МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ

ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП’ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ЕЛЕКТРОНІКИ

ЗВІТ  
З ПРАКТИЧНОЇ РОБОТИ №6

Виконала:  
студентка групи КН-24-1  
Лабущак А.В.

Перевірив:  
доцент кафедри КІЕ  
Сидоренко В. М.

Кременчук 2025

# Тема: Графи. Найкоротші шляхи

**Мета:** набути практичних навичок розв'язання задач пошуку найкоротших шляхів у графі та оцінювання їх асимптотичної складності.

## Теоретичні відомості

**Постановка задачі пошуку найкоротшого шляху в графі**

Як і раніше, формально граф може бути представлений як G = (V, E).

Тут V — множина вершин графа, E — множина ребер. Кожне ребро має собою кортеж (v, w), де w, v ∈ V. Слоні можна додавати третій компонент, що відображає вагу ребра. Позначаємо s — набір ребер є з вершини v такої, що s ∈ E і v ∈ V.

Шлях між двома вершинами — це послідовність вершин, з'єднаних ребрами. Формально шлях можна визначити як w₁, w₂, ..., wₖ як такий, що (wᵢ, wᵢ₊₁) ∈ E для всіх 1 ≤ i ≤ n - 1.

Довжина шляху без ваг стає кількість у ньому ребер. Зважений шлях у графі буде сумою ваг усіх ребер, що входять до нього.

Алгоритм Дейкстри

Найкращі алгоритми призначені найшвидшими. Дано орієнтований або неорієнтований граф G = (V, E), де V — множина вершин, а E — множина ребер. Кожне ребро може мати вагу, що відображає вартість або довжину переходу між вершинами.

Задача полягає в пошуку найкоротшого шляху між двома заданими вершинами s (початкова вершина) і t (кінцева вершина). Найкоротшим шляхом розуміється шлях з найменшою сумарною вагою або найвеличікою відстань між вершинами.

### Алгоритм Дейкстри

Найвідоміший алгоритм Дейкстри з умовною процедурою Релаксація(u, v):

1. Ініціалізувати масив Найкоротший[], так, що Найкоротший[s] = 0, а всі інші вершини мають значення нескінченності. Отже, масив: Найкоротший[] = нескінченність для всіх шляхів крім вершини s.
2. Ініціалізувати чергу з пріоритетами Q, додавши всі вершини графа разом з відстанню до них у Найкоротший[].
3. Доки черга Q не порожня:
   1. Вилучаємо вершину u з найменшою відстанню з Q.
   2. Для кожного сусіднього вузла v з вершини u:
      1. Виконуємо процедуру Релаксація(u, v), яку:
      2. Якщо Найкоротший[u] + вага(u, v) < Найкоротший[v]:
         1. Зменшуємо ключ Q для вершини v на нову відстань (Найкоротший[u] + вага(u, v)).
4. По завершенні алгоритму масив Найкоротший[] містить найкоротші відстані від вершини s до всіх інших вершин графа.

### Алгоритм Беллмана-Форда

1. Ініціалізувати найкоротший шлях до кожної вершини як нескінченність, окрім джерела, для якого найкоротший шлях установлюється на 0.
2. Провести n-1 ітерацій, де n – кількість вершин у графі: 2.1. Пройти всі ребра графа. 2.2. Для кожного ребра (u, v) з вагою w напрямити вагу шляху до вершини v через вершину u з поточним найкоротшим шляхом до вершини v. Якщо такий шлях коротший, оновити найкоротший шлях до вершини v.
3. Перевірити наявність від'ємного циклу. Для цього знову пройти всі ребра графа і спробувати оновити найкоротші шляхи до кожної вершини. Якщо після n-1 ітерацій додатково відбувається оновлення найкоротших шляхів, це означає, що у графі є від'ємний цикл, і неможливо визначити найкоротші шляхи до всіх вершин.
4. Якщо від'ємного циклу немає, найкоротші шляхи до всіх вершин успішно визначено.

### Алгоритм Флойда-Воршалла

1. Ініціалізувати найкоротший шлях до кожної вершини як нескінченність, окрім джерела, для якого найкоротший шлях установлюється на 0.
2. Провести n-1 ітерацій, де n – кількість вершин у графі: 2.1. Пройти всі ребра графа. 2.2. Для кожного ребра (u, v) з вагою w напрямити вагу шляху до вершини v через вершину u з поточним найкоротшим шляхом до вершини v. Якщо такий шлях коротший, оновити найкоротший шлях до вершини v.
3. Перевірити наявність від'ємного циклу. Для цього знову пройти всі ребра графа і спробувати оновити найкоротші шляхи до кожної вершини. Якщо після n-1 ітерацій дополніється відбувається оновлення найкоротших шляхів, це означає, що у графі є від'ємний цикл і неможливо визначити найкоротші шляхи до всіх вершин.
4. Якщо від'ємного циклу немає, найкоротші шляхи до всіх вершин успішно визначено.

