МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота №5

з дисципліни

«Дискретна математика»

Виконав:
студент групи КН-113
Зварич Адріана
Викладач: Мельникова Н.І.

Львів – 2019 р.

Тема роботи:

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.

Задача про найкоротший ланцюг. Алгоритм Дейкстри

Дано n-вершинний граф G = (V, E) , у якому виділено пару вершин v0, v* $\in$V , , і кожне ребро зважене числом w(e) $\geq$ 0 . Нехай X = {x} – множина усіх простих ланцюгів, що з'єднують v0 з v* , (Vx , Ex) . Цільова функція F(x) = $\sum$ w(e) $\rightarrow$ min . Потрібно знайти найкоротший ланцюг, тобто : x0 $\in$ X: min F(x).

Перед описом алгоритму Дейкстри подамо визначення термінів "k-а найближча вершина і "дерево найближчих вершин". Перше з цих понять визначається індуктивно так.

1-й крок індукції. Нехай зафіксовано вершину x0, E1 – множина усіх ребер e$\in$E, інцидентних v0. Серед ребер e$\in$E1 вибираємо ребро e(1) =(v0, v1), що має мінімальну вагу, тобто w(e(1))= min w(e) . Тоді v1 називаємо першою найближчою вершиною (НВ), число w(e(1)) позначаємо l(1) = l(v1) і називаємо відстанню до цієї НВ. Позначимо V1={v0, v1} – множину найближчих вершин.

2-й крок індукції. Позначимо E2 – множину усіх ребер e=(v′,v′′), e$\in$E, таких що v′$\in$V1, v′′$\in$(V \V1). Найближчим вершинам v$\in$V1 приписано відстані l(v) до кореня v0 , причому l(v0)=0. Введемо позначення: V1 – множина таких вершин v′′$\in$(V \V1), що $\exists$ ребра виду e =(v, v′′), де v$\in$V1. Для всіх ребер e$\in$E2 знаходимо таке ребро e2=(v′, v2), що величина l(v′)+w(e2) найменша. Тоді v2 називається другою найближчою вершиною, а ребра e1, e2 утворюють зростаюче дерево для виділених найближчих вершин D2 ={e1, e2}.

(s+1)-й крок індукції. Нехай у результаті s кроків виділено множину найближчих вершин Vs={v0, v1, …, vs} і відповідне їй зростаюче дерево Ds={e1, e2, …, es}... Для кожної вершини v$\in$Vs обчислена відстань l(v) від

кореня v0 до v; Vs – множина вершин v∈(V\Vs), для яких існують ребра вигляду e =(vr , v), де vr ∈Vs , v∈(V\Vs). На кроці s+1 для кожної вершини vr∈Vs обчислюємо відстань до вершини vr : L(s+1)(vr)=l(vr)+min w(vr ,v*), де min береться по всіх ребрах e=(vr , v∗), v* ∈V s , після чого знаходимо min серед величин L(s+1)(vr). Нехай цей min досягнуто для вершин vr0 і відповідної їй v*∈V s , що назвемо vs+1. Тоді вершину vs+1 називаємо (s+1)-ю НВ, одержуємо множину Vs+1 =Vs Y vs+1 і зростаюче дерево Ds+1 =Ds Y (vr0 ,vs+1). (s+1)-й крок завершується перевіркою: чи є чергова НВ vs+1 відзначеною вершиною, що повинна бути за умовою задачі зв'язано найкоротшим ланцюгом з вершиною v0 . Якщо так, то довжина шуканого ланцюга дорівнює l(vs+1)=l(vr0)+w(vr0, vs+1); при цьому шуканий ланцюг однозначно відновлюється з ребер зростаючого дерева Ds+1. У противному випадку випливає перехід до кроку s+2.

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 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.

Нехай побудоване деяке укладання підграфа G ~ графа G .

Сегментом S відносно G ~ будемо називати підграф графа G одного з наступних виглядів:

- ребро e∈ E , e = (u, v) , таке, що e∉E ~ , u,v ∈V~, G~ (V~,E~);
- зв'язний компонент графа G – G ~ , доповнений всіма ребрами графа G , інцидентними вершинам узятого компонента, і кінцями цих ребер.

Вершину v сегмента S відносно G будемо називати *контактною*, якщо v∈V~ .

Припустимою гранню для сегмента S відносно G ~ називається грань Г графа G ~ , що містить усі контактні вершини сегмента S. Через Г(S) будемо позначати множину припустимих граней для S.

