МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра систем штучного інтелекту

Лабораторна робота №5

З дисципліни «Дискретна математика»

Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи.

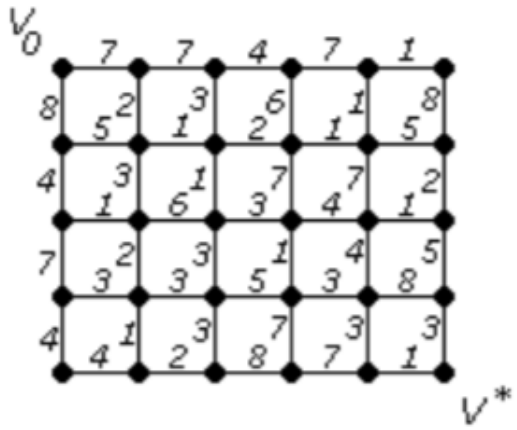
Виконав:
студент групи КН-110
Єлечко Олег Андрійович

Викладач:
Мельникова Наталія Іванівна

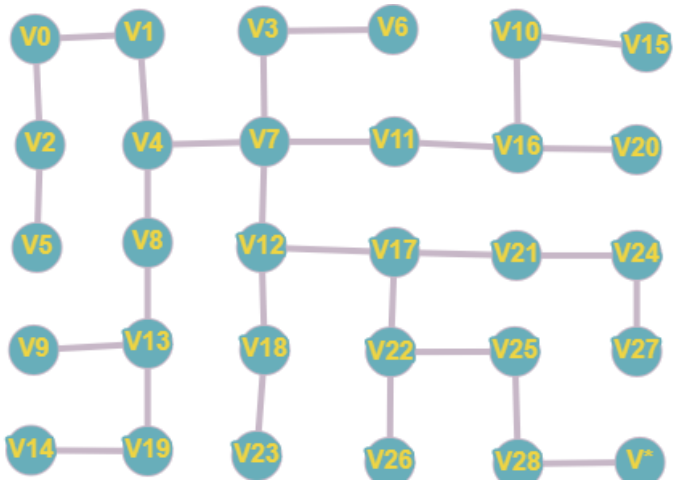
Львів – 2018р.

1. Завдання на алгоритм Дейкстри

Потрібно за допомогою алгоритму Дейкстри знайти найкоротший шлях у графі поміж парою вершин V_0 та V^*. Нехай маємо граф $G = (V, E)$, зображений нижче.



Тоді, за алгоритмом, починаючи з вершини $V_0 \in V$, вибираємо такі найближчі вершини $V_1, V_2, \ldots V_n$, щоби довжина ланцюгу до них була мінімальною. В результаті отримаємо граф G', що буде деревом. На наступному малюнку позначимо це дерево та відстані до кожної точки.

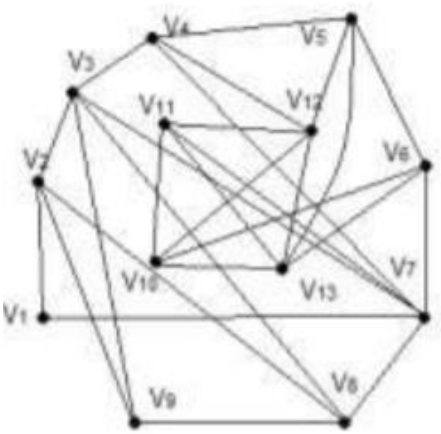


l(V_1) = 7, l(V_2) = 8, l(V_3) =13, l(V_4) = 9,
l(V_5) = 12, l(V_6) = 17, l(V_7) =10, l(V_8)=12,
l(V_9) = 17, l(V_{10}) = 14, l(V_{11}) = 12,
l(V_{12}) = 11, l(V_{13}) = 14, l(V_{14}) = 19,
l(V_{15}) = 15, l(V_{16}) = 13, l(V_{17}) = 14,
l(V_{18}) = 14, …l(V_{29}) = 22.

