Báo cáo giải thuật

Cảm ơn bạn đã góp ý chính xác. Đúng là về mặt **tư duy thiết kế thuật toán (Algorithm Design Paradigm)**, Dijkstra bản chất là một thuật toán **Tham lam (Greedy)**.

Dưới đây là bản báo cáo đã được chỉnh sửa để làm rõ phương pháp thiết kế là **Tham lam (Greedy)** kết hợp với **Mô hình hóa (Modeling)**.

Nhóm thực hiện : nhóm 10

1. Phương pháp thiết kế

Để giải quyết bài toán, tôi sử dụng kết hợp hai phương pháp thiết kế chính:

1. **Phương pháp Tham lam (Greedy Method):**
   - Đây là tư duy cốt lõi để tìm chi phí nhỏ nhất. Cụ thể, tôi áp dụng thuật toán **Dijkstra**.
   - **Nguyên lý tham lam:** Tại mỗi bước, thuật toán luôn chọn mở rộng từ đỉnh có chi phí tích lũy hiện tại nhỏ nhất (locally optimal) để cập nhật các đỉnh lân cận. Do trọng số cạnh không âm, việc lựa chọn tối ưu cục bộ này đảm bảo sẽ dẫn đến kết quả tối ưu toàn cục (global optimal).
2. **Mô hình hóa và Biến đổi (Transform and conquer):**
   - Bài toán được biến đổi từ việc lựa chọn các thao tác rời rạc thành bài toán tìm đường đi ngắn nhất trên đồ thị.
   - Sử dụng kỹ thuật **Đỉnh ảo (Auxiliary Nodes)** để tối ưu hóa đồ thị, giảm số lượng cạnh cần thiết từ $O(N^2)$ xuống $O(N \cdot M)$.

2. Tính phù hợp

Việc lựa chọn chiến lược Tham lam (cụ thể là Dijkstra) và Mô hình hóa đồ thị là phù hợp nhất vì:

- **Thỏa mãn điều kiện của thuật toán Tham lam:** Các chi phí trong bài toán (chi phí thuê c_i và chi phí tăng chỉ số k) đều luôn **không âm** (≥ 0). Đây là điều kiện tiên quyết để chiến lược tham lam của Dijkstra hoạt động chính xác. Nếu dùng Quy hoạch động (Dynamic Programming) sẽ phức tạp về trạng thái, còn nếu dùng Bellman-Ford thì không cần thiết và chậm hơn.
- **Tối ưu hóa ràng buộc dữ liệu:** Với N, M lớn nhưng tổng $N \cdot M \leq 4 \cdot 10^5$, việc xây dựng đồ thị trực tiếp giữa các Pokémon (N^2 cạnh) là bất khả thi. Phương pháp mô hình hóa qua

các đỉnh ảo giúp số lượng cạnh và đỉnh tỉ lệ tuyến tính với dữ liệu đầu vào, đảm bảo thuật toán tham lam chạy được trong giới hạn thời gian.

3. Phân tích độ phức tạp

Gọi $S = N \cdot M$ là tổng số lượng phần tử trong ma trận thuộc tính. Theo đề bài $\sum S \leq 4 \cdot 10^5$.

- **Độ phức tạp thời gian:**
  - *Xây dựng đồ thị:* Việc sắp xếp các giá trị thuộc tính để tạo đỉnh ảo tốn $O(S \log N)$.
  - *Thực thi thuật toán Tham lam (Dijkstra):* Đồ thị có số đỉnh $V \approx S$ và số cạnh $E \approx 4S$. Với cấu trúc dữ liệu Priority Queue (Heap), độ phức tạp là $O(E \log V)$.
  - **Tổng cộng:** $O(S \log S)$. Với $S = 4 \cdot 10^5$, thuật toán thực hiện khoảng vài triệu phép tính, hoàn toàn nằm trong giới hạn 5 giây ($ \approx 10^8$ phép tính/giây).
- **Độ phức tạp không gian:**
  - Lưu trữ đồ thị dưới dạng danh sách kề (Adjacency List). Vì số đỉnh và số cạnh đều tuyến tính với S.
  - **Tổng cộng:** $O(S)$. Mức tiêu thụ bộ nhớ khoảng vài chục MB, thấp hơn nhiều so với giới hạn 512 MB.