**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**Кафедра систем штучного інтелекту**

**Лабораторна робота №5**

з дисципліни «Дискретна математика»

**Виконав:**

студент групи КН-115

Сирватка Максим

**Викладач:**

Мельникова Н.І.

**Львів – 2019 р.**

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

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

**Теоретичні відомості**

**Алгоритм Дейкстри** — [алгоритм](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC" \o "Алгоритм) на [графах](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Граф (математика)), відкритий [Дейкстрою](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0_%D0%95%D0%B4%D1%81%D0%B3%D0%B5%D1%80" \o "Дейкстра Едсгер). Знаходить [найкоротший шлях](https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B9%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D1%88%D0%B8%D0%B9_%D1%88%D0%BB%D1%8F%D1%85" \o "Найкоротший шлях) від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від'ємної довжини.

Нехай **G = (V, E)** – зважений орієнтований граф, **w(vi , vj)** – вага дуги **(vi , vj)**. Почавши з вершини **a**, знаходимо віддаль від *a* до кожної із суміжних із нею вершин. Вибираємо вершину, віддаль від якої до вершини a найменша; нехай це буде вершина **v\*.** Далі знаходимо віддалі від вершини a до кожної вершини суміжної з *v\** вздовж шляху, який проходить через вершину *v\*.* Якщо для якоїсь із таких вершин ця віддаль менша від поточної, то замінюємо нею поточну віддаль. Знову вибираємо вершину, найближчу до *a* та не вибрану раніше; повторюємо процес. Описаний процес зручно виконувати за допомогою присвоювання вершинам міток. Є мітки двох типів: тимчасові та постійні. Вершини з постійними мітками групуються у множину **M**, яку називають **множиною позначених вершин**. Решта вершин має **тимчасові мітки**, і множину таких вершин позначимо як **T, T = V \ M**. Позначатимемо мітку (тимчасову чи постійну) вершини *V* як **l(V).** Значення постійної мітки *l(V)* дорівнює **довжині найкоротшого шляху від вершини *a* до вершини *V***, тимчасової – **довжині найкоротшого шляху, який проходить лише через вершини з постійними мітками.** *Фіксованою* початковою вершиною вважаємо вершину a; довжину найкоротшого шляху шукаємо до вершини z (або до всіх вершин графа). Тепер формально **опишемо алгоритм Дейкстри:**

1. Присвоювання початкових значень. Виконати l(a) = 0 та вважати цю мітку постійною. Виконати l(v) = ∞ для всіх v ≠ a й уважати ці мітки тимчасовими. Виконати x = a, M = {a}.
2. Оновлення міток. Для кожної вершини v ∈ Г(x) \ M замінити мітки: l(v) = min{ l(v), l(x) + w(x, v) }, тобто оновлювати тимчасові мітки вершин, у які з вершини x іде дуга.
3. Перетворення мітки в постійну. Серед усіх вершин із тимчасовими мітками знайти вершину з мінімальною міткою, тобто знайти вершину v\* з умови l(v\*) = min{l(v)}, v∈T, де T = V \ M.
4. Уважати мітку вершини v\* постійною й виконати M = M ∪ {v\*}; x = v\* (вершину v\* включено в множину M).
5. **а)** Для пошуку шляху від a до z: якщо x = z, то l(z) – довжина найкоротшого шляху від a до z, зупинитись; якщо a ≠ z, то перейти до кроку 2.

**б)** Для пошуку шляхів від a до всіх вершин: якщо всі вершини отримали постійні мітки (включені в множину M), то ці мітки дорівнюють довжинам найкоротших шляхів, зупинитись; якщо деякі вершини мають тимчасові мітки, то перейти до кроку 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 **Ğ** графа G нового ланцюга, обидва кінці якого належать **Ğ** . При цьому в якості початкового плоского графа **Ğ** 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 **Ğ** графа G .

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

- ребро **e ∈ E , e = (u, v)** , таке, що e **∉Ĕ; u,v** **∈ Ṽ; Ğ = (Ṽ;Ĕ)** ;

- зв'язний компонент графа G – **Ğ**, доповнений всіма ребрами графа G, інцидентними вершинам узятого компонента, і кінцями цих ребер.

