CS112.P11.KHTN - Bài tập nhóm 12

Nhóm 3
Nguyễn Hữu Đặng Nguyên - 23521045
Trần Vạn Tấn - 23521407

Ngày 19 tháng 12 năm 2024

Bài 1

Tóm tắt đề bài: Xác định chi phí và đường đi ngắn nhất từ London đến Novgorod

Yêu cầu 1 + 2: Chi phí và chi tiết đường đi nếu dùng thuật toán Greedy và UCS

Thuật toán Greedy: Tại mỗi bước, chọn thành phố gần Novgorod nhất dựa trên khoảng cách Euclid, không xét chi phí thực tế trên các cạnh.

- Bắt đầu từ London. Lựa chọn giữa: Bergen (1467), Amsterdam (1777), Hamburg (1422) => Chọm Hamburg
- Từ Hamburg. Lựa chọn giữa: Falsterbo (1166), Lubeck (1365) => Chọn Falsterbo.
- Từ Falsterbo. Lựa chọn giữa: Danzig (901) => Chọn Danzig.
- Từ Danzig. Lựa chọn giữa: Visby (768), Lubeck(1365), Copenhagen(1167) => Chọn Visby.
- Từ Visby. Lựa chọn giữa: Riga (459), Talin(387) => Chọn Talinn.
- Từ Talinn. => Chọn Novgorod(0)
- **Kết luận:**
  - Tuyến đường Greedy: London → Hamburg → Falsterbo → Danzig → Visby -> Talinn → Novgorod.

– Chi phí: 801 + 324 + 498 + 606 + 590 + 474 = 3293

Thuật toán UCS: Tại mỗi bước, chọn đường đi có tổng chi phí thực tế nhỏ nhất từ London đến Novgorod.

1. **Bắt đầu:** Thành phố **London** với chi phí = 0.
2. **Danh sách thành phố đã thăm và chi phí nhỏ nhất:**

Bước	Thành phố	Chi phí	Các thành phố kề được cập nhật
1	London	0	Amsterdam(395), Hamburg(801), Bergen(1554)
2	Amsterdam	395	Hamburg(806)
3	Hamburg	801	Copenhagen(1126),Lubeck(865), Falsterbo(1125)
4	Lubeck	865	Falsterbo(1127), Danzig(1453), Visby(1603)
5	Falsterbo	1125	Danzig(1623)
6	Copenhagen	1126	
7	Danzig	1453	
8	Visby	1603	Riga(1804), Talinn(2193)
9	Riga	1804	Talinn(2109)
10	Talinn	2109	Novgorod(2583)

Kết luận:

- Tuyến đường USC: London -> Hamburg -> Lubeck -> Visby -> Riga -> Talinn -> Novgorod.
- Chi phí: 801 + 64 + 738 + 201 + 305 + 474 = 2583.

Yêu cầu 3: Đường đi tìm được bởi Greedy và UCS có tối ưu không?

Thuật toán Greedy: Thuật toán Greedy không đảm bảo tìm được đường đi tối ưu vì nó chỉ xem xét chi phí ngay tại thời điểm hiện tại mà không nhìn vào toàn bộ bài toán. Greedy có thể chọn con đường ngắn nhất hiện tại nhưng không phải là tối ưu nhất trong dài hạn. Bù đắp lại, Greedy rất đơn giản và nhanh chóng.
Thuật toán UCS: UCS đảm bảo tìm được đường đi tối ưu vì nó luôn mở rộng các đỉnh có chi phí thấp nhất đầu tiên và tìm đường đi tối ưu trong đồ

thị có trọng số không âm. UCS phải cài đặt và quản lý phức tạp hơn Greedy nhiều.
Kết luận: Đường đi tìm được bởi Greedy đa số không được tối ưu, trong khi UCS luôn tìm được đường đi tối ưu nếu chi phí không âm.

Bài 2

Đề bài: Xác định xem có tồn tại chu trình âm trong đồ thị và in ra nếu có.

Ý tưởng thuật toán

- Thuật toán Bellman-Ford được sử dụng để phát hiện chu trình âm trong đồ thị có trọng số.
- Ý tưởng chính là lặp qua tất cả các cạnh $|V| - 1$ lần (với V là số đỉnh) để đảm bảo cập nhật khoảng cách ngắn nhất từ nguồn đến tất cả các đỉnh.
- Nếu sau $|V| - 1$ lần lặp mà vẫn có thể tối ưu khoảng cách, tồn tại chu trình âm trong đồ thị.
- Để in ra chu trình âm, sử dụng một mảng *parent[]* để lưu đỉnh cha của mỗi đỉnh, từ đó truy vết lại.

Phương pháp thiết kế

- **Khởi tạo:**
  - Gán khoảng cách từ nguồn đến tất cả các đỉnh khác và mảng parent để truy vết
- **Cập nhật khoảng cách:**
  - Với mỗi cạnh (u, v, weight), nếu $distance[u] + weight < distance[v]$, cập nhật $distance[v]$ và lưu đỉnh cha $parent[v] = u$.
- **Kiểm tra chu trình âm:**
  - Nếu sau $|V| - 1$ lần lặp, vẫn tồn tại cạnh (u, v, weight) thỏa $distance[u] + weight < distance[v]$, ta phát hiện chu trình âm.

– Lần ngược từ đỉnh v qua mảng *parent[]* để tìm và in ra chu trình.

Mã giả

```
function BellmanFord(vertices, edges, source):
    distance = [INF] * vertices
    distance[source] = 0
    parent = [-1] * vertices  # Dùng để theo dõi đường đi

    for i from 1 to vertices - 1:
        for (u, v, weight) in edges:
            if distance[u] + weight < distance[v]:
                distance[v] = distance[u] + weight
                parent[v] = u  # Lưu lại đỉnh cha

    for (u, v, weight) in edges:
        if distance[u] + weight < distance[v]:
            # Tìm chu trình âm
            cycle = []
            current = v
            for i in range(vertices):  # Tìm đỉnh thuộc chu trình
                current = parent[current]
            start = current

            while True:
                cycle.append(current)
                current = parent[current]
                if current == start and len(cycle) > 1:
                    break
            cycle.append(start)
            cycle.reverse()
            return "YES" + cycle

    return "NO"
```

Phân tích độ phức tạp

- Thời gian:
  - Thuật toán Bellman-Ford có độ phức tạp $O(V \times E)$, với V là số đỉnh và E là số cạnh do phải duyệt qua tất cả các cạnh trong $|V| - 1$ lần.
  - Tìm chu trình âm trong $O(V)$
  - DPT tổng quá $O(V \times E)$
- Không gian:
  - Sử dụng mảng khoảng cách và dùng vector để lưu lại đồ thị.