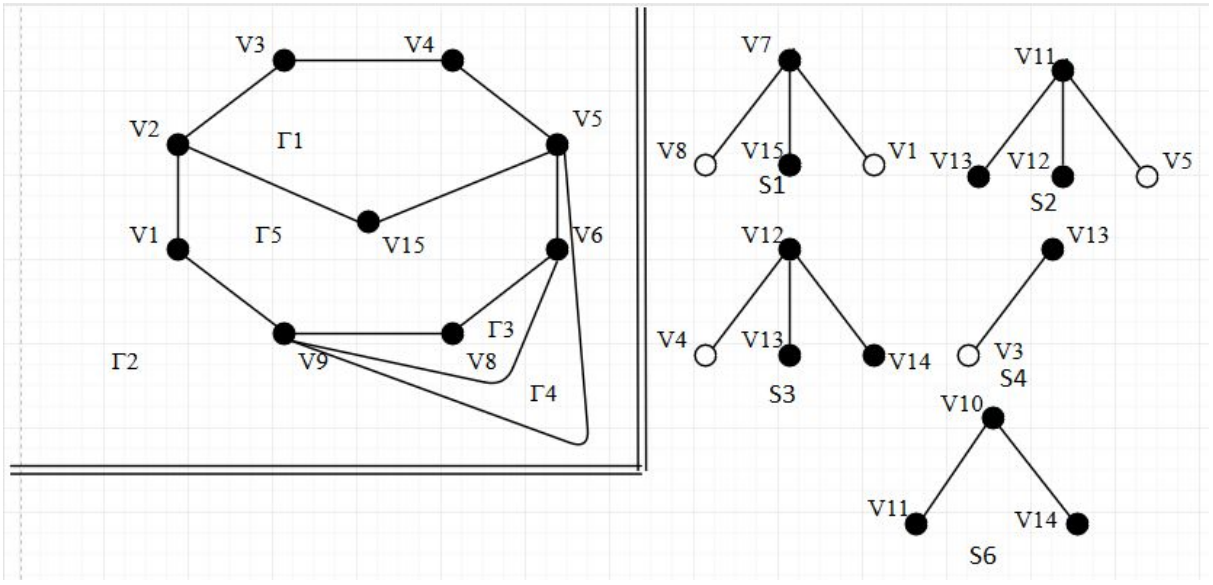
Розв’язання: Укладемо спочатку цикл [1, 2, 3, 4, 5, 6, 8, 9] що розіб’є площину на дві грані Г1 та Г2. Запишемо сегменти S1, S2, S3, S4, S5, S6 та S7.



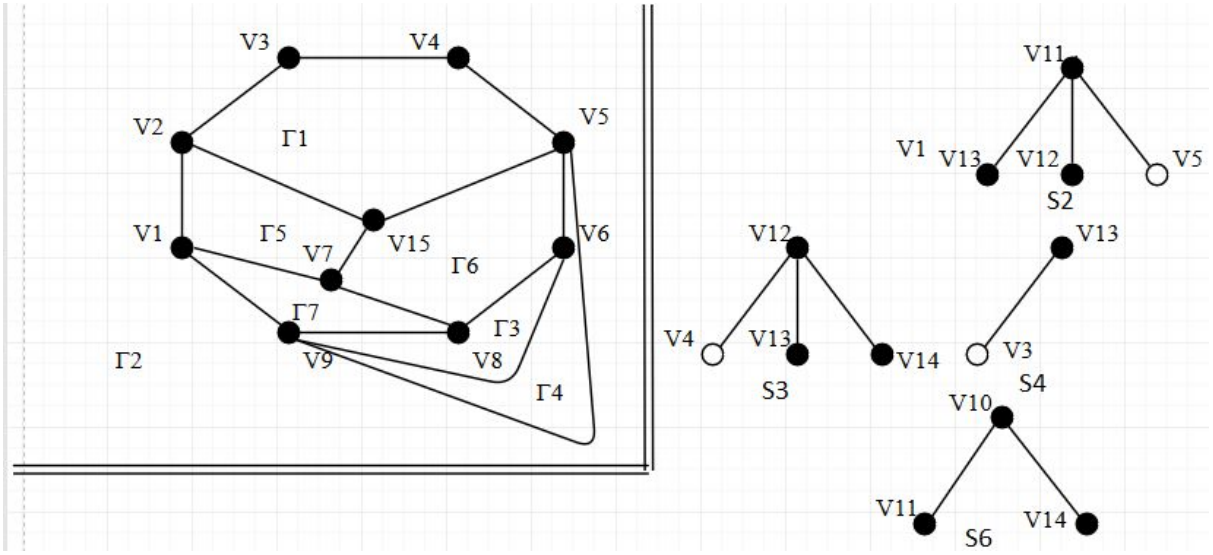
Укладемо сегмент S7 в грань Г2. Таким чином Г2 буде розбита на грані Г2, Г3 та Г4.



