МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА“

Кафедра систем штучного інтелекту

Лабораторна робота №5

З дисципліни

«Дискретна математика»

Виконала:

Студентка групи КН-115

Рокицька Анастасія

Викладач:

Мельникова Н.І.

Львів – 2019р.

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

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

Короткі 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.

Задача про найкоротший ланцюг. Алгоритм Дейкстри.

Дано n-вершинний граф $G=(V,E)$, у якому виділено пару вершин $v_0, v^* \in V$, і кожне ребро зважене числом $w(e)\geq 0$. Нехай $X=\{x\}$ – множина усіх простих ланцюгів, що з'єднують v_0 з v^*, $x=(V_x, E_x)$. Цільова функція $F(x)=\sum_{e\in E_x} w(e)\to \min$. Потрібно знайти найкоротший ланцюг, тобто $x_0 \in X:\ F(x_0)=\min_{x\in X} F(x)$

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.

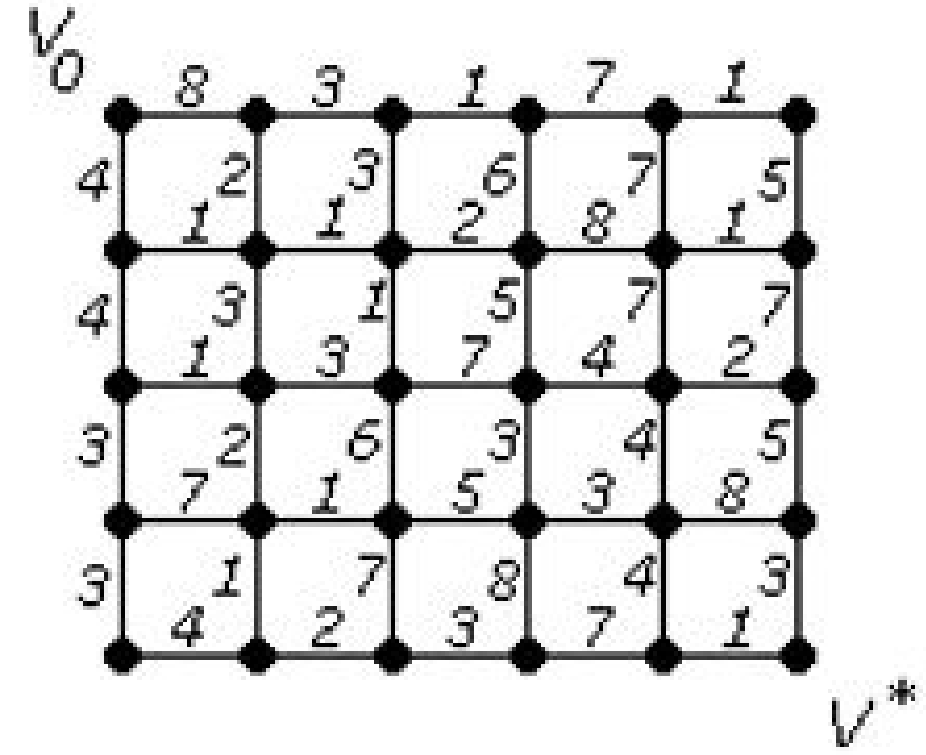
Гранню плоского графа називається максимальна по включенню множина точок площини, кожна пара яких може бути з'єднана жордановою кривою, що не перетинає ребра графа. Границею грані будемо вважати множину вершин і ребер, що належать цій грані.

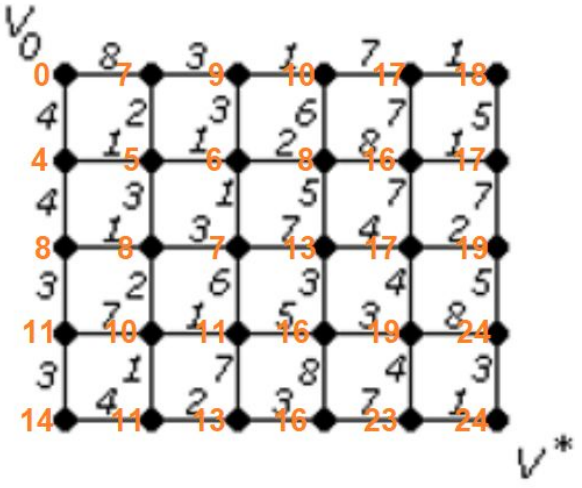
Варіант 13

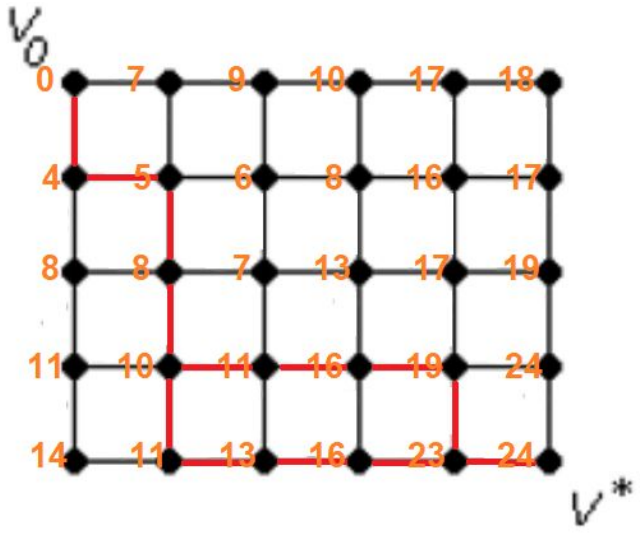
Додаток 1:

Завдання № 1. Розв'язати на графах наступні 2 задачі:

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

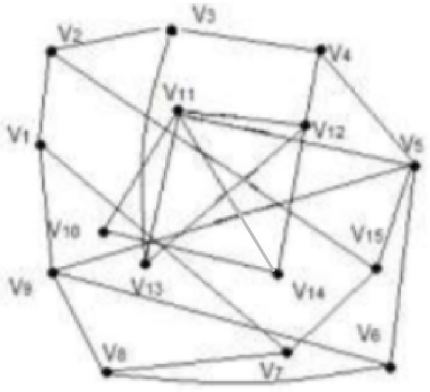


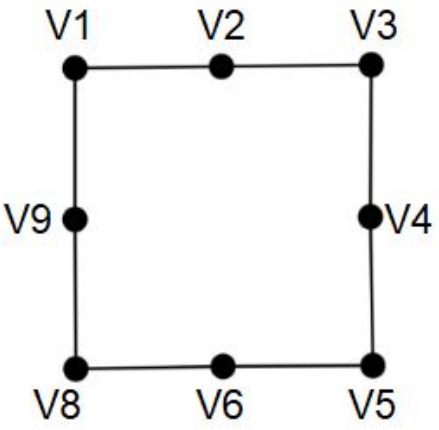




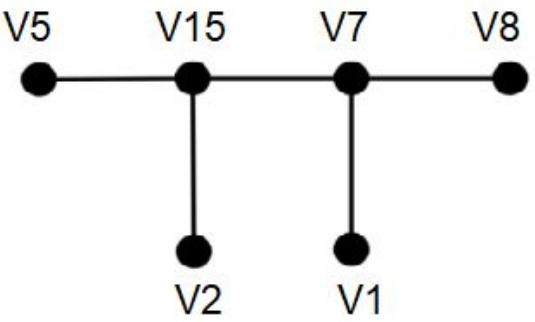
Найкоротший шлях = 24

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

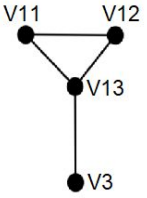




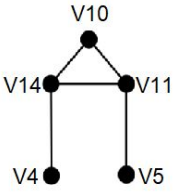
$\tilde{G}$



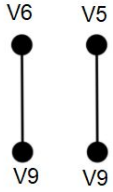
S1



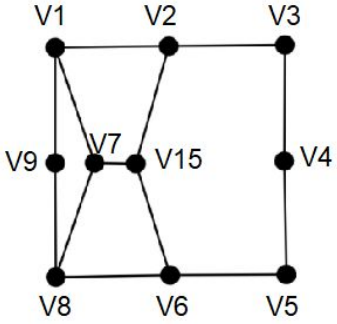
S2



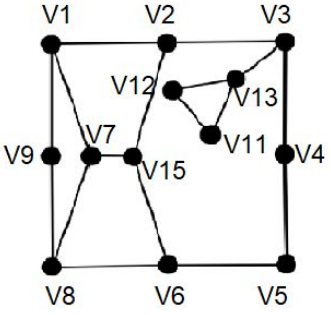
S3



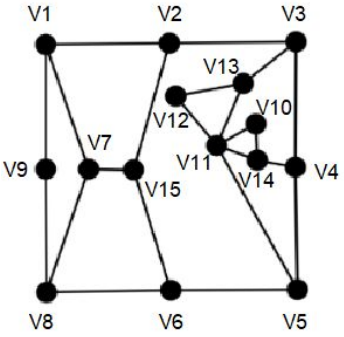
S4 **S5**



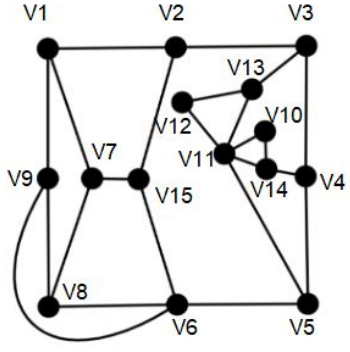
$\tilde{G} \cup S1$



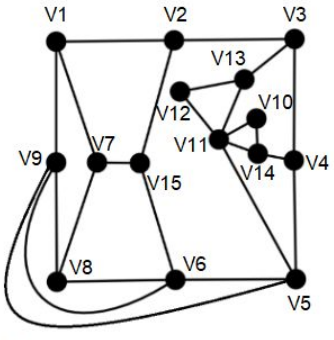
$\tilde{G} \cup S1 \cup S2$



$\tilde{G} \cup S1 \cup S2 \cup S3$

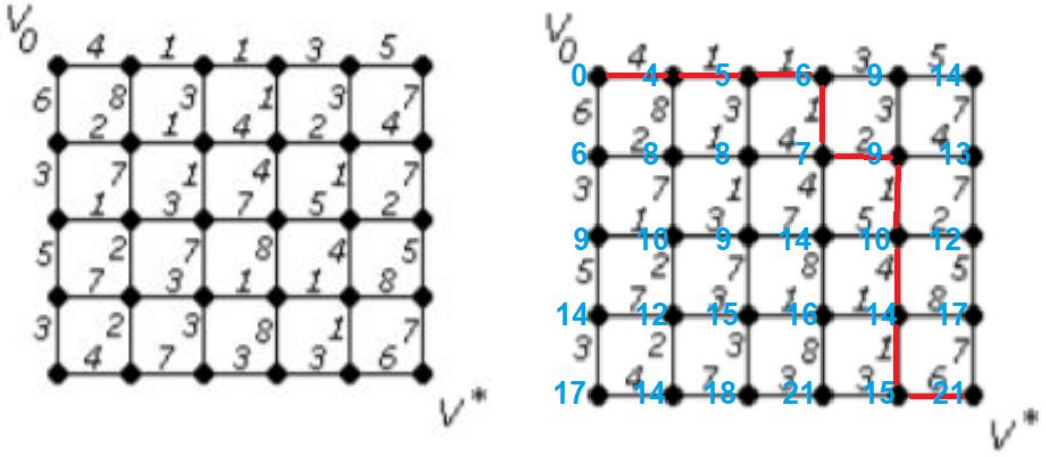


$\tilde{G} \cup S1 \cup S2 \cup S3 \cup S4$



$\tilde{G}$U S1 U S2 U S3 U S4 U S5

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<iostream>
using namespace std;
#define INF 999
int V, starting_point;
int temp;
int incidental_matrix[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},
    {4,0,1,0,0,0,0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,1,0,1,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,1,0,3,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,3,0,5,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,5,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {6,0,0,0,0,0,0,2,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,8,0,0,0,0,2,0,1,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,3,0,0,0,0,1,0,4,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,1,0,0,0,0,4,0,2,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,3,0,0,0,0,2,0,4,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0},
```

```
      {0,0,0,0,0,7,0,0,0,0,4,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,3,0,0,0,0,0,0,1,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,7,0,0,0,0,1,0,3,0,0,0,0,2,0,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,1,0,0,0,0,3,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,4,0,0,0,0,7,0,5,0,0,0,0,8,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,5,0,2,0,0,0,0,4,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,2,0,0,0,0,0,0,5,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,7,0,0,0,0,3,