МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра систем штучного інтелекту

Лабораторна робота №5

З курсу "Дискретна математика "

Виконав:
ст.гр. КН-110
Бохонко Андрій
Викладач:
Мельникова Н.І.

Львів – 2018

Лабораторна робота № 5.

В.3

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

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

Теоретичні відомості :

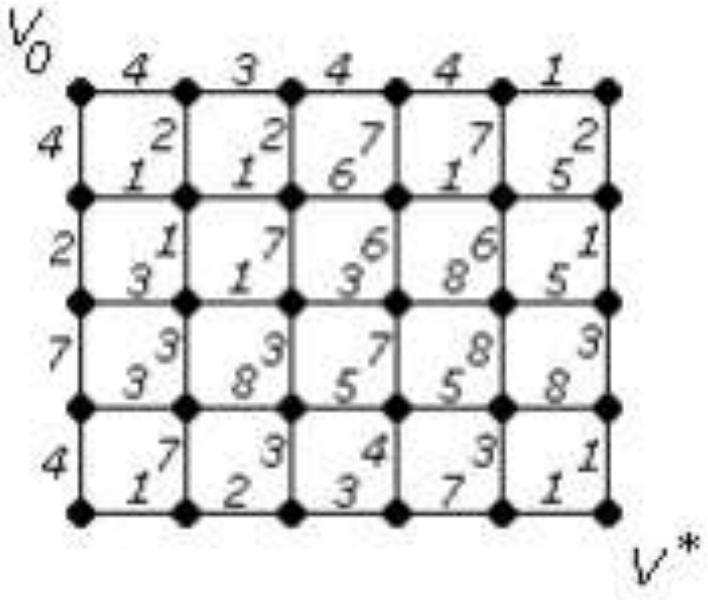
Задача знаходження найкоротшого шляху з одним джерелом полягає у знаходженні найкоротших(мається на увазі найоптимальніших за вагою) шляхів від деякої вершини(джерела) до всіх вершин графа G . Для розв'язку цієї задачі використовується «жадібний» алгоритм, який називається алгоритмом Дейкстри. «Жадібними»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

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. V9 V4 V2 V1 V3 V6 V5 V10 V12 V11 V7 V8 V14 V13 V15 V16 V17 V18 V0 V * Алгоритм γ укладання графа G являє собою процес послідовного приєднання до деякого укладеного підграфа G ~ графа G нового ланцюга, обидва кінці якого належать G ~ . При цьому в якості початкового плоского графа 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.

1. За допомогою алгоритму Дейкстра знайти найкоротший шлях у графі поміж парою вершин V0 і V * .

3



Розставляємо вершини за найменшими шляхом до них

$l(V_0) = 0\,, l(V_1) = 4, l(V_2) = 4, l(V_3) = 5, l(V_4) = 6, l(V_5) = 6, l(V_6) = 6,$

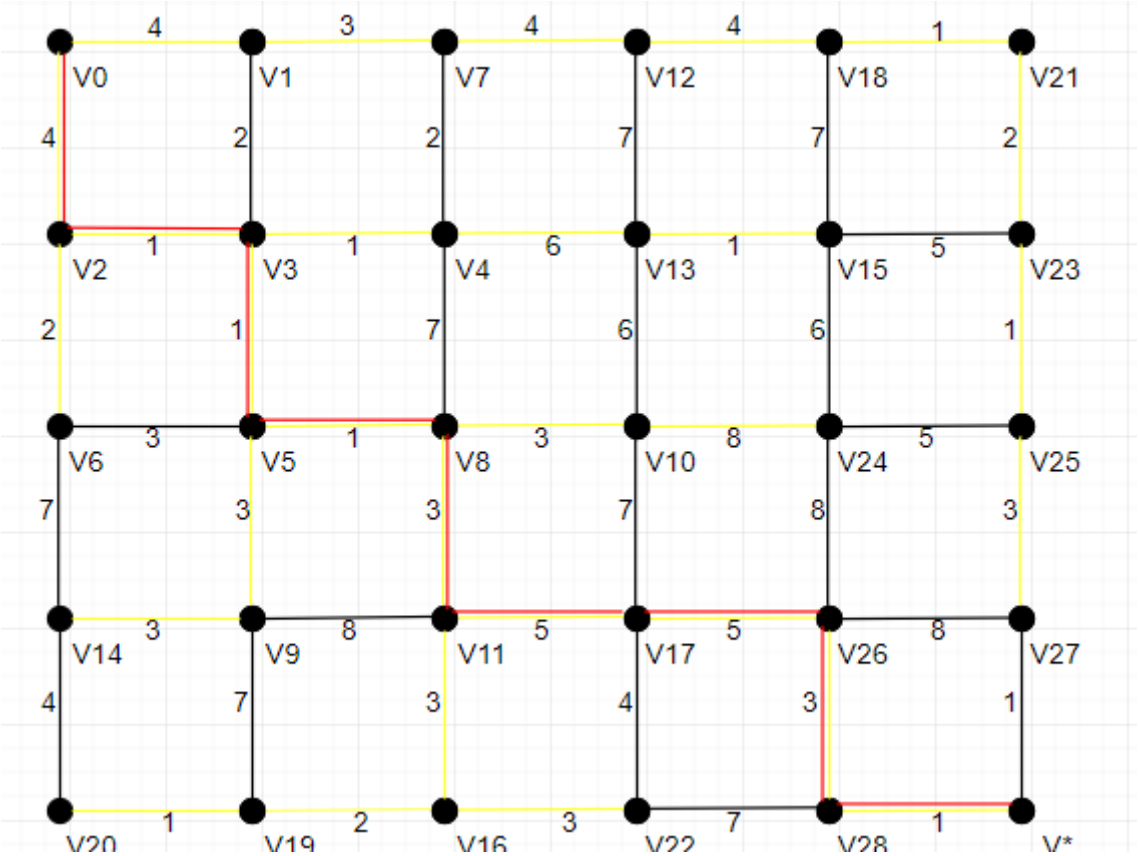
$l(V_7) = 7, l(V_8) = 7, l(V_9) = 9, l(V_{10}) = 10\,, l(V_{11}) = 10,$