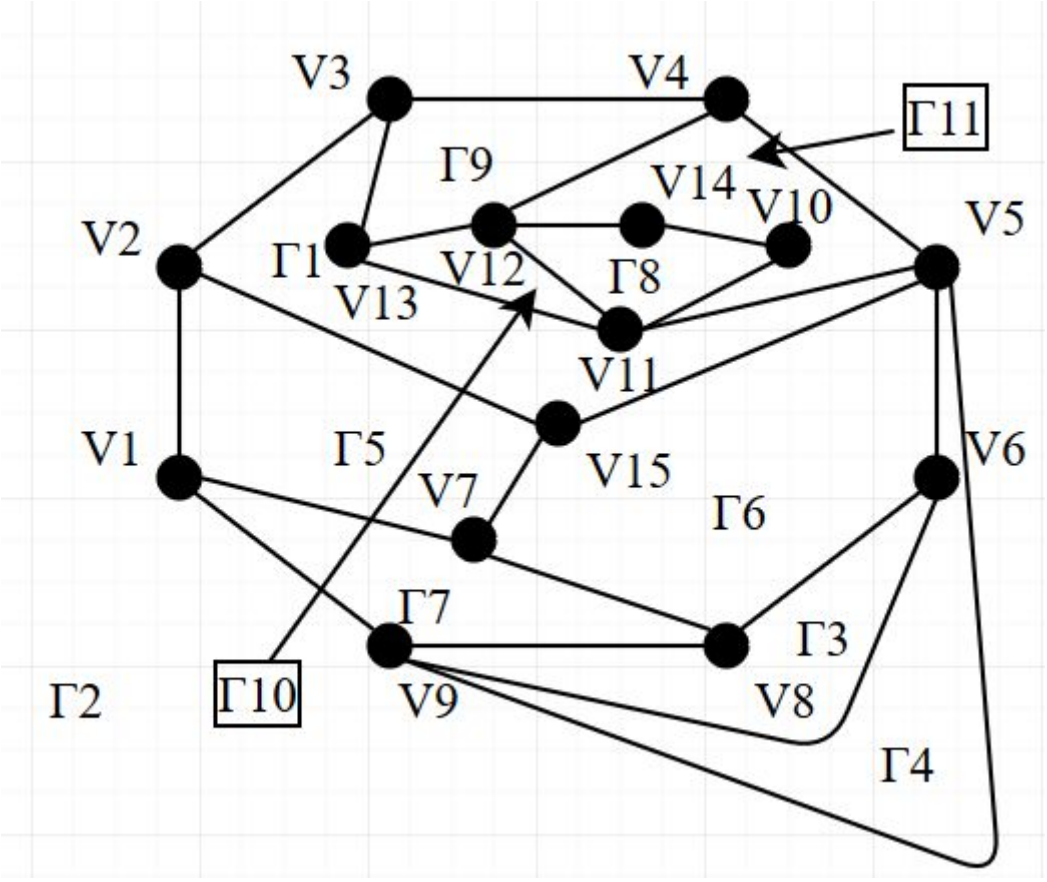
Тоді укладемо сегмент S5 у грань Г1, яка розіб’ється на грані Г1 та Г5.



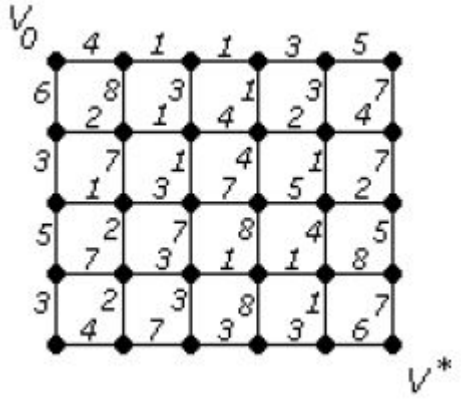
Укладаємо сегмент S1 у грань Г5. Тоді ця грань розіб’ється на грані Г5, Г6 та Г7.