Назвемо *α-ланцюгом* простий ланцюг L сегмента S, що містить дві різні контактні вершини і не містить інших контактних вершин.

Тепер формально опишемо алгоритм γ.

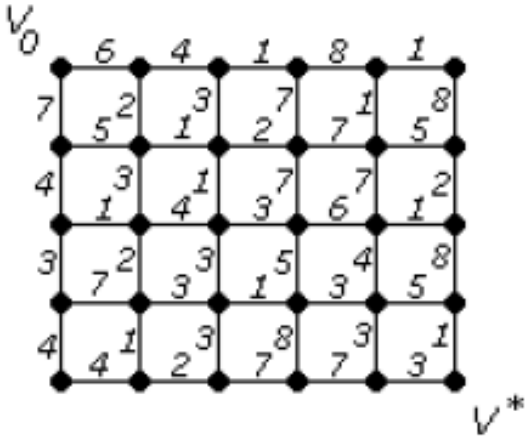
0. Виберемо деякий простий цикл C графа G і укладемо його на площині; покладемо ~G = G .
1. Знайдемо грані графа G ~ і сегменти відносно G ~ . Якщо множина сегментів порожня, то перейдемо до пункту 7.
2. Для кожного сегмента S визначимо множину Г(S).
3. Якщо існує сегмент S, для якого Г(S)=∅, то граф G не планарний. Кінець. Інакше перейдемо до п.4.
4. Якщо існує сегмент S, для якого мається єдина припустима грань Г, то перейдемо до п. 6. Інакше до п.5.
5. Для деякого сегмента S Г(S)>1. У цьому випадку вибираємо довільну припустиму грань Г.
6. Розмістимо довільний α- ланцюг L∈S у грань Г; замінимо G ~ на G~YL і перейдемо до п. 1.
7. Побудовано укладання G ~ графа G на площині. Кінець. Кроком алгоритму γ будемо вважати приєднання до G ~ α- ланцюга L.

Варіант 9

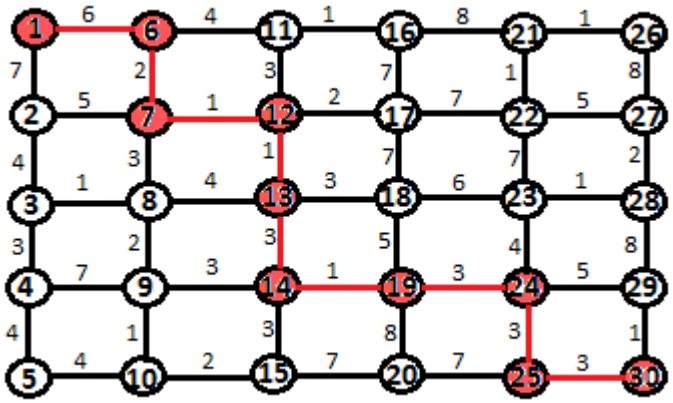
Завдання № 1. Розв'язати на графах наступні 2 задачі:

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

9



За алгоритмом починаючи з вершини V0, вибираємо такі найближчі вершини, щоб довжина ланцюга до них була мінімальною.
Результатом буде граф G:

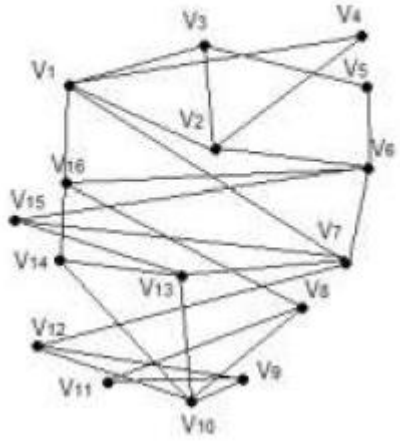


Маршрут: V1⇨V6⇨V7⇨V12⇨V13⇨V14⇨V19⇨V24⇨V25⇨V30
Його ціна рівна 23.