Тепер, здійснюючи зворотній прохід по дереву від точки V^* до V_0, можна прокласти такий маршрут: $V_0 \rightarrow V_1$
$\rightarrow V_4 \rightarrow V_7 \rightarrow V_{12} \rightarrow V_{17} \rightarrow V_{22} \rightarrow V_{25} \rightarrow V_{28} \rightarrow V^*$.

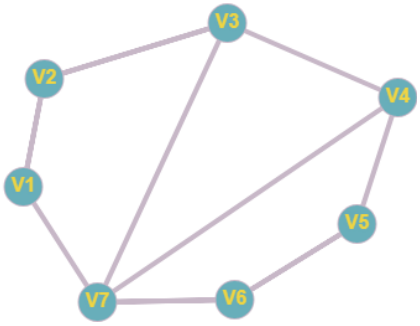
Його ціна рівна 22.

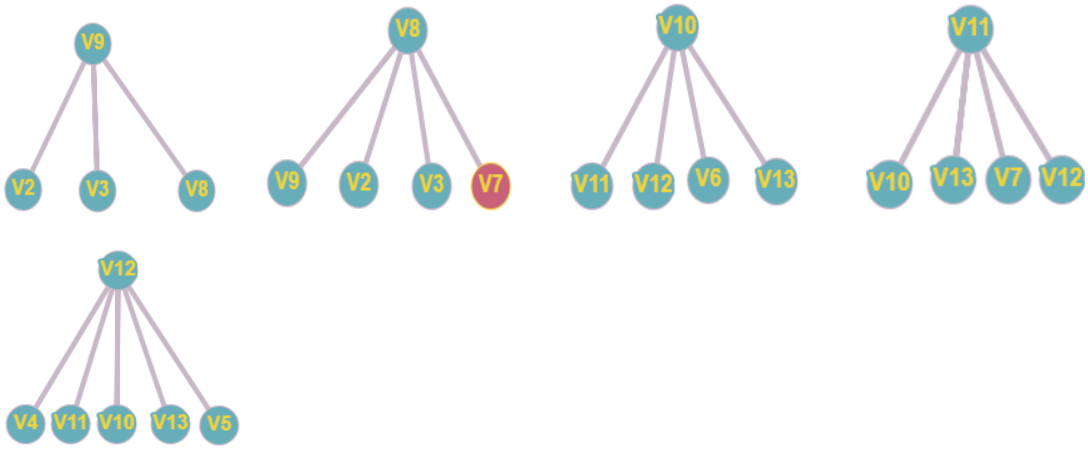
2. Завдання на γ-укладку



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

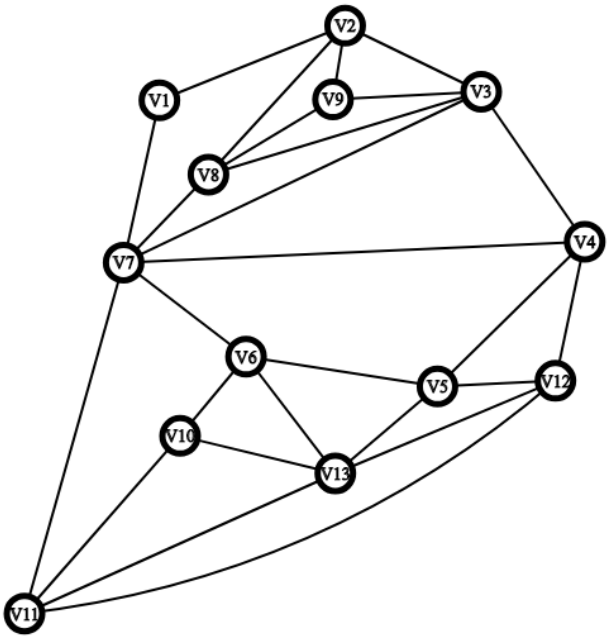
Розб'ємо даний граф на один простий цикл і інші його частини але на площині. Нам потрібно перевірити чи під час укладки графи не перетинаються, бо в разі якщо вони перетинаються даний граф буде просторовий.