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:

```
1. #include<stdio.h>
2.
3. #define inf 10000000
4.
5. int G[30][30];
6.
7. void fill_graph(int);
8. void dejkstry(int, int);
```

```
9.
10. int main()
11. {
12.         int start, n;
13.
14.         printf("Enter amount of vertexes:");
15.         scanf("%d", &n);
16.
17.         fill_graph(n);
18.
19.         printf("\nEnter start vertex:");
20.         scanf("%d", &start);
21.         dejkstry(n, start);
22.         printf("\n");
23.
24.         return 0;
25. }
26.
27.
28. void fill_graph(int n)
29. {
30.         //initializating adjancency matrix to 0
31.         for (int i = 0; i < n; i++)
32.         for (int j = 0; j < n; j++)
33.         G[i][j] = 0;
34.
35.         //entering weights of each edge
36.         //1st vertex - 2nd vertex - weight
37.         printf("Enter graph (type 999 to stop entering):\n");
38.         int st, nd, weight;
39.         while(1)
40.         {
41.         printf("1st vertex: ");
42.         scanf("%d", &st);
43.         while (((st < 0) || (st > n - 1)) && (st != 999))
44.         {
45.         printf("Retry: ");
46.         scanf ("%d", &st);
47.         }
48.         if (st == 999)
49.         break;
50.         printf("2nd vertex: ");
51.         scanf("%d", &nd);
52.         while (((nd < 0) || (nd > n - 1)) && (nd != 999))
53.         {
54.         printf("Retry: ");
```

```
55.             scanf("%d", &nd);
56.             }
57.             if (nd == 999)
58.             break;
59.             printf("Weight: ");
60.             scanf("%d", &weight);
61.             while (weight <= 0)
62.             {
63.             printf("Retry: ");
64.             scanf("%d", &weight);
65.             }
66.             if (weight == 999)
67.             break;
68.             if ((st != 999) && (nd != 999) && (weight != 999))
69.             {
70.             G[st][nd] = weight;
71.             G[nd][st] = weight;
72.             }
73.             }
74. }
75. void dejkstry(int n, int starter)
76. {
77.
78.             int weights[30][30], distance[30], prev[30];
79.             int visited[30], count, min_distance, next, i, j;
80.
81.             //weights - weights of each edge (no edge = infinity)
82.             //prev[] - previous vertex of the path
83.             //count - number of vertex
84.             //visited - vertex that are already visitied
85.
86.             //create the weights matrix
87.             for (i = 0; i < n; i++)
88.             for (j = 0; j < n; j++)
89.             if (G[i][j] == 0)
90.             weights[i][j] = inf;
91.             else
92.             weights[i][j] = G[i][j];
93.
94.             //initializating prev, distance and visited
95.             for(i = 0; i < n; i++)
96.             {
97.             distance[i] = weights[starter][i];
98.             prev[i] = starter;
99.             visited[i] = 0;
100.            }
```

```
101.
102.        distance[starter] = 0;
103.        visited[starter] = 1;
104.        count = 1;
105.
106.        while (count < n - 1)
107.        {
108.        min_distance = inf;
109.
110.        //next gives the node at minimum distance
111.        for (i = 0; i < n; i++)
112.                if ((distance[i] < min_distance) && (visited[i] == 0))
113.                {
114.                min_distance = distance[i];
115.                next = i;
116.                }
117.                visited[next] = 1;
118.
119.                //check for the better path
120.                for (i = 0; i < n; i++)
121.                if (visited[i] == 0)
122.                if (min_distance + weights[next][i] < distance[i])
123.                {
124.                        distance[i] = min_distance + weights[next][i];
125.                        prev[i] = next;
126.                }
127.        count++;
128.        }
129.
130.        //print the path and distance of each vertex
131.        for (i = 0; i < n; i++)
132.        if(i != starter)
133.        {
134.                printf("\nDistance to vertex %d = %d;", i, distance[i]);
135.                printf("\nPath = %d;", i);
136.
137.                j = i;
138.                do
139.                {
140.                j = prev[j];
141.                printf(" <- %d",j);
142.                } while (j != starter);
143.        }
144.    }
```

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.

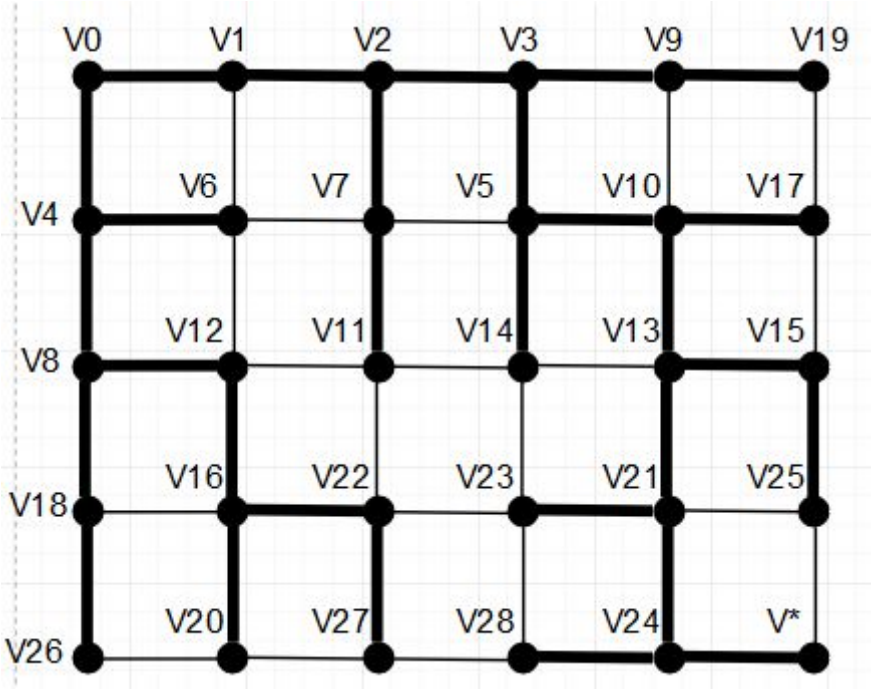
```
Enter start vertex:0

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.