$l(V_{12}) = 11, l(V_{13}) = 12, l(V_{14}) = 12, l(V_{15}) = 13,$

$l(V_{16}) = 13\,, l(V_{17}) = 15, l(V_{18}) = 15, l(V_{19}) = 15, l(V_{20}) = 16,$

$l(V_{21}) = 16, l(V_{22}) = 16, l(V_{23}) = 18, l(V_{24}) = 18, l(V_{24}) = 18, l(V_{25}) = 19$

$l(V_{26}) = 20, l(V_{27}) = 22, l(V_{28}) = 23, l(V^*) = 24$

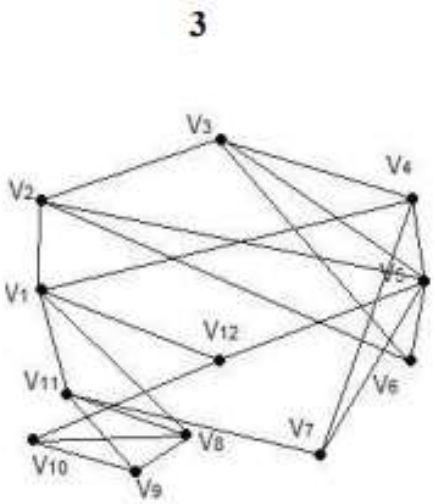


Жовтим виділено відстані до найближчих вершин. Червоним виділено найменшу відстань до V^*. Це є кістяковим дерево, яке містить усі вершини.

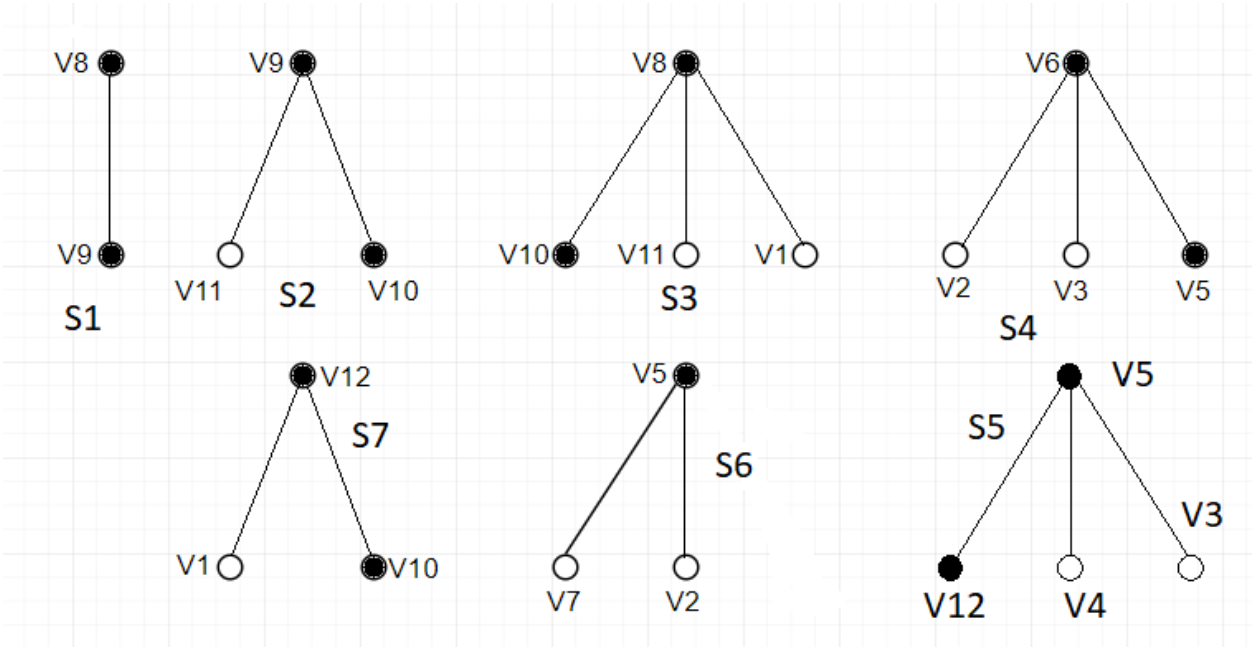
Найкоротший ланцюг $[V_0, V_2, V_3, V_5, V_8, V_{11}, V_{17}, V_{26}, V_{28}, V^*]$

