МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА
ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНОЇ РОБОТИ №6

з навчальної дисципліни
«Алгоритми та методи обчислень»

Тема «Графи. Найкоротші шляхи»

Студент гр. КІ-24-1, Варакута О.О
Викладач, Сидоренко В. М

Кременчук 2025

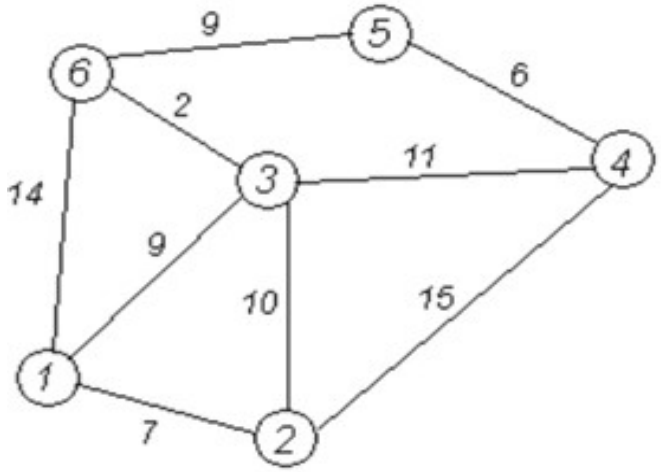
Тема. Графи. Найкоротші шляхи

Мета: Набути практичних навичок розв’язання задач пошуку найкоротших шляхів у зваженому графі за допомогою алгоритму Дейкстри та оцінювання його асимптотичної складності.

Хід роботи

Завдання : Завдання полягає у знаходженні найкоротших шляхів від вершини 1 до всіх інших за допомогою алгоритму Белмена–Форда.

Заданий граф:



Розв’язання Алгоритмом Беллмана-Форда:

Дані графа :

Вершини: 1, 2, 3, 4, 5, 6

Ребра (граф неорієнтований):
(1,2)=7
(1,3)=9
(1,6)=14
(2,3)=10
(2,4)=15
(3,4)=11
(3,6)=2

(4,5)=6
(5,6)=9

Потрібно знайти найкоротші шляхи від вершини 1 до всіх інших.

Оскільки вершин 6, виконуємо |V| - 1 = 5 проходів (ітерацій) релаксації всіх ребер.

граф неорієнтований, тому кожне ребро можна розглядати у двох напрямках (u → v і v → u).

Ітерація 1 (релаксуємо всі ребра)

Релаксація ребер від 1:
1 → 2: d(2) = min(∞, d(1)+7) = 7
1 → 3: d(3) = min(∞, d(1)+9) = 9
1 → 6: d(6) = min(∞, d(1)+14) = 14

Далі релаксуємо інші ребра:
3 → 6 (вага 2): d(6) = min(14, d(3)+2) = min(14, 9+2) = 11
3 → 4 (вага 11): d(4) = min(∞, d(3)+11) = 9+11 = 20
6 → 5 (вага 9): d(5) = min(∞, d(6)+9) = 11+9 = 20

Перевірка інших ребер:
2 → 3: d(3) = min(9, 7+10=17) = 9 (не змінюється)
2 → 4: d(4) = min(20, 7+15=22) = 20 (не змінюється)
4 → 5: d(5) = min(20, 20+6=26) = 20 (не змінюється)

Після ітерації 1 маємо:
d(1)=0
d(2)=7
d(3)=9
d(6)=11
d(4)=20
d(5)=20

Ітерація 2

Початкові значення перед ітерацією:
d(1)=0
d(2)=7
d(3)=9
d(6)=11
d(4)=20
d(5)=20

Виконуємо релаксацію всіх ребер графа.

Перевірка ребер від вершини 1:
1 → 2: 0 + 7 = 7, d(2)=7, зменшення немає
1 → 3: 0 + 9 = 9, d(3)=9, зменшення немає
1 → 6: 0 + 14 = 14, d(6)=11, зменшення немає

Перевірка ребер від вершини 2:
2 → 3: 7 + 10 = 17, d(3)=9, зменшення немає
2 → 4: 7 + 15 = 22, d(4)=20, зменшення немає

Перевірка ребер від вершини 3:
3 → 4: 9 + 11 = 20, d(4)=20, значення не змінюється
3 → 6: 9 + 2 = 11, d(6)=11, значення не змінюється

Перевірка ребер від вершини 6:
6 → 5: 11 + 9 = 20, d(5)=20, значення не змінюється

Перевірка ребра:
4 → 5: 20 + 6 = 26, d(5)=20, зменшення немає

Після ітерації 2 жодне значення не зменшилось. Відстані залишилися без змін.

Ітерація 3:

Початкові значення перед ітерацією:

d(1)=0

d(2)=7

d(3)=9

d(6)=11

d(4)=20

d(5)=20

Виконуємо релаксацію всіх ребер графа.