Алгоритм Дейкстри має кращу асимптотичну складність у найгіршому випадку, особливо коли граф є розрідженим, тобто коли кількість ребер значно менша, ніж кількість вершин. Проте, у найгіршому випадку, алгоритм Дейкстри має таку саму складність, як і алгоритм Беллмана-Форда.

Алгоритм Беллмана-Форда є універсальним і може працювати з графами з від'ємними вагами ребер і знаходити від'ємні цикли, але він має гіршу асимптотичну складність порівняно з алгоритмом Дейкстри у найгіршому випадку.

Алгоритм Флойда-Воршалла має кубічну асимптотичну складність і призначений для знаходження найкоротших шляхів між усіма парами вершин у графі. Цей алгоритм ефективний лише для невеликих графів, оскільки в інших випадках його складність може бути неприйнятною.

## Індивідуальне завдання (варіант 11)

**Завдання 3, варіант 1, але за алгоритмом Беллмана-Форда.**

Граф із завдання 3:

* Вершини: {1, 2, 3, 4, 5}
* Ребра з вагами: {(1,2,5), (1,3,2), (2,4,7), (2,5,1), (3,4,6), (3,5,8), (4,5,3)}

### Розв'язання за алгоритмом Беллмана-Форда

**Початкові дані:**

* Початкова вершина: 1
* Кількість вершин: n = 5
* Кількість ребер: m = 7

**Ініціалізація:**

* dist[1] = 0 (відстань до початкової вершини)
* dist[2] = ∞
* dist[3] = ∞
* dist[4] = ∞
* dist[5] = ∞

**Ітерація 1:** Перевіряємо всі ребра:

* Ребро (1,2) з вагою 5: dist[2] = min(∞, 0 + 5) = 5
* Ребро (1,3) з вагою 2: dist[3] = min(∞, 0 + 2) = 2
* Ребро (2,4) з вагою 7: dist[4] = min(∞, 5 + 7) = 12
* Ребро (2,5) з вагою 1: dist[5] = min(∞, 5 + 1) = 6
* Ребро (3,4) з вагою 6: dist[4] = min(12, 2 + 6) = 8
* Ребро (3,5) з вагою 8: dist[5] = min(6, 2 + 8) = 6
* Ребро (4,5) з вагою 3: dist[5] = min(6, 8 + 3) = 6

Після ітерації 1: dist = [0, 5, 2, 8, 6]

**Ітерація 2:** Перевіряємо всі ребра:

* Ребро (1,2) з вагою 5: dist[2] = min(5, 0 + 5) = 5
* Ребро (1,3) з вагою 2: dist[3] = min(2, 0 + 2) = 2
* Ребро (2,4) з вагою 7: dist[4] = min(8, 5 + 7) = 8
* Ребро (2,5) з вагою 1: dist[5] = min(6, 5 + 1) = 6
* Ребро (3,4) з вагою 6: dist[4] = min(8, 2 + 6) = 8
* Ребро (3,5) з вагою 8: dist[5] = min(6, 2 + 8) = 6
* Ребро (4,5) з вагою 3: dist[5] = min(6, 8 + 3) = 6

Після ітерації 2: dist = [0, 5, 2, 8, 6]

**Ітерація 3:** Перевіряємо всі ребра:

* Ребро (1,2) з вагою 5: dist[2] = min(5, 0 + 5) = 5
* Ребро (1,3) з вагою 2: dist[3] = min(2, 0 + 2) = 2
* Ребро (2,4) з вагою 7: dist[4] = min(8, 5 + 7) = 8
* Ребро (2,5) з вагою 1: dist[5] = min(6, 5 + 1) = 6
* Ребро (3,4) з вагою 6: dist[4] = min(8, 2 + 6) = 8
* Ребро (3,5) з вагою 8: dist[5] = min(6, 2 + 8) = 6
* Ребро (4,5) з вагою 3: dist[5] = min(6, 8 + 3) = 6

Після ітерації 3: dist = [0, 5, 2, 8, 6]

**Ітерація 4:** Перевіряємо всі ребра:

* Жодних змін не відбулося

Після ітерації 4: dist = [0, 5, 2, 8, 6]

**Перевірка на від'ємні цикли:** Виконуємо додаткову ітерацію для перевірки від'ємних циклів. Оскільки жодна відстань не змінилася, від'ємних циклів немає.