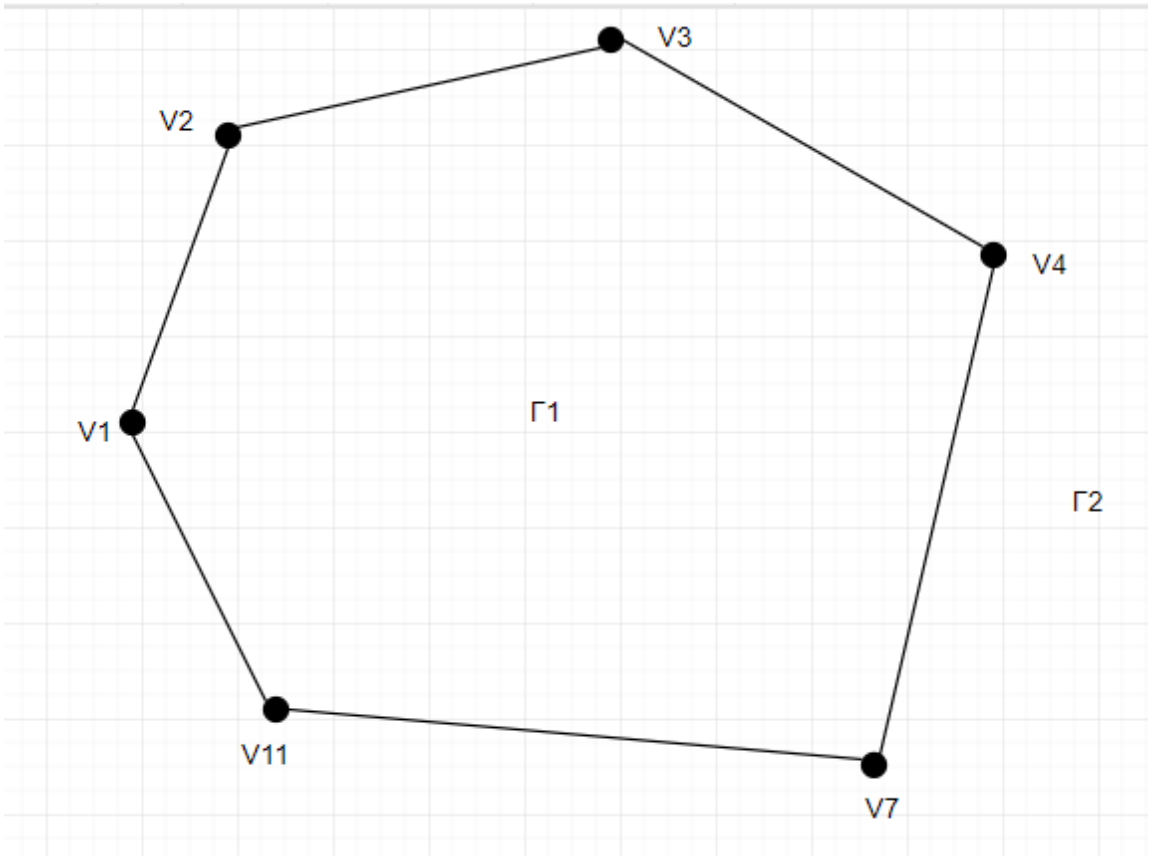
Довжина ланцюга $l = l(V^*) = 24$

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

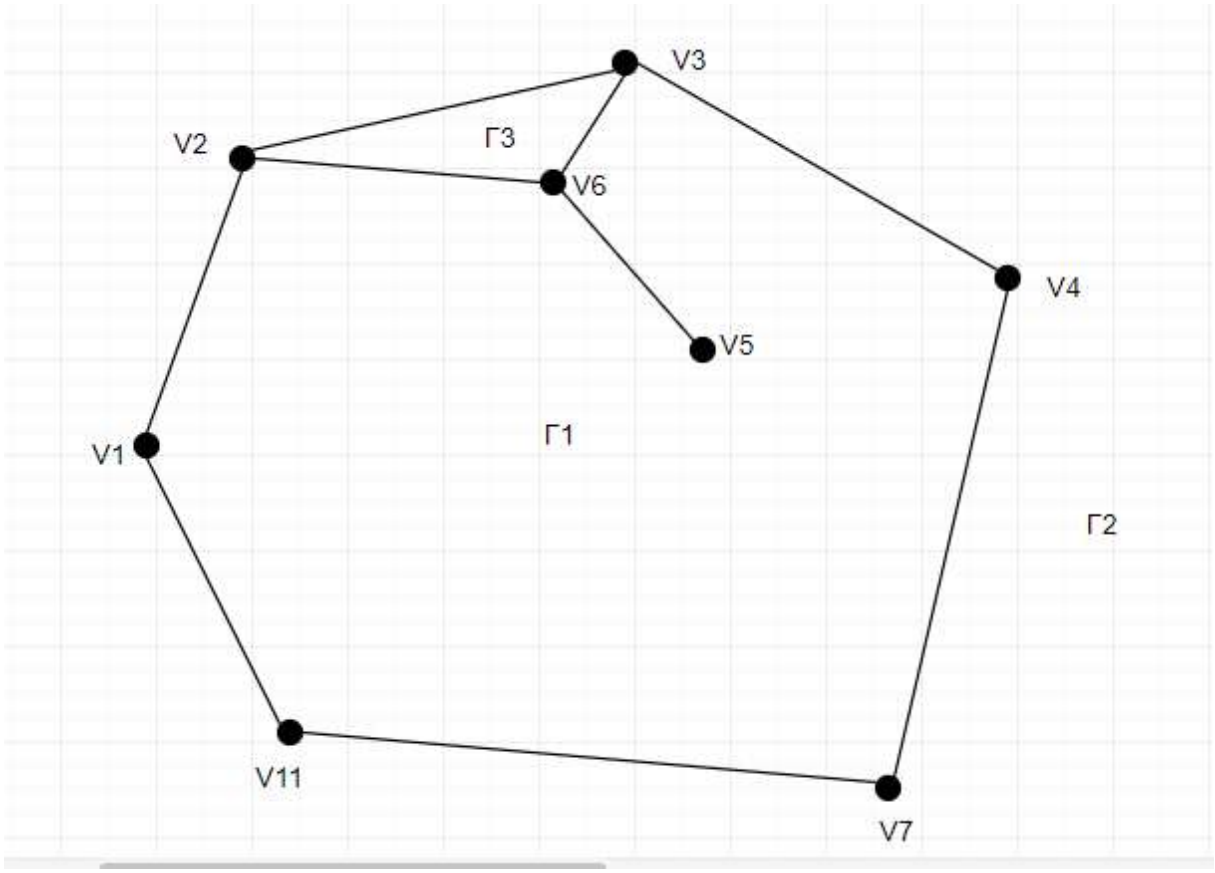


Спочатку укладемо цикл [1, 2, 3, 4, 7, 11]. Цей цикл розіб'є площину на дві грані Г1 і Г2. Запишемо сегменти S1, S2, S3, S4, S5, S6 і S7.