Вершину v сегмента S відносно **Ğ** будемо називати **контактною**, якщо **v** **∈ Ṽ**. **Припустимою гранню** для сегмента S відносно **Ğ** називається грань Г графа **Ğ**, 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 С графа G і укладемо його на площині; покладемо **Ğ = G** .

**1.** Знайдемо грані графа **Ğ** і сегменти відносно **Ğ**. 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 **Ğ** на **ĞYL** і перейдемо до п. 1.

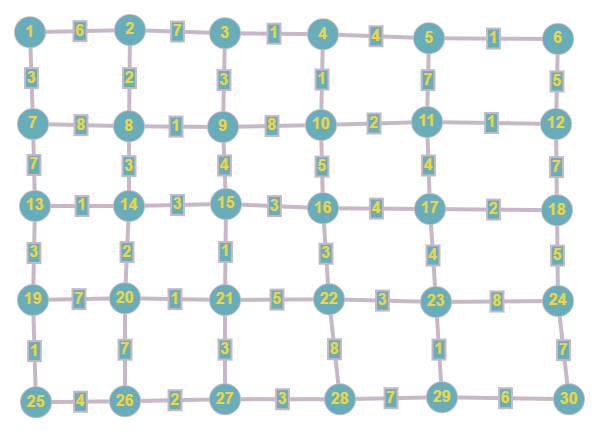
**7.** Побудовано укладання **Ğ** графа G на площині. Кінець. Кроком алгоритму **γ** будемо вважати приєднання до **Ğ** α- ланцюга L.

**Завдання лабораторної роботи**

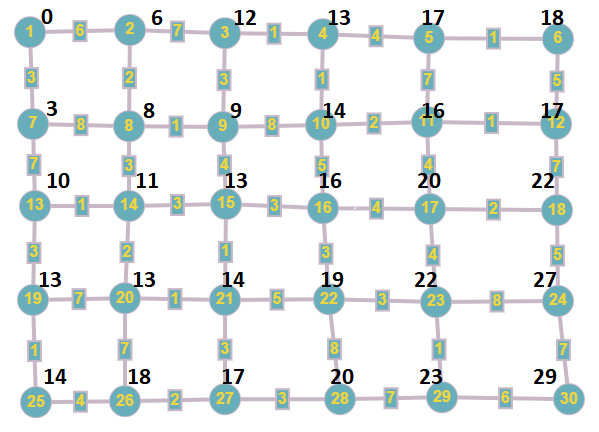
**Варіант 15**

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

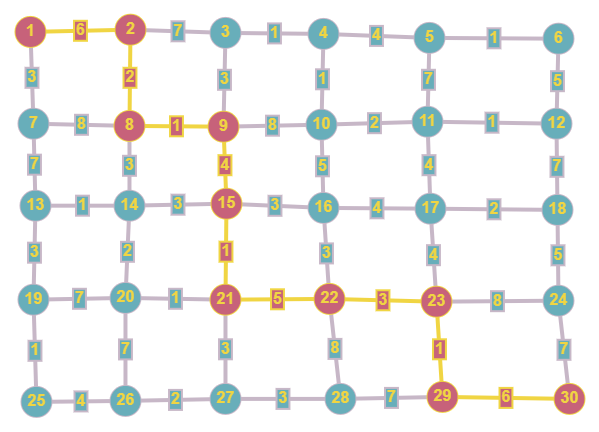
1. За допомогою алгоритму Дейкстра знайти найкоротший шлях у графі поміж парою вершин V0 і V\* .



За допомогою алгоритму Дейкстри знайдемо найкортший шлях від V0 до кожної вершини графа (мінімальний шлях відображений біля вершин):

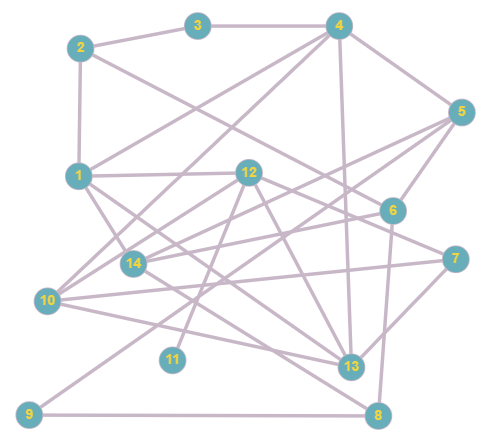


Тепер покажемо найменшу відстань від вершини 1 до вершини 30:

