CS112 - Luyện Tập 2

Hồ Hữu Tây - 24520029
Dương Hoàng Việt - 24520036

December 3, 2025

Contents

---

1 Phương pháp thiết kế

Bài toán yêu cầu tìm chi phí nhỏ nhất để đưa Pokémon thứ n lên đấu trường thông qua hai thao tác: tăng chỉ số thuộc tính (trả phí k) và thuê Pokémon thi đấu (trả phí c_i).

Để giải quyết bài toán này, nhóm thực hiện mô hình hóa bài toán dưới dạng **Tìm đường đi ngắn nhất trên đồ thị (Shortest Path on Graph)**, cụ thể là sử dụng thuật toán **Dijkstra**.

1.1 Xây dựng đồ thị trạng thái

Thay vì xem xét từng Pokémon riêng lẻ, ta xem xét các trạng thái (u, j), trong đó:

- u: Là chỉ số của Pokémon đang được cân nhắc ($1 \leq u \leq n$).
- j: Là thuộc tính được sử dụng để so sánh/thi đấu ($1 \leq j \leq m$).

Tổng số trạng thái là $N \times M$. Với giới hạn $N \cdot M \leq 4 \cdot 10^5$, việc dựng đồ thị này là hoàn toàn khả thi.

1.2 Các cạnh và trọng số (Transitions)

Dựa trên phân tích, các chuyển đổi trạng thái được thiết kế như sau:

1.2.1 Chuyển đổi giữa các Pokémon cùng thuộc tính (Cạnh dọc)

Ta sắp xếp các Pokémon theo giá trị tăng dần của từng thuộc tính j. Gọi $rank[u][j]$ là thứ hạng của Pokémon u ở thuộc tính j.

- **Di chuyển lên (Thắng thế):** Từ Pokémon có thứ hạng thấp k sang Pokémon có thứ hạng cao hơn $k + 1$ (mạnh hơn về thuộc tính j).
  - Chi phí: 0.
  - Ý nghĩa: Nếu Pokémon yếu hơn thắng được, thì Pokémon mạnh hơn chắc chắn cũng thắng mà không tốn phí.
- **Di chuyển xuống (Nâng cấp):** Từ Pokémon có thứ hạng cao k sang Pokémon có thứ hạng thấp hơn $k - 1$ (yếu hơn về thuộc tính j).
  - Chi phí: $a_{next,j} - a_{curr,j}$.
  - Ý nghĩa: Đây là thao tác "tăng $a_{i,j}$ lên k". Chi phí chính là độ chênh lệch chỉ số cần bù đắp.

1.2.2 Chuyển đổi ngữ cảnh (Cạnh ngang - Thuê)

Từ trạng thái (u, j), ta có thể chuyển sang (u, k) với bất kỳ $k \in [1, m]$.

- Chi phí: c_u (Chi phí thuê Pokémon).
- Ý nghĩa: Khi Pokémon u đã đứng trên sân, ta trả phí c_u để chuẩn bị cho lượt đấu tiếp theo bằng bất kỳ thuộc tính nào khác.

2 Chứng minh tính đúng đắn

2.1 Mệnh đề 1: Tính bao phủ của không gian trạng thái

Mọi thao tác hợp lệ trong đề bài đều được ánh xạ vào đồ thị:

- Thao tác "Tăng thuộc tính" tương ứng với cạnh đi từ Pokémon có chỉ số cao xuống thấp trong danh sách đã sắp xếp.
- Thao tác "Thuê" tương ứng với việc chuyển đổi trạng thái thuộc tính và cộng phí c_u.

2.2 Mệnh đề 2: Tính tối ưu của thuật toán Dijkstra

Đồ thị được xây dựng có các trọng số không âm:

- Chi phí nâng cấp: $a_{high} - a_{low} \geq 0$.
- Chi phí thuê: $c_i \geq 1 > 0$.
- Chi phí di chuyển "miễn phí": 0.

Do đó, thuật toán Dijkstra luôn đảm bảo tìm ra đường đi ngắn nhất từ trạng thái khởi đầu (Pokémon 1) đến trạng thái kết thúc (Pokémon n).