Перевірка ребер від вершини 1:

1 → 2: 0 + 7 = 7, d(2)=7, зменшення немає

1 → 3: 0 + 9 = 9, d(3)=9, зменшення немає

1 → 6: 0 + 14 = 14, d(6)=11, зменшення немає

Перевірка ребер від вершини 2:

2 → 3: 7 + 10 = 17, d(3)=9, зменшення немає

2 → 4: 7 + 15 = 22, d(4)=20, зменшення немає

Перевірка ребер від вершини 3:

3 → 4: 9 + 11 = 20, d(4)=20, значення не змінюється

3 → 6: 9 + 2 = 11, d(6)=11, значення не змінюється

Перевірка ребер від вершини 6:

6 → 5: 11 + 9 = 20, d(5)=20, значення не змінюється

Перевірка ребра:

4 → 5: 20 + 6 = 26, d(5)=20, зменшення немає

Після ітерації 3 жодне значення не зменшилось. Відстані залишилися без змін.

Ітерація 4:

Початкові значення перед ітерацією:

d(1)=0

d(2)=7

d(3)=9

d(6)=11

d(4)=20

d(5)=20

Виконуємо релаксацію всіх ребер графа.

Перевірка ребер від вершини 1:

1 → 2: 0 + 7 = 7, d(2)=7, зменшення немає

1 → 3: 0 + 9 = 9, d(3)=9, зменшення немає

1 → 6: 0 + 14 = 14, d(6)=11, зменшення немає

Перевірка ребер від вершини 2:

2 → 3: 7 + 10 = 17, d(3)=9, зменшення немає

2 → 4: 7 + 15 = 22, d(4)=20, зменшення немає

Перевірка ребер від вершини 3:

3 → 4: 9 + 11 = 20, d(4)=20, значення не змінюється

3 → 6: 9 + 2 = 11, d(6)=11, значення не змінюється

Перевірка ребер від вершини 6:

6 → 5: 11 + 9 = 20, d(5)=20, значення не змінюється

Перевірка ребра:

4 → 5: 20 + 6 = 26, d(5)=20, зменшення немає

Після ітерації 4 жодне значення не зменшилось. Нових коротших шляхів не знайдено.

Ітерація 5

Початкові значення перед ітерацією:

d(1)=0

d(2)=7

d(3)=9

d(6)=11

d(4)=20

d(5)=20

Виконуємо релаксацію всіх ребер графа.

Перевірка ребер від вершини 1:

1 → 2: 0 + 7 = 7, d(2)=7, зменшення немає

1 → 3: 0 + 9 = 9, d(3)=9, зменшення немає

1 → 6: 0 + 14 = 14, d(6)=11, зменшення немає

Перевірка ребер від вершини 2:

2 → 3: 7 + 10 = 17, d(3)=9, зменшення немає

2 → 4: 7 + 15 = 22, d(4)=20, зменшення немає

Перевірка ребер від вершини 3:

3 → 4: 9 + 11 = 20, d(4)=20, значення не змінюється

3 → 6: 9 + 2 = 11, d(6)=11, значення не змінюється

Перевірка ребер від вершини 6:

6 → 5: 11 + 9 = 20, d(5)=20, значення не змінюється

Перевірка ребра:

4 → 5: 20 + 6 = 26, d(5)=20, зменшення немає

Після ітерації 5 жодне значення не зменшилось

Змін також немає.

остаточні найкоротші відстані:

від 1 до 2: 7

від 1 до 3: 9

від 1 до 6: 11

від 1 до 4: 20

від 1 до 5: 20

Відновлення найкоротших шляхів (логічно з релаксацій):

1→2 (7)

1 → 3 (9)

1→ 3 → 6 (9 + 2 = 11)

1 → 3 → 4 (9 + 11 = 20)

1 → 3 → 6 → 5 (9 + 2 + 9 = 20)

