МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

НАВЧАЛЬНА ДИСЦИПЛІНА
«**АЛГОРИТМИ І СТРУКТУРИ ДАНИХ**»

ЗВІТ
З ПРАКТИЧНОЇ РОБОТИ №6

Виконав:
студент групи КН-24-1
Соломка Б. О.

Перевірив:
доцент кафедри АІС
Сидоренко В. М.

Кременчук 2025

Тема: Графи. Найкоротші шляхи

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

Хід роботи

1. Теоретичні відомості

Граф формально представляється як G = (V, E), де V – множина вершин, а E – множина ребер, що їх з'єднують. Шлях у графі – це послідовність вершин, з'єднаних ребрами.

Задача пошуку найкоротшого шляху полягає у знаходженні шляху з мінімальною сумарною вагою між двома вершинами графа.

Основні алгоритми розв'язання цієї задачі:

Алгоритм Дейкстри

Алгоритм Белмана-Форда

Алгоритм Флойда-Воршала

2. Реалізація алгоритму Дейкстри

```
import heapq

def dijkstra(graph, start):
    """
    Реалізація алгоритму Дейкстри для пошуку найкоротших шляхів
    від початкової вершини до всіх інших вершин графа

    Args:
        graph: словник, де ключ - вершина, значення - словник сусідніх вершин та ваг ребер
        start: початкова вершина

    Returns:
        shortest_paths: словник найкоротших шляхів від start до кожної вершини
```

```
    """
    # Ініціалізація
    shortest_paths = {vertex: float('infinity') for vertex in graph}
    shortest_paths[start] = 0
    priority_queue = [(0, start)]

    while priority_queue:
        # Вибираємо вершину з найменшою відстанню
        current_distance, current_vertex = heapq.heappop(priority_queue)

        # Якщо поточна відстань більша за відому найкоротшу, пропускаємо
        if current_distance > shortest_paths[current_vertex]:
            continue

        # Перевіряємо всіх сусідів поточної вершини
        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight

            # Релаксація: Якщо знайдено коротший шлях до сусіда
            if distance < shortest_paths[neighbor]:
                shortest_paths[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return shortest_paths
```

3. Реалізація алгоритму Белмана-Форда

```
def bellman_ford(graph, start):
    """
    Реалізація алгоритму Белмана-Форда для пошуку найкоротших шляхів
    від початкової вершини до всіх інших вершин графа

    Args:
        graph: словник, де ключ - вершина, значення - словник сусідніх
вершин та ваг ребер
        start: початкова вершина

    Returns:
        shortest_paths: словник найкоротших шляхів від start до кожної
вершини
        або False, якщо виявлено від'ємний цикл
    """
    # Ініціалізація
```

```
    shortest_paths = {vertex: float('infinity') for vertex in graph}
    shortest_paths[start] = 0
    edges = []

    # Формуємо список всіх ребер
    for u in graph:
        for v, weight in graph[u].items():
            edges.append((u, v, weight))

    # Релаксація всіх ребер |V| - 1 разів
    for _ in range(len(graph) - 1):
        for u, v, weight in edges:
            if shortest_paths[u] != float('infinity') and
shortest_paths[u] + weight < shortest_paths[v]:
                shortest_paths[v] = shortest_paths[u] + weight

    # Перевірка на від'ємні цикли
    for u, v, weight in edges:
        if shortest_paths[u] != float('infinity') and shortest_paths[u] +
weight < shortest_paths[v]:
            return False  # Знайдено від'ємний цикл

    return shortest_paths
```

4. Реалізація алгоритму Флойда-Воршала

```
def floyd_warshall(graph):
    """
    Реалізація алгоритму Флойда-Воршала для пошуку найкоротших шляхів
    між усіма парами вершин у графі

    Args:
        graph: матриця суміжності, де graph[i][j] - вага ребра від i до
j,
                  або float('infinity'), якщо ребро відсутнє

    Returns:
        dist: матриця найкоротших шляхів між усіма парами вершин
    """
    # Копіюємо матрицю, щоб не змінювати вхідні дані
    dist = [row[:] for row in graph]
    n = len(dist)
```

```
    # Алгоритм Флойда-Воршала
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist
```

5. Практичне завдання - розв'язання прикладу з тексту

Розглянемо приклад з тексту (рис. 1.2), де потрібно знайти найкоротші шляхи від вершини 1 до всіх інших за допомогою алгоритму Дейкстри.

```
# Представлення графа з прикладу 1.2
graph = {
    1: {2: 10, 3: 5, 4: 3},
    2: {3: 1},
    3: {},
    4: {2: 1}
}

# Запуск алгоритму Дейкстри
start_vertex = 1
shortest_paths = dijkstra(graph, start_vertex)

# Виведення результатів
print(f"Найкоротші шляхи від вершини {start_vertex} до всіх інших:")
for vertex, distance in shortest_paths.items():
    print(f"До вершини {vertex}: {distance}")
```

Результат виконання:

```
Найкоротші шляхи від вершини 1 до всіх інших:
До вершини 1: 0
До вершини 2: 4
До вершини 3: 5
До вершини 4: 3
```

6. Порівняння асимптотичної складності алгоритмів

Алгоритм	**Найгірший випадок**	**Найкращий випадок**
Дейкстри	O(|E| + |V|log|V))	Θ(|V|^2)
Белмана-Форда	Θ(|V||E|)	Θ(|E|)
Флойда-Воршала	Θ(|V|^3)	Θ(|V|^3)

Таблиця 1 - Порівняння складності алгоритмів

Висновки

В ході практичної роботи були розглянуті та реалізовані три основні алгоритми пошуку найкоротших шляхів у графах: алгоритм Дейкстри, алгоритм Белмана-Форда та алгоритм Флойда-Воршала.

Алгоритм Дейкстри є найефективнішим для графів з невід'ємними вагами ребер, особливо для розріджених графів, завдяки логарифмічній складності. Однак він не працює з від'ємними вагами.

Алгоритм Белмана-Форда має гіршу асимптотичну складність, але може працювати з графами, що містять ребра з від'ємними вагами, та виявляти від'ємні цикли.

Алгоритм Флойда-Воршала має кубічну складність і підходить для знаходження найкоротших шляхів між усіма парами вершин у графах невеликого розміру.

У практичному завданні було успішно застосовано алгоритм Дейкстри для пошуку найкоротших шляхів від вершини 1 до всіх інших вершин графа. Отримані результати підтверджують коректність роботи алгоритму і відповідають теоретичним очікуванням.