УДК 006.4

Сапожнікова В.Є., студентка 1 курсу
Спеціальність 122 «Комп'ютерні науки»
Науковий керівник:
Гончар В. М., асистент кафедри
інформаційних технологій

ТЕОРІЯ ГРАФІВ: ЗАДАЧІ ПРО МАКСИМАЛЬНУ КІЛЬКІСТЬ РЕБЕР, ЯКУ МОЖНА ВИДАЛИТИ З ГРАФА БЕЗ РОЗ'ЄДНАННЯ ЙОГО НА ДВІ ЧАСТИНИ

Донецький національний університет імені Василя Стуса, м. Вінниця

Теорія графів є важливою галуззю математики, яка досліджує взаємозв'язки між об'єктами за допомогою графів. Одна з цікавих задач теорії графів полягає в визначенні максимальної кількості ребер, які можна видалити з графа без роз'єднання його на дві частини.

Припустимо, у нас є граф з N вершинами і M ребрами. Якщо ми видаляємо ребро з графа, то можливо створити дві окремі компоненти, які раніше були з'єднані. Метою задачі є знайти максимальну кількість ребер, які можна видалити з графа, так щоб граф залишився з'єднаним.

Одним із підходів до розв'язання цієї задачі є використання алгоритму пошуку в ширину (BFS). Алгоритм BFS дозволяє перевірити, чи можна з однієї вершини досягти іншої вершини в графі. Застосовуючи цей алгоритм до кожної пари вершин графа, ми можемо визначити, чи є ребро необхідним для збереження з'єднаності графа[1].

Наприклад:

```
from collections import deque

def bfs(graph, start, end):
    visited = set()  # відвідані вершини
    queue = deque()  # черга для обходу
    queue.append(start)
    while queue:
        vertex = queue.popleft()  # виймаємо вершину з початку черги
        if vertex == end:  # якщо досягли кінцевої вершини, повертаємо True
            return True
        if vertex not in visited:
            visited.add(vertex)  # додаємо вершину в список відвіданих
            neighbors = graph[vertex]  # отримуємо сусідні вершини поточної вершини
            queue.extend(neighbors)  # додаємо сусідні вершини в кінець черги
    return False  # якщо кінцева вершина недосяжна, повертаємо False
graph = {               # Приклад використання
```

```
        'A': ['B', 'C'],
        'B': ['D', 'E'],
        'C': [],
        'D': [],
        'E': ['F'],
        'F': []
    }
    start_vertex = 'A'
    end_vertex = 'F'
    result = bfs(graph, start_vertex, end_vertex)
    print(result)
```

Результат виконання коду буде виводити True або False, в залежності від того, чи існує шлях від стартової вершини до кінцевої у графі.

Інший підхід до розв'язання цієї задачі базується на понятті моста. Містом у графі називається ребро, яке при видаленні роз'єднує граф на дві компоненти. Знаходження мостів у графі допоможе визначити, які ребра є критичними для збереження з’єднаності.

Для розв'язання цієї задачі також можна використовувати алгоритм Максимального Потоку-Мінімального Перерізу. Цей алгоритм дозволяє визначити максимальний потік у графі та знайти мінімальний переріз, що розділяє джерело та стік у графі.

Наприклад:

```
    import networkx as nx

    def find_critical_edges(graph, source, sink):
        flow_value, flow_dict = nx.maximum_flow(graph, source, sink, capacity='capacity')  # Знаходимо максимальний потік та мінімальний переріз
        cut_value, partition = nx.minimum_cut(graph, source, sink, capacity='capacity')
        critical_edges = []  # Визначаємо критичні ребра
        for u, neighbors in flow_dict.items():
            for v, flow in neighbors.items():
                if flow > 0 and (u, v) not in critical_edges and (v, u) not in critical_edges:
                    if u in partition[0] and v in partition[1]:
                        critical_edges.append((u, v))
        return critical_edges
    graph = nx.DiGraph()  # Приклад використання
    # Додавання ребер та їх пропускних здатностей
    graph.add_edge('A', 'B', capacity=1)
    graph.add_edge('A', 'C', capacity=1)
    graph.add_edge('B', 'D', capacity=1)
    graph.add_edge('C', 'D', capacity=1)
```

```
    graph.add_edge('D', 'E', capacity=1)
    graph.add_edge('E', 'F', capacity=1)
    source = 'A'
    sink = 'F'
    result = find_critical_edges(graph, source, sink)
    print(result)
```

Результат виконання коду буде виводити список критичних ребер, які можна видалити з графа без роз'єднання його на дві частини.

Алгоритм Максимального Потоку-Мінімального Перерізу базується на понятті мережі, де кожен зв'язок між вузлами має певну пропускну здатність (capacity). Задача полягає в тому, щоб знайти максимальний потік, який може пройти через мережу від джерела (source) до стоку (sink), дотримуючись обмежень пропускної здатності кожного зв’язку[2].

Одним з відомих алгоритмів для розв'язання цієї задачі є алгоритм Форда-Фалкерсона, який використовує пошук шляху з більшим потоком через ребра мережі. Ітераційно цей алгоритм збільшує потік через мережу, поки існує можливий шлях від джерела до стоку.

Після знаходження максимального потоку можна використати алгоритм мінімального перерізу, який дозволяє знайти найменшу сумарну пропускну здатність ребер, що розділяють джерело та стік. Цей переріз відокремлює джерело від стоку і представляє мінімальну пропускну здатність, через яку не може пройти максимальний потік.

Список літератури

1. *"Introduction to Graph Theory" by Douglas B. West, URL: https://athena.nitc.ac.in/summerschool/Files/West.pdf*
2. *"Graphs and Their Uses" by Oystein Ore, URL: http://www.ams.org/books/nml/034/nml034-endmatter.pdf*