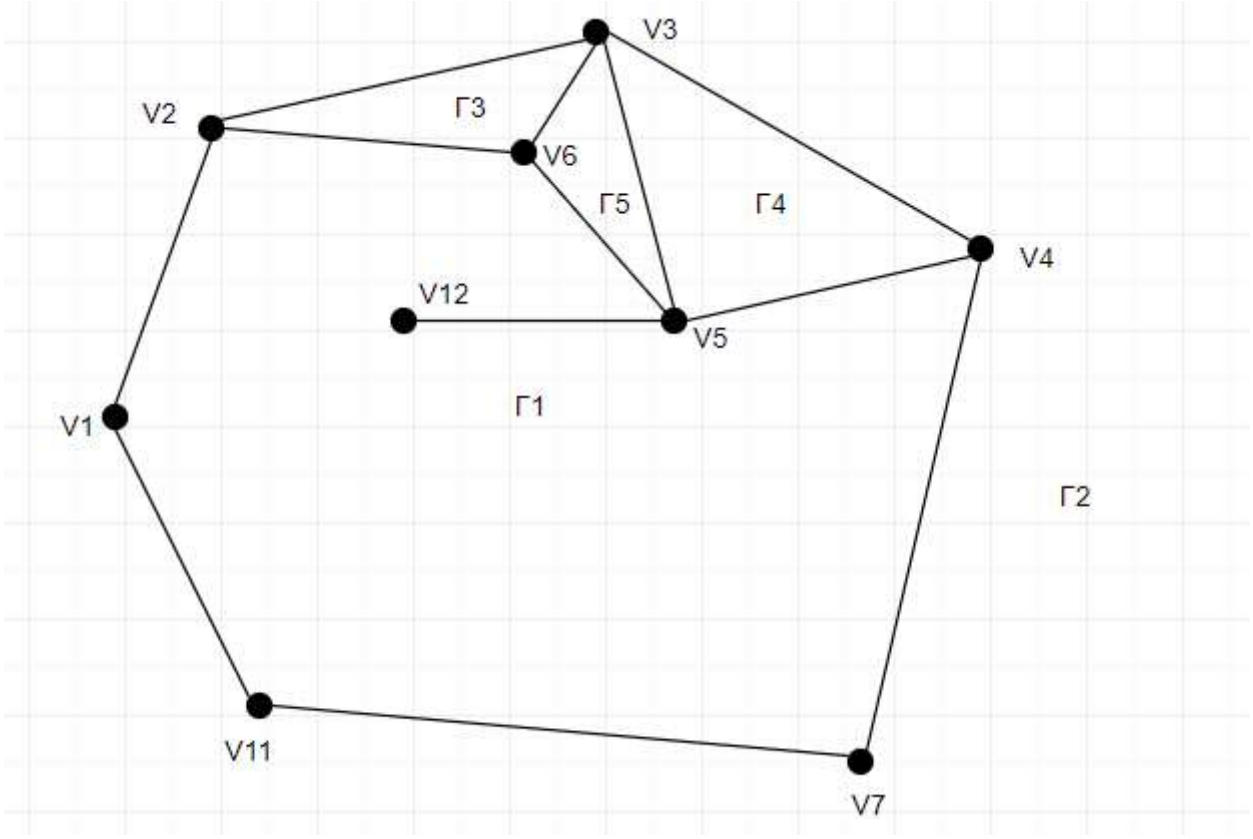




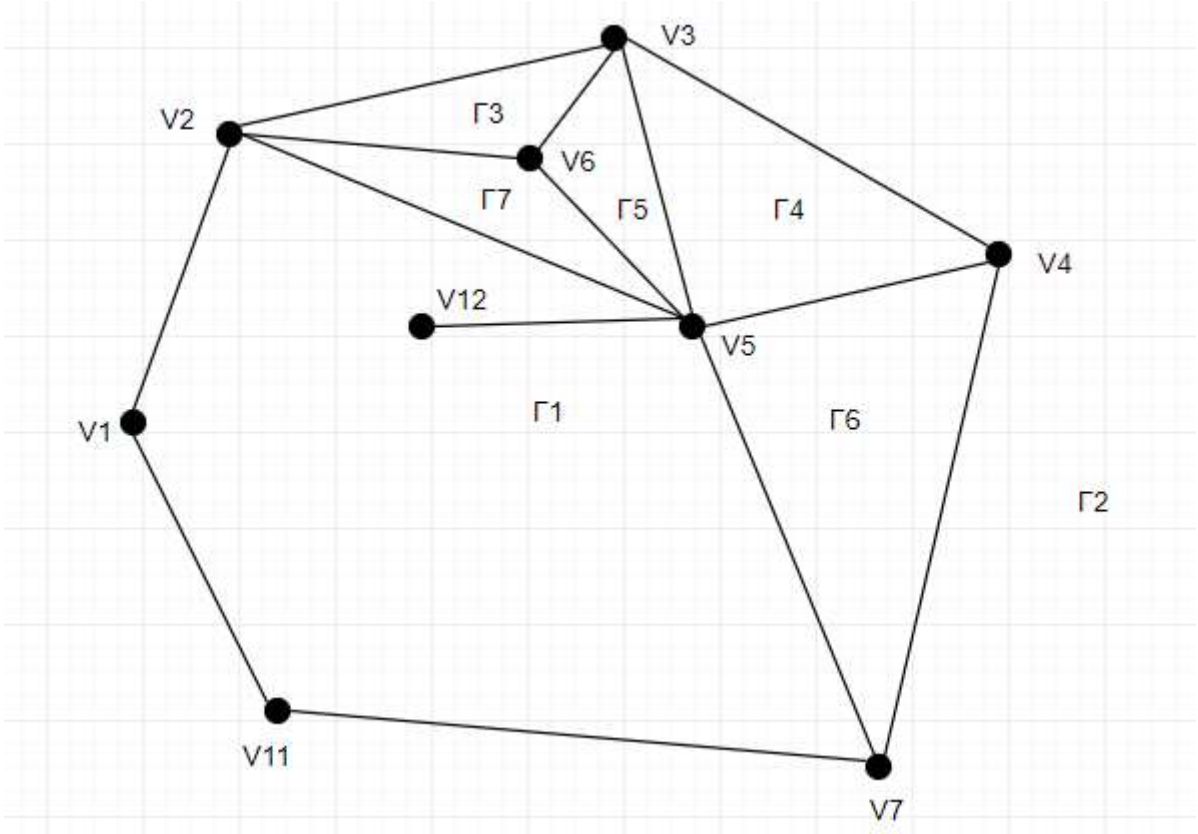
Укладемо сегмент S4. Утворить грань Г3.