Після того як ми розбили граф на елементарний цикл та решту вершин.

Спробуємо їх поєднати і перевірити чи даний граф можна укласти на площині.



Залишилось з’єднати тільки вершини V_{10} та V_{12} але це зробити неможливо без перетину одного з ребер. Отже даний граф не можливо побудувати на площині тобто він просторовий.

3. Комп’ютерна програма

Потрібно написати програму для тестування алгоритму Дейкстри на заданому графі. Програма повинна виводити найкоротший шлях від точки V_0 до V^* та основні етапи алгоритму.

Заданий граф виглядає так:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

Програмний код виглядає так:

```
 #include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 30

void shortest_path(int G[MAX][MAX],int n,int startnode);

int main()
{
    int n = 30, u = 0;

   int G[30][30]
={{0,4,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
}, //матриця суміжності
        /*2*/
{4,0,4,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*3*/
{0,4,0,3,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*4*/
{0,0,3,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*5*/
{0,4,0,1,0,3,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*6*/
{0,4,0,0,3,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*7*/
{6,4,0,0,0,0,0,4,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*8*/
{0,2,0,0,0,0,4,0,1,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*9*/
{0,0,3,0,0,0,0,1,0,1,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        /*10*/
{0,0,0,4,0,0,0,0,1,0,2,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
```

```
        /*11*/
{0,0,0,0,5,0,0,0,0,2,0,7,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0},
        /*12*/
{0,0,0,0,0,8,0,0,0,0,7,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0},
        /*13*/
{0,0,0,0,0,0,1,0,0,0,0,0,0,4,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0},
        /*14*/
{0,0,0,0,0,0,0,7,0,0,0,0,4,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0},
        /*15*/
{0,0,0,0,0,0,0,0,5,0,0,0,0,1,0,2,0,0,0,0,7,0,0,0,0,0,0,0,0},
        /*16*/
{0,0,0,0,0,0,0,0,0,7,0,0,0,0,2,0,7,0,0,0,0,1,0,0,0,0,0,0,0},
        /*17*/
{0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,7,0,7,0,0,0,0,2,0,0,0,0,0,0},
        /*18*/
{0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,7,0,0,0,0,0,0,8,0,0,0,0,0},
        /*19*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,8,0,0,0,5,0,0,0,0},
        /*20*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,8,0,3,0,0,0,2,0,0,0,0},
        /*21*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,3,0,8,0,0,0,3,0,0,0},
        /*22*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,8,0,1,0,0,0,1,0,0},
        /*23*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,1,0,5,0,0,0,3,0},
        /*24*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,5,0,0,0,0,0,7},
        /*25*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,1,0,0,0,0},
        /*26*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,1,0,3,0,0,0},
        /*27*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,3,0,3,0,0},
        /*28*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,3,0,3,0},
```

```
        /*29*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,3,0,6},
        /*30*/
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,6,0}}
;

    shortest_path(G,n,u);

    return 0;
}

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

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

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(G[i][j]==0)
                cost[i][j]=INFINITY;  //створює матрицю ваг і
якщо в матриці суміжності елемент
            else                      //дорівнює нулю то
відстань до нього безкінечна і шлях до нього не існує
                cost[i][j]=G[i][j];   //в іншому випадку
передає значення  матриці суміжності до матриці ваг



    for(i=0;i<n;i++)
    {
        distance[i]=cost[startnode][i];  // створює масив ваг
ребер і передає в нього значення з
        pred[i]=startnode;               // матриці ваг
```

```
        visited[i]=0;                    // заповнює пройдені
вершини нулями
    }
    distance[startnode]=0;   // заповнює вагу початковоï
вершини нулем
    visited[startnode]=1;    // записує що початкову вершину
пройдено
    count=1;                 // індикатор пройдених точок


    while(count<n-1)
    {
        min_distance=INFINITY;


        for(i=0;i<n;i++)
            if(distance[i]<min_distance&&!visited[i])
//знаходження найменшого шляху і перевірка чи вершина вже
пройдена
            {                                          // щоб
не утворити цикл
                min_distance=distance[i];
                nextnode=i;                            //
зберігає місце знаходження найменшої ваги
            }



            visited[nextnode]=1; //записує що ребро з
найменшою вагою пройдено


            for(i=0;i<n;i++)           // додаткова перевірка
не пройдених ребер і перевірка чи дійсно в минулій перевірці
                if(!visited[i])        // було знайдено
найменше ребро
```

```
if(min_distance+cost[nextnode][i]<distance[i]) //записує мінімальний шлях від початкової точки до заданої
                    {

distance[i]=min_distance+cost[nextnode][i];
                        pred[i]=nextnode;
//зберігає в масив вершину мінімальний шлях до якої було знайдено
                    }
        count++;
    }


//вивід інформації про мінімальні шляхи до вершин і як саме пройдено шлях
    for(i=0;i<n;i++)
        if(i!=startnode) // пропуск початкової вершини
        {
            printf("\nDistance of node[V%d] = %d",i,distance[i]);
            printf("\nPath = [V%d]",i);

            j=i;
            do
            {
                j=pred[j];
                printf("<-[V%d]",j);
            }while(j!=startnode);
    }
}
```