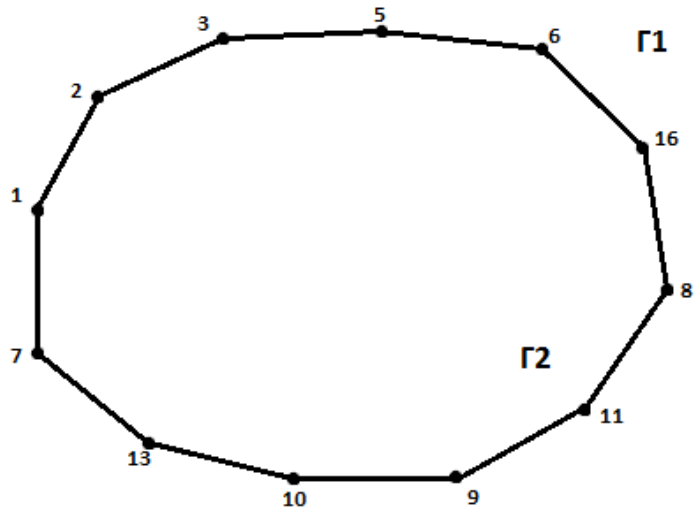
V1=0	V11=10	V21=19
V2=7	V12=9	V22=18
V3=11	V13=10	V23=19
V4=14	V14=13	V24=17
V5=18	V15=16	V25=20
V6=6	V16=11	V26=20
V7=8	V17=11	V27=22
V8=11	V18=13	V28=20
V9=13	V19=14	V29=22
V10=14	V20=22	V30=23

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

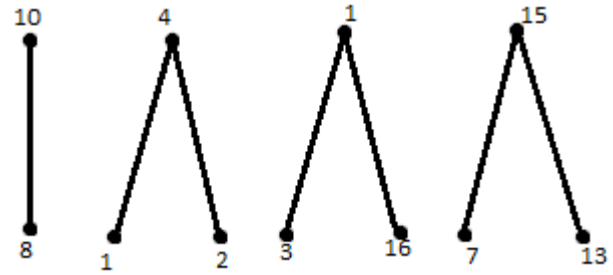
9

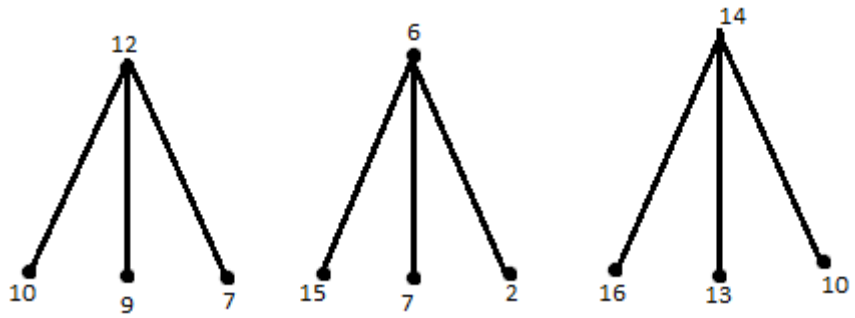


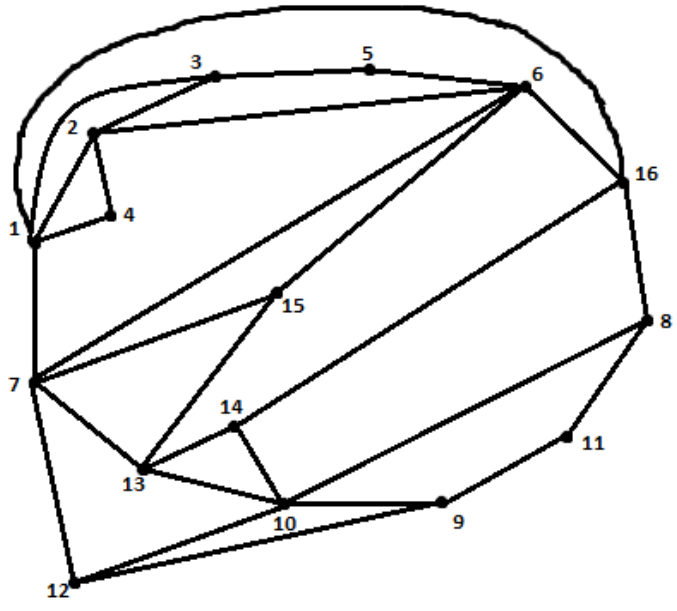
Вибираємо будь-який простий цикл в G і украдаємо його на площині. Утворюються дві грані: Г1-зовнішня, Г2-внутрішня.



Виділяємо сегменти:



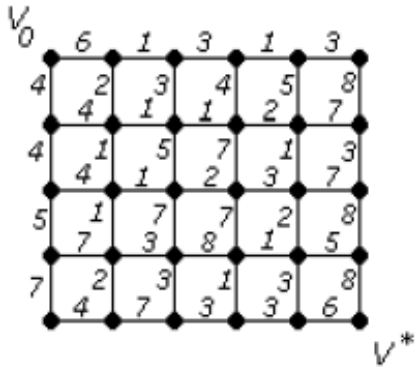




Укладка проведена успішно.