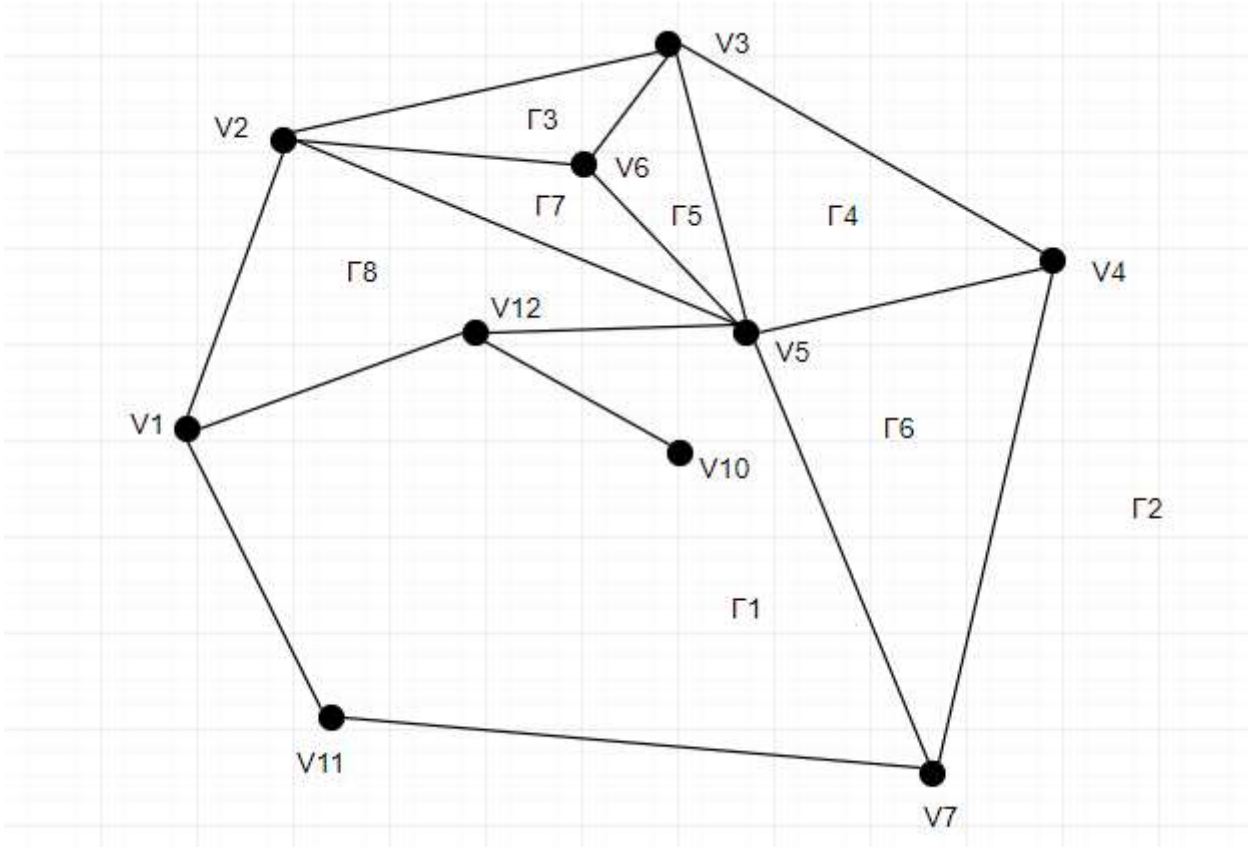
Укладемо сегмент S5. В результаті утвориться грань Г4 і Г5.