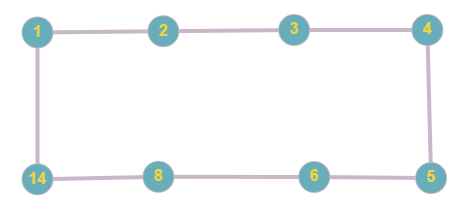


Отже, найкоротша відстань від вершини 1 до вершини 30 рівна **29**.

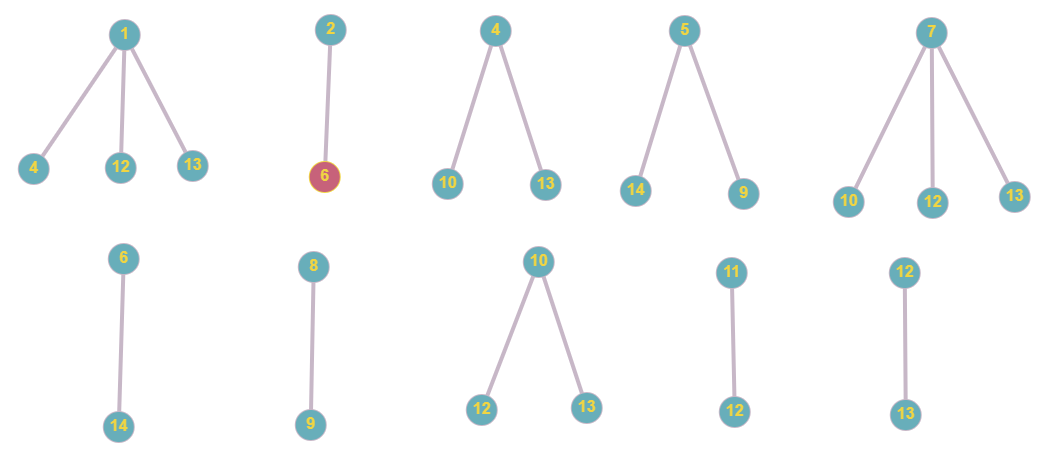
1. За допомогою γ -алгоритма зробити укладку графа у площині, або довести що вона неможлива.



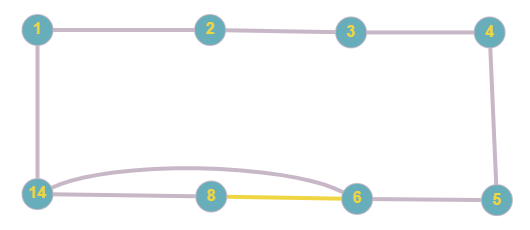
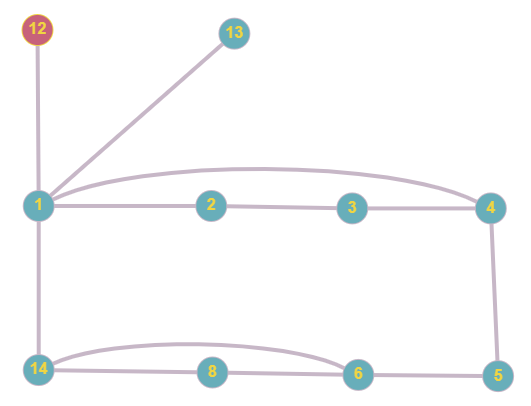
**Розв’язання:** вибираємо з даного графа простий цикл:

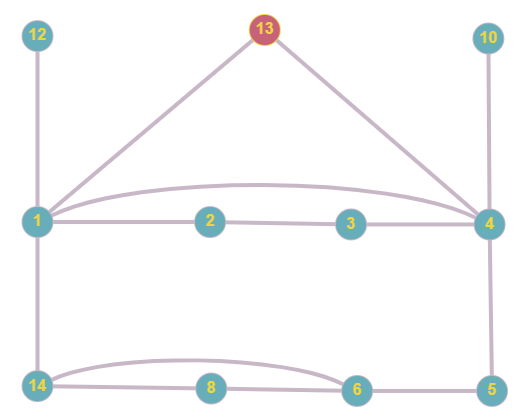
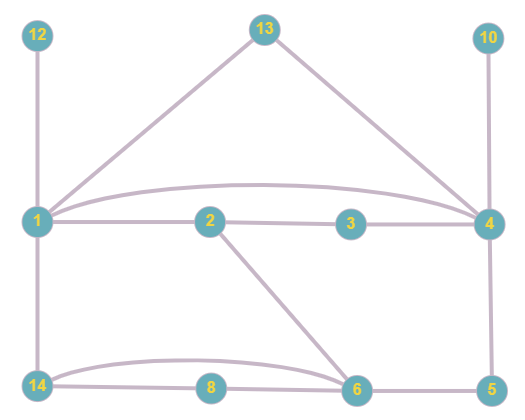


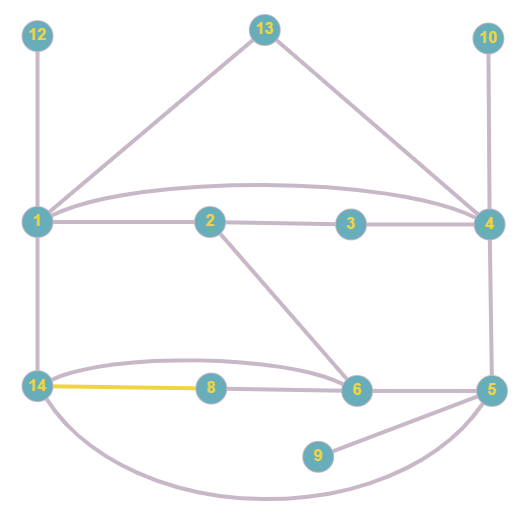
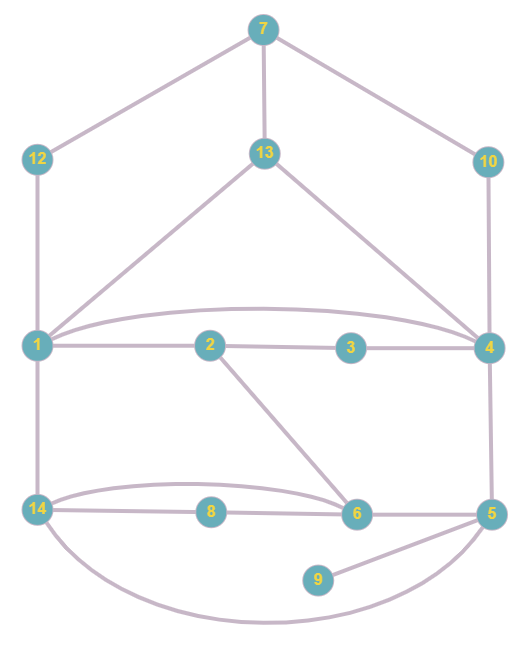
Розбиваємо граф на сегменти:

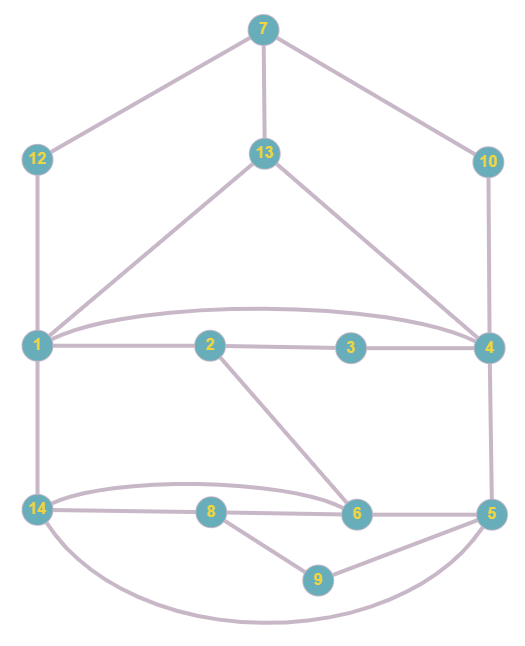
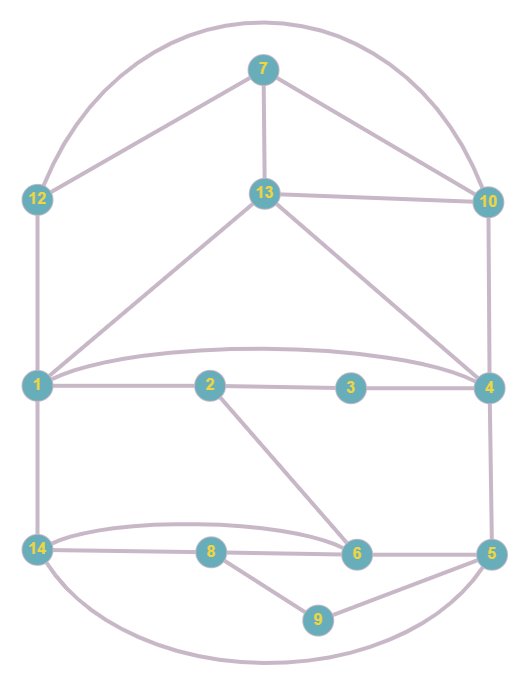


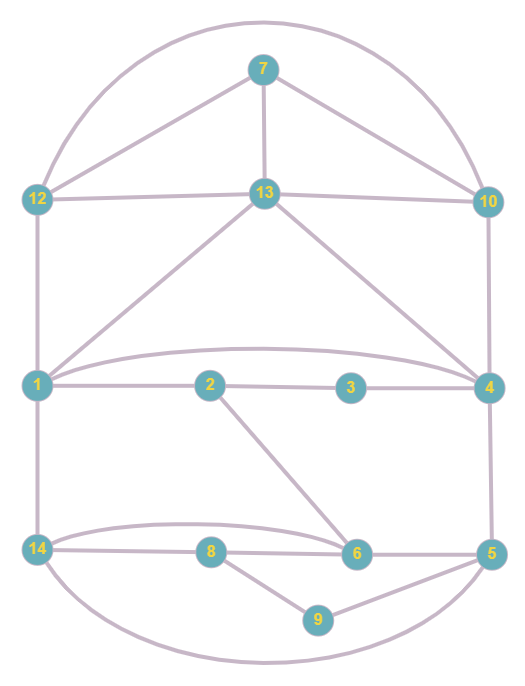
Поступово додавати сегменти у планарний граф:

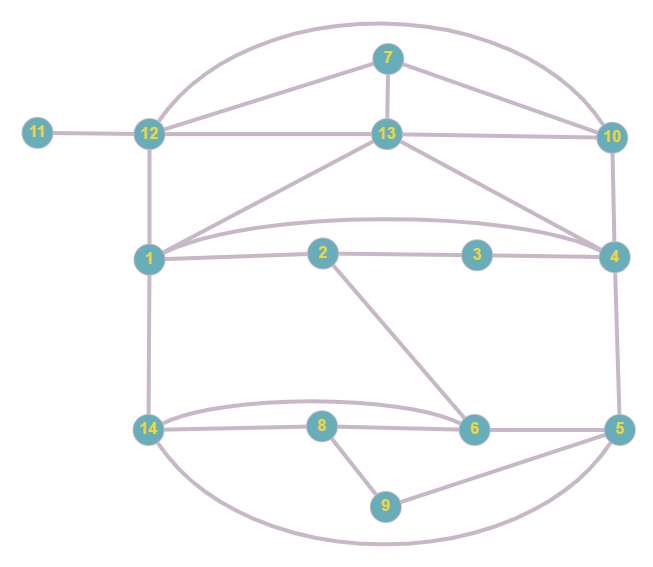
 

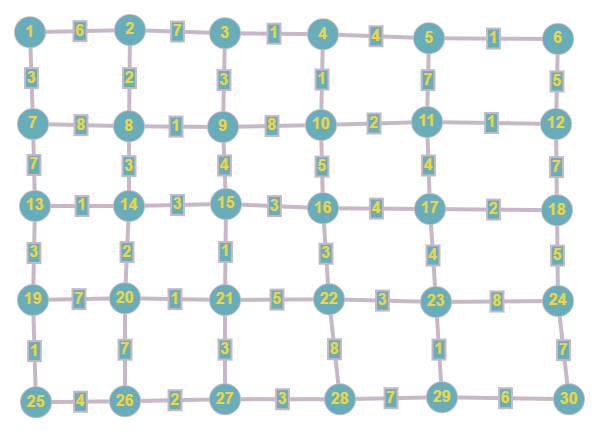
 



**Кінцевий результат:**



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



**Текст програми**

#include <iostream>

using namespace std;