**Результат:**

* Найкоротша відстань від вершини 1 до вершини 1: 0
* Найкоротша відстань від вершини 1 до вершини 2: 5
* Найкоротша відстань від вершини 1 до вершини 3: 2
* Найкоротша відстань від вершини 1 до вершини 4: 8
* Найкоротша відстань від вершини 1 до вершини 5: 6

**Найкоротші шляхи:**

* 1 → 1: шлях 1 (довжина 0)
* 1 → 2: шлях 1→2 (довжина 5)
* 1 → 3: шлях 1→3 (довжина 2)
* 1 → 4: шлях 1→3→4 (довжина 8)
* 1 → 5: шлях 1→2→5 (довжина 6)

## Практична реалізація

def bellman\_ford(graph, start):

"""

Алгоритм Беллмана-Форда для знаходження найкоротших шляхів

graph: словник, де ключі - вершини, значення - список кортежів (сусід, вага)

start: початкова вершина

"""

# Ініціалізація відстаней

distances = {vertex: float('infinity') for vertex in graph}

distances[start] = 0

# Ініціалізація попередників для відновлення шляхів

predecessors = {vertex: None for vertex in graph}

# Релаксація ребер (n-1) разів

for \_ in range(len(graph) - 1):

for vertex in graph:

if distances[vertex] != float('infinity'):

for neighbor, weight in graph[vertex]:

if distances[vertex] + weight < distances[neighbor]:

distances[neighbor] = distances[vertex] + weight

predecessors[neighbor] = vertex

# Перевірка на від'ємні цикли

for vertex in graph:

if distances[vertex] != float('infinity'):

for neighbor, weight in graph[vertex]:

if distances[vertex] + weight < distances[neighbor]:

raise ValueError("Граф містить від'ємний цикл")

return distances, predecessors

def reconstruct\_path(predecessors, start, end):

"""Відновлення шляху від початкової до кінцевої вершини"""

path = []

current = end

while current is not None:

path.append(current)

current = predecessors[current]

path.reverse()

if path[0] == start:

return path

else:

return None # Шлях не існує

# Граф для тестування (варіант 11)

graph = {

1: [(2, 5), (3, 2)],

2: [(4, 7), (5, 1)],

3: [(4, 6), (5, 8)],

4: [(5, 3)],

5: []

}

# Знаходимо найкоротші шляхи від вершини 1

distances, predecessors = bellman\_ford(graph, 1)

print("Найкоротші відстані від вершини 1:")

for vertex, distance in distances.items():

print(f"До вершини {vertex}: {distance}")

print("\nНайкоротші шляхи:")

for vertex in range(1, 6):

if vertex != 1:

path = reconstruct\_path(predecessors, 1, vertex)

if path:

print(f"1 → {vertex}: {' → '.join(map(str, path))} (довжина {distances[vertex]})")

**Асимптотична складність алгоритмів**

Алгоритм Беллмана-Форда має часову складність O(|V| × |E|), де |V| - кількість вершин, |E| - кількість ребер. У найгіршому випадку, коли граф є повним, E = V², тому складність становить O(V³).

Алгоритм Беллмана-Форда є універсальним і може працювати з від'ємними вагами ребер, на відміну від алгоритму Дейкстри. Це робить його корисним для розв'язання більш широкого кола задач, хоча за рахунок гіршої асимптотичної складності.

## Контрольні запитання

· **Що таке граф і які головні складові його структури?** Граф - це математична структура, що складається з множини вершин (V) та множини ребер (E), які з'єднують ці вершини. Формально граф записується як G = (V, E).

· **Які алгоритми використовуються для пошуку найкоротших шляхів у графах?** Основні алгоритми: Дейкстри, Беллмана-Форда, Флойда-Воршалла, A\*, алгоритм пошуку в ширину (BFS) для невизначених графів.

· **Як працює алгоритм Дейкстри і його особливості?** Алгоритм Дейкстри використовує жадібний підхід, завжди вибираючи вершину з найменшою відстанню. Він працює лише з невід'ємними вагами ребер і має складність O((V + E) log V).

· **Що таке алгоритм Беллмана-Форда і коли варто застосовувати?** Алгоритм Беллмана-Форда може працювати з від'ємними вагами ребер і виявляти від'ємні цикли. Він має складність O(V × E) і використовується, коли граф містить від'ємні ваги.

· **Як працює алгоритм Флойда-Воршалла і які його переваги та недоліки?** Алгоритм Флойда-Воршалла знаходить найкоротші шляхи між усіма парами вершин. Має складність O(V³), що робить його ефективним лише для невеликих графів, але він простий у реалізації та може працювати з від'ємними вагами.

## Висновки

У ході виконання практичної роботи було вивчено основні алгоритми пошуку найкоротших шляхів у графах, зокрема алгоритм Беллмана-Форда. Було розглянуто теоретичні основи алгоритмів та їх практичну реалізацію.

Отримано навички:

* Роботи з зваженими орієнтованими графами
* Застосування алгоритму Беллмана-Форда для пошуку найкоротших шляхів
* Виявлення від'ємних циклів у графах
* Аналізу асимптотичної складності алгоритмів
* Практичної реалізації алгоритмів на мові програмування Python
* Порівняння різних алгоритмів пошуку найкоротших шляхів

Алгоритм Беллмана-Форда є важливим інструментом для розв'язання задач пошуку найкоротших шляхів, особливо коли граф містить від'ємні ваги ребер. Хоча він має гіршу асимптотичну складність порівняно з алгоритмом Дейкстри, його універсальність робить його незамінним для певного класу задач.

Практичне застосування таких алгоритмів знаходить широке використання в навігаційних системах, мережевій маршрутизації, логістиці та багатьох інших областях.