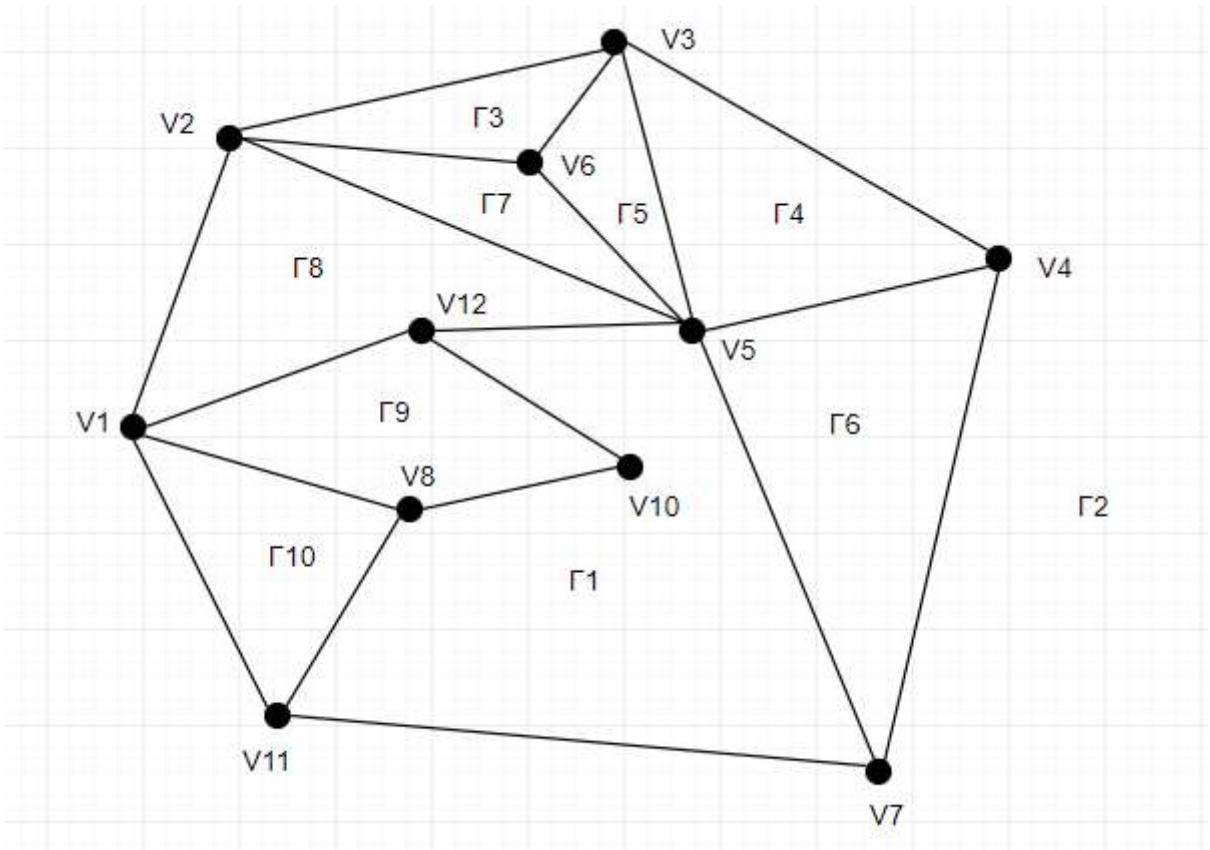
Тоді укладемо сегмент S6. Утворилась грань Г6 і Г7.



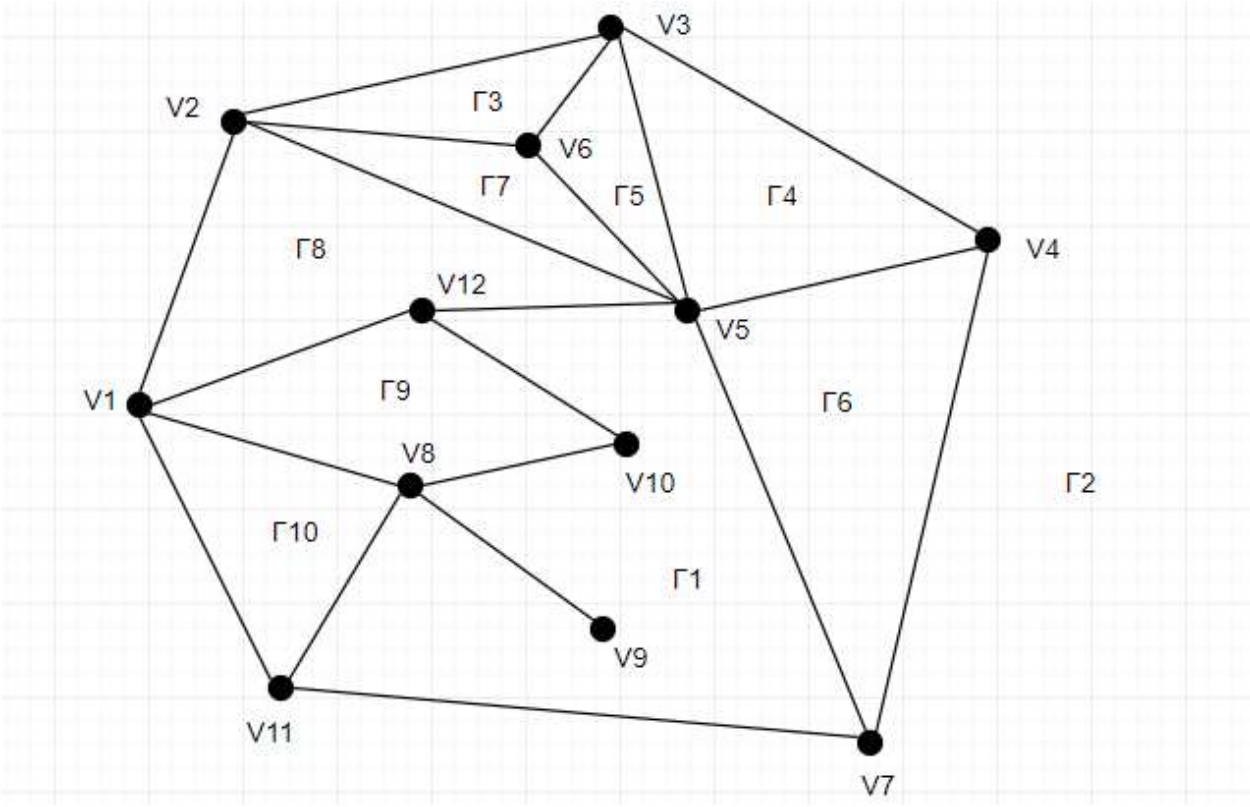
Укладаємо сегмент S7. Утворилась грань Г8.



