Bài tập về nhà - Nhóm 3

Thành viên:
1. Hà Xuân Thiện - 24520031
2. Trần Quang Trường - 24521901

Ngày 18 tháng 12 năm 2025

Bài toán

Cho hoán vị p độ dài n. Ta được phép lặp lại thao tác:

Chọn $1 \le i < j < k \le n$ sao cho

$$p_i = \max(p_j, p_k) + 1 \quad \text{và} \quad p_i = \min(p_j, p_k) + 2.$$

Tức là bộ ba giá trị tại (i, j, k) đúng dạng $\{x, x+1, x+2\}$ và $p_i = x + 2$ (lớn nhất), còn p_j, p_k là x và $x + 1$ (không quan trọng thứ tự).

Sau đó cập nhật:

$$p_i \leftarrow p_i - 2, \quad p_j \leftarrow p_j + 1, \quad p_k \leftarrow p_k + 1.$$

Yêu cầu: tìm hoán vị nhỏ nhất theo thứ tự từ điển có thể thu được.

1 Phân tích & Ý tưởng thuật toán

Nhìn vào phép biến đổi

Gọi $x = \min(p_j, p_k)$, khi đó bộ ba là $(x+2, x, x+1)$ theo một thứ tự vị trí thỏa $i < j < k$ và $p_i = x + 2$. Sau phép biến đổi ta có:

$$(x + 2) \to x, \qquad x \to x + 1, \qquad (x + 1) \to x + 2.$$

Như vậy **ba giá trị $\{x, x+1, x+2\}$ chỉ bị hoán chuyển theo chu trình**, và đặc biệt:

- Vị trí i (nhỏ nhất trong ba chỉ số) luôn nhận **giá trị nhỏ nhất** x sau thao tác.
- Các giá trị nhỏ hơn x **không bao giờ bị ảnh hưởng**.

Điều này gợi ý một chiến lược tham lam (greedy) theo giá trị tăng dần:

> *Nếu có cơ hội đưa giá trị nhỏ x lên một vị trí sớm hơn bằng cách dùng bộ ba $\{x, x+1, x+2\}$ với $x+2$ đang đứng trước cả x và $x+1$, thì ta nên làm ngay vì sẽ cải thiện từ điển mà không làm thay đổi các số nhỏ hơn x.*

Điều kiện thực hiện theo vị trí các giá trị

Xét ba giá trị liên tiếp theo giá trị: $m, m{+}1, m{+}2$ (dùng chỉ số m 0-based trong cài đặt). Gọi:

$$a = \text{pos}(m), \quad b = \text{pos}(m{+}1), \quad c = \text{pos}(m{+}2).$$

Ta thực hiện được thao tác **khi và chỉ khi** $c < a$ và $c < b$ (tức $m{+}2$ đứng trước cả m và $m{+}1$). Khi đó chọn:

$$i = c, \quad j = \min(a, b), \quad k = \max(a, b)$$

sẽ đảm bảo $i < j < k$ và đúng dạng điều kiện đề bài.

Phương pháp thiết kế Greedy

Ta sẽ sử dụng phương pháp thiết kế Greedy.

Greedy phù hợp vì mục tiêu là tối ưu theo thứ tự từ điển, tức ưu tiên làm nhỏ các vị trí sớm nhất. Mỗi thao tác hợp lệ luôn làm giảm giá trị tại vị trí nhỏ nhất trong bộ ba, nên tạo ra cải thiện từ điển tức thời. Quan trọng hơn, thao tác chỉ ảnh hưởng đến ba giá trị liên tiếp $\{x, x+1, x+2\}$ và không làm thay đổi các giá trị nhỏ hơn x. Do đó, các quyết định đã tối ưu cho các giá trị nhỏ không bị phá vỡ khi xử lý các giá trị lớn hơn. Nhờ tính chất này, chiến lược tham lam tại mỗi bước là an toàn và dẫn đến nghiệm tối ưu toàn cục.

Ta duyệt $m = 0 \to n - 3$ và luôn kiểm tra điều kiện $c < a$ và $c < b$:

- Nếu đúng, thực hiện thao tác trên bộ $\{m, m{+}1, m{+}2\}$, cập nhật mảng p và mảng vị trí `pos`.
- Sau khi thao tác, có thể làm thay đổi quan hệ vị trí của các bộ lân cận, nên ta lùi m lại một chút (ví dụ $m \leftarrow \max(0, m-2)$) để kiểm tra lại.

2 Chứng minh tính đúng đắn

Bổ đề 1: Thao tác không ảnh hưởng các giá trị nhỏ hơn m

Khẳng định: Khi thao tác trên bộ $\{m, m{+}1, m{+}2\}$, mọi giá trị $< m$ không thay đổi.

Chứng minh: Phép cập nhật chỉ tác động đúng ba phần tử có giá trị $m, m{+}1, m{+}2$. Các phần tử khác giữ nguyên, nên mọi giá trị $< m$ không bị thay đổi. □

Bổ đề 2: Nếu $pos(m{+}2)$ đứng trước cả $pos(m)$ và $pos(m{+}1)$ thì luôn nên thao tác

Khẳng định: Nếu $c < a$ và $c < b$ với $a = pos(m)$, $b = pos(m{+}1)$, $c = pos(m{+}2)$, thì việc thao tác sẽ làm hoán vị **nhỏ hơn theo từ điển** so với không thao tác (trong mọi trạng thái hiện tại).