Вивід програми виводить таці результати:

```
Distance of node[V1] = 4
Path = [V1]<-[V0]
Distance of node[V2] = 8
Path = [V2]<-[V1]<-[V0]
Distance of node[V3] = 11
Path = [V3]<-[V2]<-[V1]<-[V0]
Distance of node[V4] = 12
Path = [V4]<-[V3]<-[V2]<-[V1]<-[V0]
Distance of node[V5] = 15
Path = [V5]<-[V4]<-[V3]<-[V2]<-[V1]<-[V0]
Distance of node[V6] = 6
Path = [V6]<-[V0]
Distance of node[V7] = 6
Path = [V7]<-[V1]<-[V0]
Distance of node[V8] = 7
Path = [V8]<-[V7]<-[V1]<-[V0]
Distance of node[V9] = 8
Path = [V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V10] = 10
Path = [V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V11] = 17
Path = [V11]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V12] = 7
Path = [V12]<-[V6]<-[V0]
Distance of node[V13] = 11
Path = [V13]<-[V12]<-[V6]<-[V0]
Distance of node[V14] = 12
Path = [V14]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V15] = 14
Path = [V15]<-[V14]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V16] = 11
Path = [V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V17] = 18
Path = [V17]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V18] = 14
Path = [V18]<-[V12]<-[V6]<-[V0]
Distance of node[V19] = 15
Path = [V19]<-[V13]<-[V12]<-[V6]<-[V0]
Distance of node[V20] = 18
Path = [V20]<-[V19]<-[V13]<-[V12]<-[V6]<-[V0]
Distance of node[V21] = 14
Path = [V21]<-[V22]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V22] = 13
Path = [V22]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V23] = 18
Path = [V23]<-[V22]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V24] = 18
Path = [V24]<-[V25]<-[V19]<-[V13]<-[V12]<-[V6]<-[V0]
Distance of node[V25] = 17
Path = [V25]<-[V19]<-[V13]<-[V12]<-[V6]<-[V0]
Distance of node[V26] = 18
Path = [V26]<-[V27]<-[V21]<-[V22]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V27] = 15
Path = [V27]<-[V21]<-[V22]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V28] = 16
Path = [V28]<-[V22]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Distance of node[V29] = 22
Path = [V29]<-[V28]<-[V22]<-[V16]<-[V10]<-[V9]<-[V8]<-[V7]<-[V1]<-[V0]
Process returned 0 (0x0)   execution time : 0.088 s
```