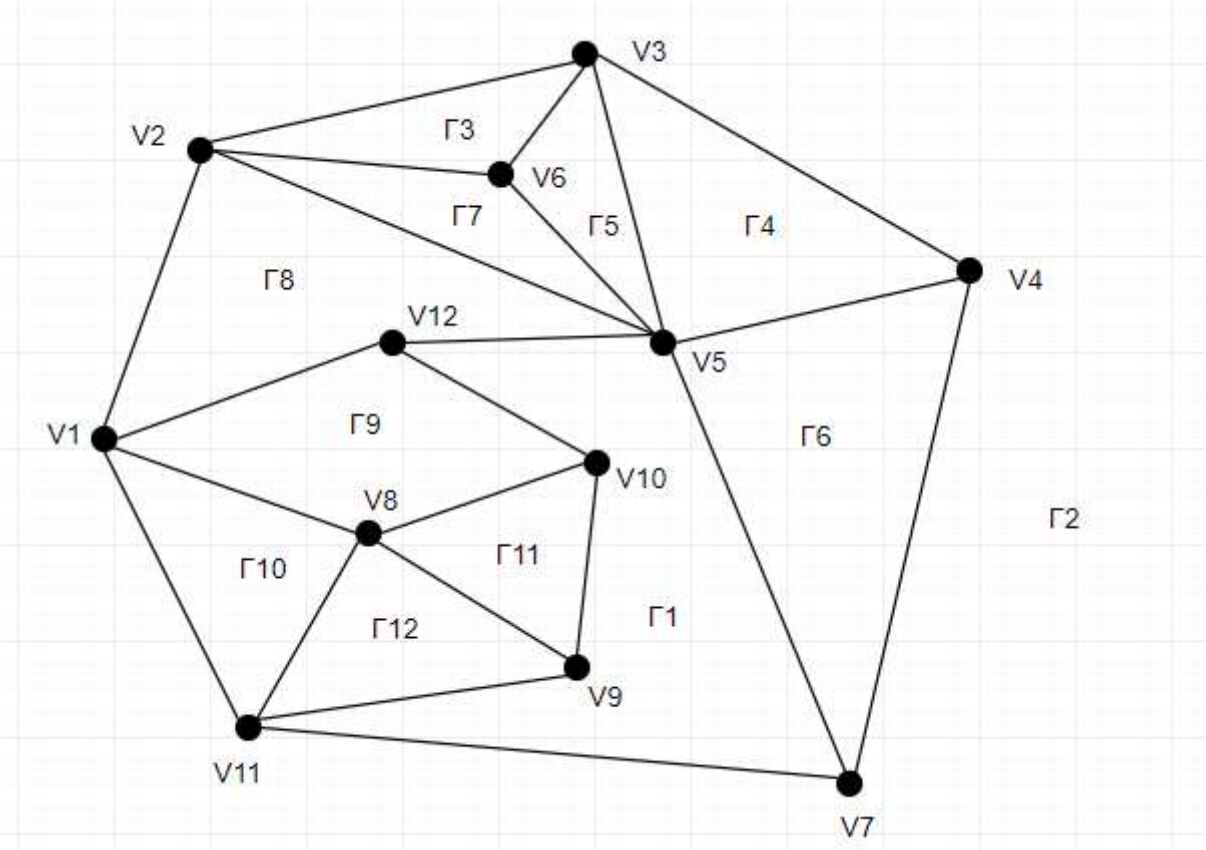
Тоді укладаємо сегмент S3. З'явилась грань Г9 і Г10.



Укладаємо сегмент S1. Нових граней не утворилось.



Укладаємо останній сегмент S2. Утворилась грань Г11 і Г12.



Отже, граф укладений на площині.

