Практична робота № 8

Тема. Жадібні алгоритми. Наближене розв’язання екстремальних задач

Мета: набути практичних навичок застосування деяких жадібних алгоритмів для розв’язання екстремальних задач.

Постановка задачі.

Виконати індивідуальне завдання. Завдання полягає у розв’язанні єдиного завдання для всіх, вибравши граф згідно з варіантом. Номер варіанта відповідає номеру студента у списку групи. У разі, якщо було досягнуто кінця списку задач, потрібно циклічно повернутися на його початок.

Завдання №14

1,2

```
import itertools
import networkx as nx
import matplotlib.pyplot as plt


# Заданий зважений граф
edges = [(1, 3, 12), (1, 4, 15), (1, 5, 18), (2, 3, 20), (2, 4, 22), (2, 5, 25)]


# Створення графа з реберами та їх вагами
G = nx.Graph()
for edge in edges:
    G.add_edge(edge[0], edge[1], weight=edge[2])


# Розв'язання задачі комівояжера
def traveling_salesman(graph):
    min_cost = float('inf')
    min_path = None
    nodes = graph.nodes()
    for perm in itertools.permutations(nodes):
```

```
        cost = sum(graph[perm[i]][perm[i + 1]]['weight'] for i in range(len(perm) - 1))
        cost += graph[perm[-1]][perm[0]]['weight']  # Замикання циклу
        if cost < min_cost:
            min_cost = cost
            min_path = perm
    return min_path, min_cost


# Отримання найдовшого шляху та його ваги
optimal_path, optimal_cost = traveling_salesman(G)
print("Найдовший шлях комівояжера:", optimal_path)
print("Мінімальна вага:", optimal_cost)


# Візуалізація графа
pos = nx.spring_layout(G)  # Позиціонування вершин для кращого відображення
nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=1500,
edge_color='black', linewidths=1, font_size=15)
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.title("Зважений граф")
plt.show()
```

3. Алгоритм найближчого сусіда має асимптотичну складність $O(N2)$, де N - кількість точок.

Основні операції в алгоритмі:

1. Вибір початкової точки - $O(1)$.
2. Прохід через всі точки, щоб знайти найближчу точку для кожної - $O(N)$.
3. Повторення цього процесу для кожної точки - $O(N)$.

Отже, загальна асимптотика алгоритму найближчого сусіда становить $O(N2)$.

Для використаного коду відповідність асимптотиці може бути такою:

- Вибір початкової точки: $O(1)$.
- Пошук найближчої точки для кожної: $O(N)$ (вкладений цикл).
- Повний прохід через всі точки: $O(N2)$.

Таким чином, загальна асимптотична складність алгоритму становить $O(N2)$.

Контрольні запитання.

4. **Жадібний алгоритм** - це алгоритм для розв'язання оптимізаційних задач, який працює шляхом вибору найкращого доступного варіанту на кожному кроці з метою максимізації або мінімізації деякої цільової функції.
5. **Принципи роботи жадібних алгоритмів**:
    - **Локальне оптимальне рішення**: на кожному кроці алгоритму вибирається локально найкращий варіант.
    - **Безвідкладність**: алгоритм здійснює вибір кожного кроку без урахування виборів, зроблених на попередніх кроках.
    - **Недорогість**: жадібний вибір кожного кроку не вимагає обчислення всіх можливих варіантів.
6. **Відмінність між жадібними алгоритмами та динамічним програмуванням**:
    - У динамічному програмуванні розв'язок знаходиться шляхом розбиття вихідної задачі на підзадачі, а потім використовується підзадачі для знаходження розв'язку.
    - У жадібних алгоритмах кожне рішення приймається без зазирання в майбутні можливості, і вибір кожного кроку здійснюється на основі поточно доступної інформації без урахування можливих наслідків.
7. **Приклади задач, які можна розв'язати за допомогою жадібних алгоритмів**:
    - Задача про каскадні монети (грошовий розподіл).
    - Задача про розміщення датчиків або точок доступу.
    - Задача про покриття інтервалами.
    - Задача про маршрутизацію в комп'ютерних мережах.
8. **Обмеження у використанні жадібних алгоритмів для розв'язання екстремальних задач**:
    - Недостатня глобальна інформація: жадібні алгоритми можуть обирати локально оптимальні рішення, які не завжди призводять до оптимального загального розв'язку.
    - Наявність локальних максимумів або мінімумів: у деяких задачах можуть існувати локальні максимуми або мінімуми, які жадібний алгоритм може випадково опустити.
9. **Чому жадібні алгоритми часто використовуються для наближеного розв'язання екстремальних задач**:

- Жадібні алгоритми часто використовуються для наближеного розв'язання екстремальних задач через їхню простоту та ефективність. Вони зазвичай працюють швидко і не вимагають великої обчислювальної потужності, що робить їх привабливими для використання у реальних задачах. Крім того, жадібні алгоритми можуть надавати достатньо прийнятний розв'язок, особливо якщо точний розв'язок недосяжний за прийнятний час. Однак важливо розуміти, що жадібні алгоритми не гарантують знаходження оптимального розв'язку для всіх задач, тому їх слід використовувати з обережністю та з розумінням їхніх обмежень.