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,7,0,3,0,0,0,0,2,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,3,0,1,0,0,0,0,3,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,1,0,1,0,0,0,0,8,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,1,0,8,0,0,0,0,1},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,8,0,0,0,0,0,7},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,4,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,4,0,7,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,7,0,3,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,3,0,3},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,3,0,6},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,6,0}
};


int shortest_dist[30];
bool visited[30] = {0};
int parent[30];
int getNearest() {
  int minValue = 999, minNode = 0;
  for (int i = 0; i < V; i++) {
    if (!visited[i] && shortest_dist[i] < minValue) {
      minValue = shortest_dist[i];
      minNode = i;
    }
  }
```

```
  return minNode;


}


int main(void) {
  V = 30;
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      if (incidental_matrix[i][j] == 0 && i != j) {
        incidental_matrix[i][j] = 999;
      }
    }
  }
  starting_point = 0;
  for (int i = 0; i < V; i++) {
    parent[i] = i;
    shortest_dist[i] = INF;
  }
  shortest_dist[starting_point] = 0;


  for (int i = 0; i < V; i++) {
    int nearest = getNearest();
    visited[nearest] = true;
    for (int adj = 0; adj < V; adj++) {
      if (incidental_matrix[nearest][adj] != INF && shortest_dist[adj] > shortest_dist[nearest] +
incidental_matrix[nearest][adj]) {
        shortest_dist[adj] = shortest_dist[nearest] + incidental_matrix[nearest][adj];
        parent[adj] = nearest;
      }
    }
  }
```

```
  cout << "Cost : \t\t\tPath" << endl;
  cout << shortest_dist[V - 1] << "\t\t\t" << " ";
  cout << V << " ";
  temp = parent[V - 1];
  while (temp != starting_point) {
     if (temp == 1) {
        cout << " <- " << temp + 1 << " <- 1";

     } else {
        cout << " <- " << temp + 1 << " ";
     }
     temp = parent[temp];
  }
  cout << endl;

  return 0;
}
```

Результат програми:

```
Cost :          Path
21          30  <- 29  <- 23  <- 17  <- 11  <- 10  <- 4  <- 3  <- 2 <- 1
```

Висновок: набула практичних вмінь та навичок з використання алгоритму Дейкстри.