Chứng minh: Chọn $i = c$ (vị trí sớm nhất trong ba vị trí), sau thao tác ta đặt p_i từ $m{+}2$ thành m. Ở vị trí i, giá trị giảm từ $m{+}2$ xuống m là một cải thiện nghiêm ngặt. Các vị trí trước i không đổi. Do đó, theo định nghĩa thứ tự từ điển, mảng sau thao tác nhỏ hơn mảng trước thao tác. □

Bổ đề 3: Quyết định tham lam ở mức m không làm mất tối ưu toàn cục

Khẳng định: Khi xử lý giá trị m, nếu điều kiện $c < a$ và $c < b$ đúng thì mọi nghiệm tối ưu (nhỏ nhất từ điển) đều phải thực hiện thao tác (ít nhất một lần) để đưa m về vị trí c tại thời điểm đó.

Chứng minh: Ở thời điểm đang xét, vị trí c đang chứa $m{+}2$ và c đứng trước a, b. Muốn có hoán vị nhỏ nhất theo từ điển, ta cần tối thiểu hóa phần tử tại vị trí c (vì các vị trí trước c không đổi khi chỉ thao tác trên các số $\geq m$ theo Bổ đề 1). Trong tập $\{m, m{+}1, m{+}2\}$, giá trị nhỏ nhất có thể đặt ở vị trí c chính là m, và thao tác hiện tại đặt đúng m vào c ngay lập tức. Nếu không thực hiện, phần tử tại c sẽ $\geq m{+}1$ (vì m và $m{+}1$ đều đang nằm sau c), khi đó mảng kết quả chắc chắn không thể nhỏ nhất theo từ điển so với phương án có đặt m ở c. □

Định lý: Thuật toán greedy cho ra hoán vị nhỏ nhất theo từ điển đạt được

Chứng minh: Ta xét m tăng dần từ 0 đến $n - 3$. Theo Bổ đề 2 và 3, mỗi khi điều kiện $pos(m{+}2)$ đứng trước cả $pos(m)$ và $pos(m{+}1)$ xảy ra, thao tác là bắt buộc để tối ưu từ điển và không ảnh hưởng các giá trị $< m$ (Bổ đề 1). Do đó, việc áp dụng greedy tại mọi thời điểm có thể đảm bảo các quyết định ở mức giá trị nhỏ được chốt đúng theo tiêu chí từ điển, và không bị "phá" bởi các thao tác xử lý các giá trị lớn hơn. Thuật toán kết thúc khi không còn bộ nào thỏa điều kiện, khi đó không thể cải thiện thêm vị trí sớm nhất của bất kỳ m nào, nên kết quả là nhỏ nhất có thể. □

3 Đánh giá độ phức tạp

Độ phức tạp thời gian

Thuật toán gồm các phần chính sau:

- Khởi tạo mảng vị trí `pos`: duyệt qua hoán vị một lần, tốn
$$\Theta(n).$$
- Vòng lặp chính với biến m chạy trong đoạn $[0, n - 3]$. Mỗi lần lặp:
  - Truy xuất các vị trí `pos[m]`, `pos[m+1]`, `pos[m+2]` trong thời gian $\Theta(1)$.

– Nếu thỏa điều kiện, thực hiện một thao tác cập nhật đúng 3 phần tử và 3 vị trí, tốn $\Theta(1)$.

Trong trường hợp xấu nhất, biến m có thể bị lùi lại nhiều lần, nhưng tổng số lần lặp bị chặn bởi số lần thực hiện thao tác, mà mỗi thao tác tạo ra một cải thiện từ điển hữu hạn. Với $n \leq 2000$, tổng số bước là $\Theta(n^2)$ trong trường hợp tệ nhất.

Kết luận: Độ phức tạp thời gian của thuật toán là

$$\Theta(n^2).$$

Độ phức tạp không gian

Thuật toán sử dụng các cấu trúc dữ liệu sau:

- Mảng hoán vị `p` kích thước n: $\Theta(n)$.
- Mảng vị trí `pos` kích thước n: $\Theta(n)$.

Kết luận: Độ phức tạp không gian của thuật toán là

$$\Theta(n).$$

4 Cài đặt (Implementation)

Cài đặt dưới đây dùng 0-based cho tiện. Mảng $pos[v]$ lưu vị trí hiện tại của giá trị v.

```python
import sys

def solve_one(p):
    n = len(p)
    # convert to 0-based values
    p = [x - 1 for x in p]

    # pos[v] = current index of value v
    pos = [0] * n
    for i, v in enumerate(p):
        pos[v] = i

    m = 0
    while m <= n - 3:
        a = pos[m]
        b = pos[m + 1]
        c = pos[m + 2]

        # condition: value (m+2) must be before both m and (m+1)
```

```
        if c < a and c < b:
            i = c
            j = a if a < b else b
            k = b if a < b else a

            # apply operation on indices (i<j<k)
            p[i] -= 2
            p[j] += 1
            p[k] += 1

            # update positions for affected values
            pos[p[i]] = i
            pos[p[j]] = j
            pos[p[k]] = k

            # step back to re-check neighbors
            m = max(0, m - 2)
        else:
            m += 1

    # back to 1-based
    return [x + 1 for x in p]


def solve():
    data = sys.stdin.read().strip().split()
    t = int(data[0])
    idx = 1
    out = []
    for _ in range(t):
        n = int(data[idx]); idx += 1
        p = list(map(int, data[idx:idx+n])); idx += n
        ans = solve_one(p)
        out.append(" ".join(map(str, ans)))
    print("\n".join(out))


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