ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

BỘ MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Báo cáo

Đề tài: Giải bài toán Set Cover và TSP bằng thuật toán gần đúng

---

Môn học: Phân tích và thiết kế thuật toán

Sinh viên thực hiện:

Nguyen Minh Huy(23520634)

Do Quang Luc(23520902)

Giáo viên hướng dẫn:

Nguyen Thanh Son

Ngày 19 tháng 12 năm 2024

Bài 1

Câu hỏi 1: Chi phí với hai thuật toán GBFS và UCS

1.1. Thuật toán Greedy Best-First Search (GBFS)

- Đường đi: London → Hamburg → Falsterbo → Danzig → Visby → Tallinn → Novgorod.
- Tổng chi phí thực tế:

$$801 + 324 + 498 + 588 + 606 + 474 = 3291.$$

1.2. Thuật toán Uniform Cost Search (UCS)

- Đường đi: London → Amsterdam → Hamburg → Lubeck → Falsterbo → Copenhagen → Danzig → Visby → Riga → Tallinn → Novgorod.
- Tổng chi phí thực tế:

$$395 + 411 + 64 + 262 + 143 + 570 + 588 + 297 + 310 + 474 = 3514.$$

Kết luận:

- GBFS tìm đường nhanh hơn, nhưng không đảm bảo tối ưu. Tổng chi phí là 3291.
- UCS đảm bảo tìm đường đi có chi phí thấp nhất, với tổng chi phí là 3514.
- 2 thuật toán trên không tối ưu, vì nếu chạy Dijkstra thì đáp án là 2667

Bài 2

1. Ý tưởng và phương pháp thiết kế thuật toán

Bài toán yêu cầu kiểm tra xem trong đồ thị có hồi quy (chu trình âm) hay không. **Thuật toán Bellman-Ford** được sử dụng để giải quyết bài toán này. Thuật toán này vừa tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh khác, vừa phát hiện chu trình âm (nếu tồn tại).

Phương pháp thiết kế:

- **Biểu diễn đồ thị**:

  - Đồ thị được biểu diễn bằng danh sách cạnh (edges). Mỗi cạnh bao gồm ba thông tin: u (nút đầu), v (nút cuối), w (trọng số).

- **Thuật toán Bellman-Ford**:

  1. Khởi tạo khoảng cách từ *nguồn* s đến các đỉnh là ∞, riêng $distance[s] = 0$.
  2. Lặp $N - 1$ lần (với N là số đỉnh):
     - Duyệt qua tất cả các cạnh (u, v, w).
     - Cập nhật khoảng cách: $distance[v] = \min(distance[v], distance[u] + w)$.
  3. Sau $N - 1$ lần lặp, duyệt qua danh sách cạnh lần nữa:
     - Nếu $distance[v] > distance[u] + w$, thì tồn tại chu trình âm.

- **Truy vết chu trình âm**:

  Nếu phát hiện chu trình âm, duyệt ngược bằng mảng *parent*[] để xác định các đỉnh trong chu trình.

2. Mã giả

```
function BellmanFord(graph, N, M):
    distance = [\infty, \infty, ..., \infty]  # Khởi tạo mảng khoảng cách
    parent = [-1, -1, ..., -1]                # Mảng lưu vết
    distance[1] = 0                           # Giả sử bắt đầu từ đỉnh 1

    # Thực hiện N-1 lần lặp để cập nhật khoảng cách
    for i = 1 to N-1:
        for (u, v, w) in graph:  # Duyệt qua tất cả các cạnh
            if distance[u] + w < distance[v]:
                distance[v] = distance[u] + w
```

```
            parent[v] = u

# Kiểm tra chu trình âm
for (u, v, w) in graph:
    if distance[u] + w < distance[v]:
        # Chu trình âm được phát hiện, truy vết chu trình
        x = v
        for i = 1 to N:  # Truy ngược N lần để đảm bảo x nằm trong chu trình
            x = parent[x]

        # Truy vết thứ tự các đỉnh trong chu trình
        cycle = []
        current = x
        while True:
            cycle.append(current)
            if current == x and len(cycle) > 1:
                break
            current = parent[current]
        cycle.reverse()
        return ("YES", cycle)

return "NO"
```

3. Tính toán độ phức tạp

- **Thời gian**:
  - Thuật toán Bellman-Ford thực hiện $N - 1$ lần lặp, duyệt qua M cạnh trong mỗi lần: $O(N \cdot M)$.
  - Phần truy vết chu trình mất $O(N)$.
  - **Tổng thời gian**: $O(N \cdot M)$.

- **Không gian**:
  - Mảng $distance[]$ và $parent[]$: $O(N)$.
  - **Tổng không gian**: $O(N)$.