Lớp: CS112.P11.CTTN
Nhóm: 14
Sinh viên: Lê Nguyễn Anh Khoa. MSSV: 23520742
Sinh viên: Cáp Kim Hải Anh. MSSV: 23520036

BÀI TẬP
Vận dụng thiết kế thuật toán Graph Algorithms

Bài 1:

Đường đi từ London đến Novgorod:

a. Thuật toán Greedy:

- Cách thực hiện:
  - Bắt đầu từ London
  - Ưu tiên đi đến các node có chi phí Heuristic là nhỏ nhất mà không quan tâm đến chi phí di chuyển thực tế
  - Dừng khi đến Novgorod
- Đường đi: London(2114) → Hamburg(1422) → Falsterbo(1166) → Danzig(901) → Visby(768) → Talinn(387) → Novgorod(0).
- Chi phí: 801 + 324 + 498 + 606 + 590 + 474 = 3293
- Nhận xét: Thuật toán trên không tối ưu vì nó chỉ dựa trên chi phí Heuristic mà bỏ mất các chi phí thực tế khác.

b. Thuật toán UCS:

- Cách thực hiện:
  - Bắt đầu từ London
  - Ưu tiên đi đến các thành phố có tổng chi phí thực tế nhỏ nhất,
  - Dừng lại khi đã đến địa điểm đích hoặc không còn thành phố nào có thể di chuyển, ở đây dừng khi đến Novgorod
- Đường đi: London(2114) → Ansterdam(1777) → Hamburg(1422) → Lubeck(1365) → Danzig(901) → Visby(768) → Riga(459) → Talinn(387) → Novgorod(0).
- Chi phí: 395 + 411 + 64 + 262 + 738 + 201 + 305 + 474 = 2850
- Nhận xét: Đường đi tối ưu cho ra chi phí di chuyển nhỏ nhất

*** So sánh thuật toán greedy và UCS:**

- Greedy:
  - Ưu điểm: có thể tìm được một đường đi phù hợp và nhanh chóng
  - Nhược điểm: không tối ưu vì chỉ dựa trên chi phí Heuristic mà bỏ mất các chi phí thực tế khác, mặc dù giá trị heuristic các điểm trong tuyến đường thấp, nhưng nó không đảm bảo chi phí thực tế cũng nhỏ nhất

- UCS:
  - Ưu điểm: tối ưu nhờ xét chi phí thực tế từ gốc, đảm bảo tìm được đường đi có tổng chi phí thực tế nhỏ nhất mặc dù tuyến đường có thể dài hơn về số node trung gian nhưng tổng chi phí thực tế nhỏ hơn so với các tuyến đường khác
  - Nhược điểm: mở nhiều node hơn nên có thể chậm hơn so với greedy

- Độ phức tạp của cả 2 thuật toán: O(E.logV) với V là số đỉnh, E là số cạnh

- Kết luận: Đường đi UCS tối ưu hơn Greedy

Bài 2:

- Đề bài: Cho đồ thị gồm N đỉnh và M cạnh có hướng trọng số. Xác định xem có chu trình âm hay không?

- Lời giải:

+ Sử dụng thuật toán Bellman-Ford để kiểm tra xem có chu trình âm hay không

+ Thực hiện thuật toán trên N-1 lần từ đỉnh xuất phát

+ Nếu có cạnh nào làm giảm chi phí chắc chắn có chu trình âm.

+ Lưu vết vào mảng (parent) để truy xuất ra chu trình âm.

- Mã giả:

```
Input: N, M, edge(u, v, c)
Initialization:
Dist[1..n] = INF
Dist[start] = 0
Parent[1..n] = -1

Bellman-Ford:
Loop N times:
  For each edges (u, v, c):
     If Dist[u] + c < Dist[v]:
          Dist[v] = Dist[u] + c
          Parent[v] = u
```

Check for negative cycle:

For each edges (u, v, c):

If Dist[u] + c < Dist[v]:

Output: Yes

Tracing negative cycles using Parent

Exit

Output: No

- Độ phức tạp:

+ Thời gian: O(N x M) với N là số đỉnh, M là số cạnh của đồ thị

+ Không gian: O(N), lưu mảng dist và parent

- Kết luận: Thuật toán Bellman-Ford được sử dụng để phát hiện chu trình âm trong đồ thị có trọng số âm, thuật toán có độ phức tạp ổn định với các đồ thị có số đỉnh và cạnh lớn