Завдання №2. Написати програму, яка реалізує алгоритм Дейкстри знаходження найкоротшого шляху між парою вершин у графі. Протестувати розроблену програму на графі згідно свого варіанту.

9



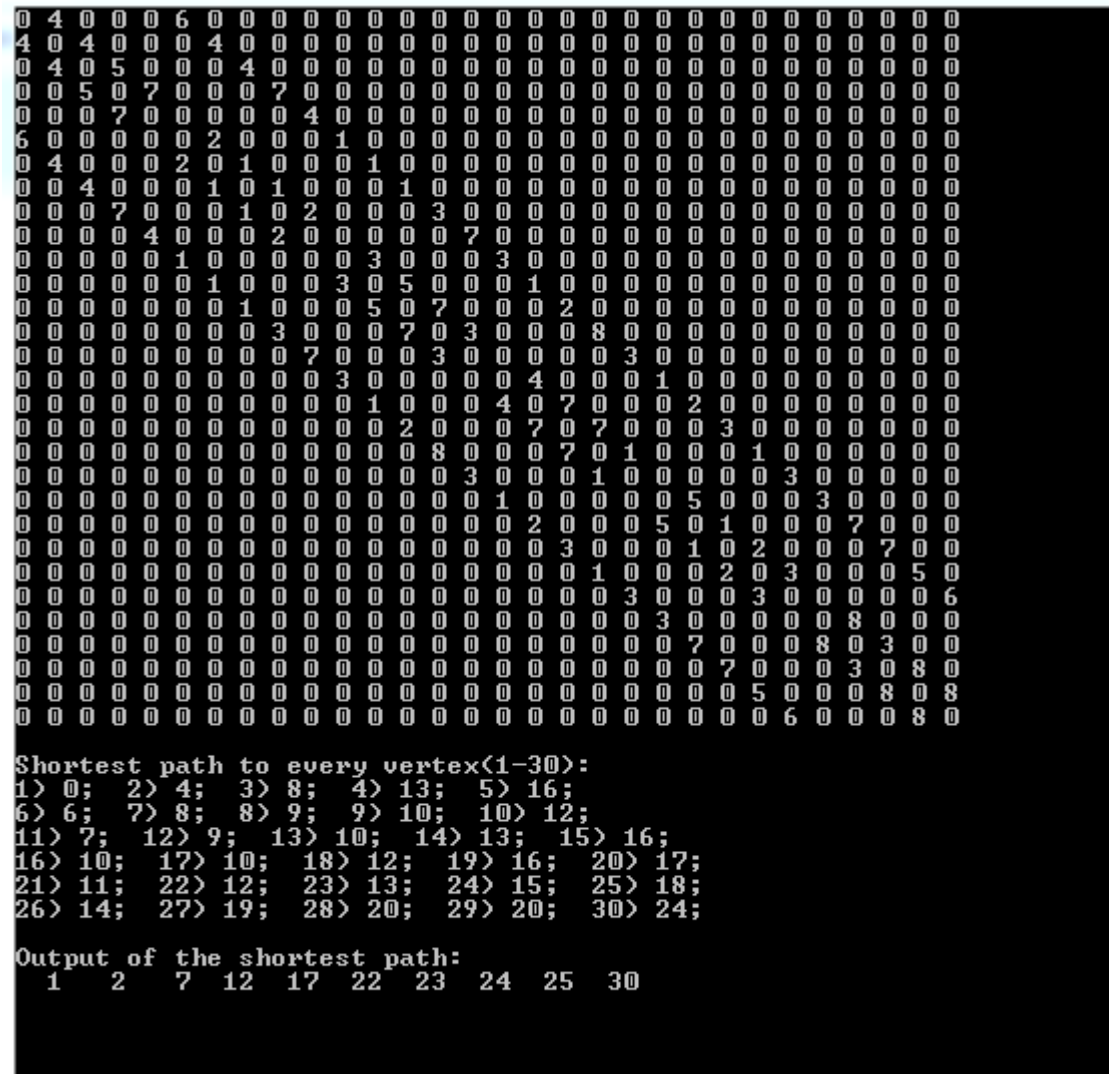
Код програми:

```
 #include <stdio.h>
#include <stdlib.h>
#define SIZE 30              // Кількість вершин, розмір матриці

int Array[SIZE][SIZE];       // Масив матриці графа
int distance[SIZE];          // Масив мінімальних відстаней
int vertex[SIZE];            // Масив вершин

// Заповнення матриці нулями
void zeroArray()
{
   for(int i=0; i<SIZE; i++)
      for(int j=0; j<SIZE; j++)
        Array[i][j]=0;
}

// Запис дуг в матрицю
void enterEdges()
{
   printf("Enter edges:\n");
   int r, c, n;
   for(int i=0; i<49; i++)
   {
      scanf("%d %d %d", &r, &c, &n);
      Array[r-1][c-1]=n;
      Array[c-1][r-1]=n;
   }
}
// Вивід матриці графа
void printArray()
{
   for(int i=0; i<SIZE; i++)
   {
      for(int j=0; j<SIZE; j++)
      {
```

```
            printf("%d ", Array[i][j]);
        }
        printf("\n");
    } }
// Ініціалізація масивів відстаней та вершин
void initArray()
{
    for(int i=0; i<SIZE; i++)
    {
        distance[i]=10000;        // Всі відстані поки що є невизначеними, тому 10000
        vertex[i]=1;              // Всі вершини є необійденими, мають значення 1
    }
    distance[0]= 0;               // Відстань до першої вершини 0
}

// Вивід найкоротших відстаней до вершин 1-30
void printDistance()
{
    printf("\nShortest path to every vertex(1-30): \n");
        for(int i=0; i<SIZE; i++)
        {
            printf("%d) %d;  ", i+1, distance[i]);
            if((i+1)%5==0 && i!=0)
                    printf("\n");
        }
}

// Головна функція
int main(void)
{
    int temp;                  // Тимчасова змінна для запису матриці графа
    int minindex;              // Змінна для ітерацій циклу основного алгоритму
    int min;

    zeroArray();               // Заповнення матриці нулями
    enterEdges();              // Ввід ребер та їхньої ваги
    printArray();              // Вивід матриці
    initArray();               // Ініціалізація масивів відстаней та вершин

    // Основний алгоритм
    do
    {
        minindex=10000;
        min=10000;
        for (int i=0; i<SIZE; i++)
        {
            if ((vertex[i]==1) && (distance[i]<min))        // Якщо вершину ще не обійшли і вага