Перевірка розв’язку за допомогою Python:

```
Import math

def bellman_ford_undirected(nodes, edges, start):

    # Перетворюємо неорієнтовані ребра на два напрямки
    directed_edges = []
    for u, v, w in edges:
        directed_edges.append((u, v, w))
        directed_edges.append((v, u, w))

    # Ініціалізація
    dist = {v: math.inf for v in nodes}
    parent = {v: None for v in nodes}
    dist[start] = 0

    # |V| - 1 ітерацій релаксації
    for _ in range(len(nodes) - 1):
        for u, v, w in directed_edges:
            if dist[u] != math.inf and dist[u] + w <
dist[v]:
                dist[v] = dist[u] + w
                parent[v] = u

    # Перевірка на від’ємний цикл
    for u, v, w in directed_edges:
        if dist[u] != math.inf and dist[u] + w <
dist[v]:
            return dist, parent, True

    return dist, parent, False

def restore_path(parent, start, target):
    """
    Відновлення найкоротшого шляху
    """
    path = []
    cur = target
    while cur is not None:
        path.append(cur)
        if cur == start:
            break
        cur = parent[cur]

    path.reverse()
    if path and path[0] == start:
```

```
            return path
        return None

def main():
    nodes = [1, 2, 3, 4, 5, 6]
    edges = [
        (1, 2, 7),
        (1, 3, 9),
        (1, 6, 14),
        (2, 3, 10),
        (2, 4, 15),
        (3, 4, 11),
        (3, 6, 2),
        (4, 5, 6),
        (5, 6, 9),
    ]

    start = 1
    dist, parent, neg_cycle =
bellman_ford_undirected(nodes, edges, start)

    if neg_cycle:
        print("У графі є від’ємний цикл, найкоротші
шляхи не визначаються.")
        return

    print("Найкоротші відстані від вершини", start)
    for v in nodes:
        print("до", v, "=", dist[v])

    print("\nНайкоротші шляхи:")
    for v in nodes:
        path = restore_path(parent, start, v)
        print("1 ->", v, ":", path, "довжина =",
dist[v])
main()
```

Найкоротші відстані від вершини 1

до 1 = 0

до 2 = 7

до 3 = 9

до 4 = 20

до 5 = 20

до 6 = 11

Найкоротші шляхи:

1 → 1 : [1] довжина = 0

1 → 2 : [1, 2] довжина = 7

1→3 : [1, 3] довжина = 9

1 → 4 : [1, 3, 4] довжина = 20

1 → 5 : [1, 3, 6, 5] довжина = 20

1 →6 : [1, 3, 6] довжина = 11

Висновок : завдання розв’язано коректно.

Контрольні питання

1.Що таке граф і які головні складові його структури?

Граф -це математична структура, яка використовується для моделювання відношень між об’єктами. Граф складається з множини вершин і множини ребер, які з’єднують пари вершин. Вершини представляють об’єкти, а ребра - зв’язки між ними. Граф може бути орієнтованим або неорієнтованим, зваженим або незваженим. У зваженому графі кожному ребру ставиться у відповідність числова вага, яка визначає вартість, довжину або час переходу між вершинами.

2.Які алгоритми використовуються для пошуку найкоротших шляхів у графах?

Для пошуку найкоротших шляхів у графах використовуються такі основні алгоритми: алгоритм Дейкстри, алгоритм Беллмена-Форда та алгоритм Флойда–Форшала. Алгоритм Дейкстри застосовується для знаходження найкоротших шляхів у графах без від’ємних ваг ребер. Алгоритм Беллмена–Форда є універсальнішим, оскільки дозволяє працювати з від’ємними вагами та виявляти від’ємні цикли. Алгоритм Флойда-Форшала використовується для знаходження найкоротших шляхів між усіма парами вершин у графі.

3.Як працює алгоритм Дейкстри і які його особливості?

Алгоритм Дейкстри знаходить найкоротші шляхи від однієї початкової вершини до всіх інших вершин графа. Робота алгоритму починається з ініціалізації відстаней: для початкової вершини відстань дорівнює нулю, для всіх інших - нескінченності. Далі алгоритм ітеративно вибирає вершину з найменшою поточною відстанню, яка ще не була оброблена, та виконує релаксацію всіх ребер, що виходять з цієї вершини. Процес триває до обробки всіх вершин. Особливістю алгоритму є те, що він працює швидко для розріджених графів, але не може застосовуватися до графів з від'ємними вагами ребер.

4.Що таке алгоритм Белмена–Форда і коли його варто застосовувати?

Алгоритм Беллмена-Форда - це алгоритм пошуку найкоротших шляхів від однієї вершини до всіх інших у зваженому графі. Його особливістю є можливість роботи з від'ємними вагами ребер та здатність виявляти від'ємні цикли. Алгоритм працює шляхом багаторазової релаксації всіх ребер графа протягом |V| - 1 ітерацій, де |V| - кількість вершин. Алгоритм Беллмена-Форда доцільно застосовувати у випадках, коли у графі можуть бути від'ємні ваги або коли необхідно перевірити наявність від'ємних циклів.

5.Як працює алгоритм Флойда–Форшала і які його переваги та недоліки?

Алгоритм Флойда-Форшала призначений для знаходження найкоротших шляхів між усіма парами вершин у графі. Він ґрунтується на методі динамічного програмування і послідовно перевіряє, чи може шлях між двома вершинами бути коротшим через проміжну вершину. Алгоритм простий у реалізації та працює як для орієнтованих, так і для неорієнтованих графів, допускає наявність від'ємних ваг ребер (за відсутності від'ємних циклів). Основним недоліком алгоритму є висока обчислювальна складність $O(|V|^3)$, що робить його неефективним для графів з великою кількістю вершин.