Програма

```
1.  #include<stdio.h>


2.  #define INFINITY 9999
3.  #define MAX 30


4.  void dejkstra(int G[MAX][MAX], int n, int startnode);


5.  int main()
6.  {
7.  int G[MAX][MAX], i, j, n, u;
8.  printf("Amount of vertex");
9.  scanf("%d", &n);
10. //initializating adjancency matrix to 0
11. for (i = 0; i < n; i++)
12. for (j = 0; j < n; j++)
13. G[i][j] = 0;


14. //entering weights of each edge
15. //1st vertex - 2nd vertex - weight
16. printf("Input your graph (if you finished tap 999):\n");
17. int a,b,w;
18. while(1)
19. {
20. printf("FIRST VERTEX: ");
21. scanf("%d", &a);
22. while (((a < 0) || (a > n - 1)) && (a != 999))
23. {
24. printf("TRY AGAIN: ");
25. scanf ("%d", &a);
26. }
27. if (a == 999)
28. break;
29. printf("SECOND VERTEX: ");
30. scanf("%d", &b);
31. while (((b < 0) || (b > n - 1)) && (b != 999))
32. {
33. printf("TRY AGAIN: ");
34. scanf("%d", &b);
35. }
36. if (b == 999)
37. break;
38. printf("WEIGHT: ");
39. scanf("%d", &w);
40. while (w <= 0)
41. {
42. printf("TRY AGAIN: ");
43. scanf("%d", &w);
44. }
45. if (w == 999)
46. break;
47. if ((a != 999) && (b != 999) && (w != 999))
```

```
{
G[a][b] = w;
G[b][a] = w;
}
}

printf("Adjacency matrix:\n");
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
printf("%d ", G[i][j]);
printf("\nEnter the starting point:");
scanf("%d", &u);
dejkstra(G, n, u);
printf("\n");
return 0;
}

void dejkstra(int G[MAX][MAX], int n, int startnode)
{

int cost[MAX][MAX], distance[MAX], pred[MAX];
int visited[MAX], count, mindistance, nextnode, i, j;

//pred[] stores the predecessor of each node
//count gives the number of nodes that we already seen

//create the cost matrix
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
if (G[i][j] == 0)
cost[i][j] = INFINITY;
else
cost[i][j] = G[i][j];

//initialize pred[],distance[] and visited[]
for(i = 0; i < n; i++)
{
distance[i] = cost[startnode][i];
pred[i] = startnode;
visited[i] = 0;
}

distance[startnode] = 0;
visited[startnode] = 1;
count = 1;

while (count < n - 1)
{
```

```
88.  mindistance = INFINITY;


89.  //nextnode gives the node at minimum distance
90.  for (i = 0; i < n; i++)
91.  if (distance[i] < mindistance && !visited[i])
92.  {
93.  mindistance = distance[i];
94.  nextnode = i;
95.  }


96.  //check if a better path exists through nextnode
97.  visited[nextnode] = 1;
98.  for (i = 0; i < n; i++)
99.  if (!visited[i])
        a.  if (mindistance + cost[nextnode][i] < distance[i])
        b.  {
        c.  distance[i] = mindistance + cost[nextnode][i];
        d.  pred[i] = nextnode;
        e.  }
100.        count++;
101.        }


102.        //print the path and distance of each node
103.        for (i = 0; i < n; i++)
104.        if(i != startnode)
105.        {
106.        printf("\nLENGHT OF ROUTE%d=%d", i, distance[i]);
107.        printf("\nMINIMAL ROUTE TO VERTEX=%d", i);


108.        j = i;
109.        do
110.        {
111.        j = pred[j];
112.        printf(" -----%d",j);
113.        } while (j != startnode);
114.        }
115.        }
```

Результат

```
LENGHT OF ROUTE1=3
MINIMAL ROUTE TO VERTEX=1 -----0
LENGHT OF ROUTE2=4
MINIMAL ROUTE TO VERTEX=2 -----0
LENGHT OF ROUTE3=6
MINIMAL ROUTE TO VERTEX=3 -----2 -----0
LENGHT OF ROUTE4=6
MINIMAL ROUTE TO VERTEX=4 -----1 -----0
LENGHT OF ROUTE5=7
MINIMAL ROUTE TO VERTEX=5 -----4 -----1 -----0
LENGHT OF ROUTE6=8
MINIMAL ROUTE TO VERTEX=6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE7=8
MINIMAL ROUTE TO VERTEX=7 -----1 -----0
LENGHT OF ROUTE8=9
MINIMAL ROUTE TO VERTEX=8 -----4 -----1 -----0
LENGHT OF ROUTE9=9
MINIMAL ROUTE TO VERTEX=9 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE10=9
MINIMAL ROUTE TO VERTEX=10 -----5 -----4 -----1 -----0
LENGHT OF ROUTE11=10
MINIMAL ROUTE TO VERTEX=11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE12=10
MINIMAL ROUTE TO VERTEX=12 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE13=11
MINIMAL ROUTE TO VERTEX=13 -----10 -----5 -----4 -----1 -----0
LENGHT OF ROUTE14=11
MINIMAL ROUTE TO VERTEX=14 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE15=12
MINIMAL ROUTE TO VERTEX=15 -----14 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE16=12
MINIMAL ROUTE TO VERTEX=16 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE17=13
MINIMAL ROUTE TO VERTEX=17 -----9 -----6 -----5 -----4 -----1 -----0
```

```
LENGHT OF ROUTE13=11
MINIMAL ROUTE TO VERTEX=13 -----10 -----5 -----4 -----1 -----0
LENGHT OF ROUTE14=11
MINIMAL ROUTE TO VERTEX=14 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE15=12
MINIMAL ROUTE TO VERTEX=15 -----14 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE16=12
MINIMAL ROUTE TO VERTEX=16 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE17=13
MINIMAL ROUTE TO VERTEX=17 -----9 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE18=13
MINIMAL ROUTE TO VERTEX=18 -----15 -----14 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE19=13
MINIMAL ROUTE TO VERTEX=19 -----16 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE20=13
MINIMAL ROUTE TO VERTEX=20 -----12 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE21=14
MINIMAL ROUTE TO VERTEX=21 -----18 -----15 -----14 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE22=14
MINIMAL ROUTE TO VERTEX=22 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE23=16
MINIMAL ROUTE TO VERTEX=23 -----20 -----12 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE24=16
MINIMAL ROUTE TO VERTEX=24 -----16 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE25=18
MINIMAL ROUTE TO VERTEX=25 -----18 -----15 -----14 -----6 -----5 -----4 -----1 -----0
LENGHT OF ROUTE26=20
MINIMAL ROUTE TO VERTEX=26 -----16 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE27=20
MINIMAL ROUTE TO VERTEX=27 -----19 -----16 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE28=21
MINIMAL ROUTE TO VERTEX=28 -----24 -----16 -----11 -----8 -----4 -----1 -----0
LENGHT OF ROUTE29=15
MINIMAL ROUTE TO VERTEX=29 -----21 -----18 -----15 -----14 -----6 -----5 -----4 -----1 -----0
```

Висновок

В ході цієї лабораторної роботи я освоїв алгоритм Дейкстри. Навчився реалізовувати його програмно. Також навчився укладати граф на площині.