менша за min
            {
                min=distance[i];                    // Встановлюємо змінній min мінімальне значення
                minindex=i;                         //Визначаємо позицію мінімальної відстані
            }
        }
```

```
      if(min!=10000)                    // Якщо попередня умова виконалась
      {
         for(int i=0; i<SIZE; i++)
         {
            if(Array[minindex][i]>0)
            {
               temp=min+Array[minindex][i];
               if(temp<distance[i])
               {
                  distance[i]=temp;              // Відстань записуємо в масив відстаней
               }
            }
         }
         vertex[minindex]=0;                 // Помічаємо вершину пройденою
      }
   }
   while(minindex < 10000);                  // Допоки не знайдемо всіх відстаней

   printDistance();                          // Вивід найкоротших відстаней

// Відновлення шляху
   int ver[SIZE];                            // Масив відвіданих вершин
   int end = 29;                             // Індекс кінцевої вершини 30-1=29
   ver[0] = end + 1;                   // Перший елемент -- остання вершина
   int k = 1;                                // Індекс попередньої вершини
   int weight = distance[end]; // Вага кінцевої вершини

   while (end > 0) // Поки не дійшли до початкової вершини
   {
      for(int i=0; i<SIZE; i++) // Переглядаємо всі вершини
         if (Array[end][i] != 0)   // Якщо зв'язок є
         {
            int temp = weight - Array[end][i]; // Визначаємо вагу шляху з попередньої
вершини
            if (temp == distance[i]) // Якщо вага співпала з вирахуваною
            {                        // Значить, з цієї вершини був здійснений перехід
               weight = temp;       // Зберігаємо нову вагу
               end = i;             // Зберігаємо попередню вершину
               ver[k] = i + 1;      // І записуємо її в масив
               k++;
            }
         }
   }

   // Вивід шляху (початкова вершина опинилась в кінці масиву з k елементів)
   printf("\nOutput of the shortest path:\n");
   for(int i = k-1; i>=0; i--)
      printf("%3d ", ver[i]);

   scanf("%d", &temp);
   return 0;
}
```

Результат програми:



Висновки: на цій лабораторній роботі ми набули практичних вмінь та навичок з використання алгоритму Дейкстри.