3 Tính phù hợp của phương pháp

3.1 Tại sao không dùng Quy hoạch động (DP)?

Đồ thị trạng thái chứa các chu trình (đặc biệt là việc chuyển đổi qua lại giữa các thuộc tính và thuê lại Pokémon). Quy hoạch động thông thường khó xử lý chu trình dương. Việc sử dụng Dijkstra giúp xử lý các chu trình này một cách tự nhiên thông qua cơ chế hàng đợi ưu tiên (`priority_queue`).

3.2 Hiệu quả trong Python

- **Overhead thấp:** Sử dụng module `heapq` của Python (được viết bằng C) giúp các thao tác push/pop diễn ra rất nhanh.
- **Tối ưu bộ nhớ:** Việc sử dụng danh sách kề ẩn (tính toán trực tiếp chỉ số rank thay vì lưu ma trận cạnh tường minh) giúp tiết kiệm bộ nhớ đáng kể so với việc lưu toàn bộ $O((NM)^2)$ cạnh.

4 Phân tích độ phức tạp

Gọi $S = N \cdot M$ là tổng kích thước dữ liệu đầu vào.

4.1 Độ phức tạp Thời gian

1. **Sắp xếp:** Ta sắp xếp N Pokémon cho mỗi M thuộc tính. Chi phí là $O(M \cdot N \log N)$.
2. **Dijkstra:**
   - Số đỉnh $V = S$.
   - Số cạnh $E \approx 3S$ (mỗi đỉnh có tối đa 2 cạnh dọc và 1 cạnh ngang tượng trưng).
   - Độ phức tạp: $O(E \log V) \approx O(S \log S)$.

Tổng quát: $O(S \log S)$. Với $S \leq 4 \cdot 10^5$, thuật toán chạy tốt trong giới hạn 5 giây.

4.2 Độ phức tạp Không gian

Các mảng `a`, `rank`, `dec`, `dist` đều có kích thước $O(N \cdot M)$. Tổng không gian: $O(N \cdot M)$. Với giới hạn đề bài, bộ nhớ sử dụng khoảng vài chục MB, an toàn so với giới hạn 512MB.

5 Code minh họa (Python)

```
import sys
import heapq

input = sys.stdin.read
INF = 10**18

def solve():
    data = input().split()
    if not data:
        return
```

```
iterator = iter(data)

try:
    t = int(next(iterator))
except StopIteration:
    return

for _ in range(t):
    n = int(next(iterator))
    m = int(next(iterator))

    c = [0] * (n + 1)
    for i in range(1, n + 1):
        c[i] = int(next(iterator))

    a = [[0] * (m + 1) for _ in range(n + 1)]
    b = [[] for _ in range(m + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            val = int(next(iterator))
            a[i][j] = val
            b[j].append((val, i))

    rank = [[0] * (m + 1) for _ in range(n + 1)]
    dec = [[0] * (m + 1) for _ in range(n + 1)]

    for j in range(1, m + 1):
        b[j].sort()
        for k in range(n):
            val, idx = b[j][k]
            rank[idx][j] = k + 1
            dec[k + 1][j] = idx

    ans = INF
    vis = [False] * (n + 1)
    dist = [[INF] * (m + 1) for _ in range(n + 1)]
    pq = []

    vis[1] = True
    for j in range(1, m + 1):
        dist[1][j] = 0
        heapq.heappush(pq, (0, 1, j))

    while pq:
```

```
            w, x, t = heapq.heappop(pq)

            if w > dist[x][t]:
                continue

            if x == n:
                if w + c[n] < ans:
                    ans = w + c[n]

            curr_rank = rank[x][t]

            if curr_rank < n:
                z = dec[curr_rank + 1][t]
                if w < dist[z][t]:
                    dist[z][t] = w
                    heapq.heappush(pq, (w, z, t))

            if curr_rank > 1:
                z = dec[curr_rank - 1][t]
                new_w = w + a[x][t] - a[z][t]
                if new_w < dist[z][t]:
                    dist[z][t] = new_w
                    heapq.heappush(pq, (new_w, z, t))

            if not vis[x]:
                vis[x] = True
                switch_cost = w + c[x]
                for j in range(1, m + 1):
                    if switch_cost < dist[x][j]:
                        dist[x][j] = switch_cost
                        heapq.heappush(pq, (switch_cost, x, j))

        print(ans)

if __name__ == "__main__":
    solve()
```