Nhóm 10 giải bt nhóm 15

Nguyễn Trần Quang Minh - 23520943
Hồ Ngọc Luật - 23520900

Bài toán

Tìm đường đi tối ưu từ London (2114) đến Novgorod (0) sử dụng hai thuật toán Greedy và UCS:

- **Greedy**: Dựa trên hàm heuristic từ nút hiện tại đến đích.
- **UCS**: Tìm đường đi có chi phí nhỏ nhất tại mỗi bước.

Thuật toán Greedy

- Bắt đầu từ **London (2114)**.
- Tại mỗi bước, chọn nút con có giá trị heuristic thấp nhất.

Đường đi tìm được bởi Greedy:

1. London → Amsterdam (1777)
2. Amsterdam → Hamburg (1422)
3. Hamburg → Danzig (901)
4. Danzig → Novgorod (0)

Kết quả: Chi phí tìm được không tối ưu, do Greedy chỉ quan tâm tới heuristic.

Thuật toán UCS

- Bắt đầu từ **London** và duyệt các nút dựa trên chi phí đường đi tới nút đó.

Đường đi tìm được bởi UCS:

1. London → Amsterdam (395)
2. Amsterdam → Copenhagen (953)
3. Copenhagen → Danzig (376)
4. Danzig → Novgorod (901)

Chi phí tổng: $395 + 953 + 376 + 901 = 2625$.

So sánh kết quả

- Thuật toán Greedy đi theo heuristic nhưng không tối ưu chi phí.
- Thuật toán UCS đảm bảo tìm được đường đi tối ưu.

Kết luận

- Thuật toán UCS cho kết quả tối ưu hơn so với Greedy.
- Greedy nhanh nhưng không chắc chắn tối ưu chi phí.

Bài toán vòng lặp âm vô tận

Kaiser là một cảnh sát kỳ cựu và cần xác định xem có **vòng lặp âm vô tận** trong hệ thống các thành phố hay không. Bài toán yêu cầu xác định sự tồn tại của vòng lặp âm và in ra đường đi nếu có.

Dữ liệu vào

- Dòng đầu tiên chứa hai số nguyên N và M - số thành phố và số đường đi giữa các thành phố.
- M dòng tiếp theo, mỗi dòng gồm ba số nguyên a, b, c: đường một chiều nối từ thành phố a đến b với chi phí c.

Dữ liệu ra

- In **YES** nếu có vòng lặp âm vô tận và in ra đường đi của vòng lặp.
- Nếu không có vòng lặp âm, in **NO**.

Ví dụ

Input:

```
4 5
1 2 1
2 4 1
2 3 1
3 1 -3
4 3 -2
```

Output:

```
YES
1 2 4 3 1
```

Mã giả

```
function BellmanFord(N, M, edges):
    dist = [inf] * (N + 1)
    parent = [-1] * (N + 1)
    dist[1] = 0
    x = -1

    for i from 1 to N:
        x = -1
        for edge (u, v, w) in edges:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                parent[v] = u
                x = v

    if x == -1:
        print("NO")
        return

    for i from 1 to N:
        x = parent[x]

    cycle = []
    v = x
    while True:
```

```
        cycle.append(v)
        if v == x and len(cycle) > 1:
            break
        v = parent[v]

    cycle.reverse()
    print("YES")
    print(*cycle)
```

Độ phức tạp

- Thời gian: $O(N \times M)$ với N là số thành phố và M là số đường đi.
- Không gian: $O(N)$.