const int N = 30;

const int INF = 20000;

void main()

{

setlocale(LC\_ALL, "ukr");

int start;

int graph[N][N] = {

{0, 6, 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, 0, 0},

{6, 0, 7, 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},

{0, 7, 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},

{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, 0, 0, 0, 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, 0, 0, 0},

{0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{3, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 2, 0, 0, 0, 0, 8, 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, 3, 0, 0, 0, 0, 1, 0, 8, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 0, 0, 0, 0, 8, 0, 2, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 7, 0, 0, 0, 0, 2, 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, 5, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 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, 3, 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, 0, 4, 0, 0, 0, 0, 3, 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, 5, 0, 0, 0, 0, 3, 0, 4, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 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, 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, 0, 3, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 7, 0, 1, 0, 0, 0, 0, 7, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 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, 3, 0, 0, 0, 0, 5, 0, 3, 0, 0, 0, 0, 8, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 3, 0, 8, 0, 0, 0, 0, 1, 0},

{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, 0, 7},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 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, 7, 0, 0, 0, 0, 4, 0, 2, 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, 2, 0, 3, 0, 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, 7, 0},

{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, 7, 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} };

do

{

cout << "Початкова вершина: ";

cin >> start;

} while (start < 1 || start > N);

int Dist[N];

int count, index, i, u, m;

m = start;

bool visited[N];

for (i = 0; i < N; i++)

{

Dist[i] = INF;

visited[i] = false;

}

Dist[start - 1] = 0;

for (count = 0; count < N; count++)

{

int min = INF;

for (i = 0; i < N; i++)

{

if (!visited[i] && Dist[i] <= min)

{

min = Dist[i];

index = i;

}

}

u = index;

visited[u] = true;

for (i = 0; i < N; i++)

if (!visited[i] && (graph[u][i] && Dist[u] != INF) && Dist[u] + graph[u][i] < Dist[i])

{

Dist[i] = Dist[u] + graph[u][i];

}

}

cout << "Вiдстань від заданої вершини до всiх вершин графа: " << endl;

for (i = 0; i < N; i++)

{

if (Dist[i] != INF)

{

cout << m << " -> " << i + 1 << " = " << Dist[i] << endl;

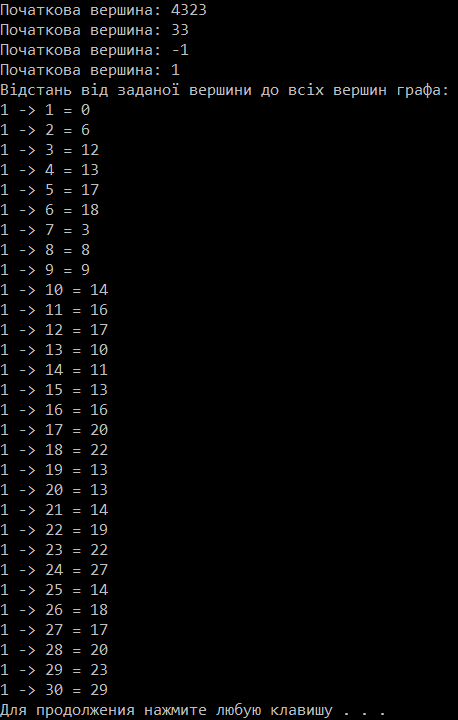
}

}

system("pause");

}

**Результати виконання програми**

****

**Висновок:** на цій лабораторній роботі я навчився знаходити найкоротший шлях за алгоритмом Дейкстри та укладати граф за допомогою алгоритму γ-укладання графа.