# Trận thư hùng ở Namek

Nhóm 6
Lê Quang Trung - Nguyễn Hồng Phúc

## 1 Tóm tắt bài toán

Goku đang chuẩn bị cho trận chiến quyết định với Frieza trên hành tinh Namek. Để tăng cơ hội chiến thắng, anh có danh sách $n$ chiến binh Trái Đất (với $1 \leq n \leq 1000$). Mỗi chiến binh được mô tả bằng bộ ba số nguyên $(a_i, b_i, c_i)$ lần lượt biểu diễn *tuổi* (*age*), *độ thông minh* (*intelligence*) và *sức mạnh chiến đấu* (*power*). Các chỉ số này có thể biến thiên rộng: tuổi và độ thông minh nằm trong khoảng $[1, 100000]$, trong khi sức mạnh có thể âm, dương hoặc bằng không trong đoạn $[-200, 200]$.

## 2 Ràng buộc tuyển chọn

Goku chỉ muốn tập hợp một đội hình mà ở đó các chiến binh có thể "xếp tầng" theo thứ bậc về tuổi và độ thông minh, tránh việc một người vượt trội hoàn toàn so với người khác. Cụ thể, với mọi cặp chiến binh được chọn, ít nhất một trong hai điều kiện sau phải đúng:

- Chiến binh thứ nhất có tuổi không lớn hơn chiến binh thứ hai *và* độ thông minh không vượt quá;
- Hoặc chiều ngược lại: chiến binh thứ hai không lớn tuổi hơn *và* không thông minh hơn chiến binh thứ nhất.

Cách diễn đạt tương đương: nếu sắp xếp đội hình được chọn theo tuổi tăng dần (cho phép bằng nhau) thì dãy độ thông minh tương ứng cũng phải là một dãy không giảm. Như vậy, tập chiến binh được chọn phải tạo thành một chuỗi "tăng dần" đồng thời ở cả hai thuộc tính *age* và *intelligence*.

## 3 Mục tiêu tối ưu

Sau khi đảm bảo điều kiện sắp xếp nói trên, Goku muốn tổng năng lượng hợp lực là lớn nhất. Chỉ số đo lường được dùng là tích các giá trị *power* của mọi chiến binh trong đội hình được chọn. Vì có thể tồn tại chiến binh mang *power* âm, việc lựa chọn thứ tự và số lượng thành viên rất quan trọng:

- Tích nhiều số âm có thể tạo ra kết quả dương lớn nếu số lượng là chẵn;
- Ngược lại, việc chọn lẻ số chiến binh có *power* âm có thể làm doanh trại suy yếu đáng kể.

Nhiệm vụ là tìm giá trị tối đa của tích *power* sao cho đội hình vẫn thỏa điều kiện sắp xếp.

# 4 Định dạng vào/ra

## 4.1 Input

- Dòng đầu: số nguyên $n$ — số lượng chiến binh.
- $n$ dòng tiếp theo: mỗi dòng chứa ba số nguyên $a_i, b_i, c_i$ ứng với *age*, *intelligence*, *power* của chiến binh thứ $i$.

## 4.2 Output

- Một số nguyên duy nhất: tích *power* lớn nhất đạt được của một tập chiến binh thỏa các ràng buộc.

# 5 Các lưu ý quan trọng

- Các chiến binh có thể có cùng tuổi hoặc cùng độ thông minh; điều quan trọng là vẫn vạch được thứ tự để tạo ra một dãy không giảm ở cả hai chỉ số.
- Không bắt buộc phải chọn tất cả $n$ chiến binh; có thể chọn bất kỳ số lượng nào (kể cả một người duy nhất) miễn tuân thủ điều kiện sắp xếp.
- Vì *power* có thể mang giá trị âm, kết quả lớn nhất đôi khi đạt được bằng cách loại bỏ một vài chiến binh dù họ có chỉ số tuổi và trí tuệ phù hợp.

# 6 Hướng giải quyết

## 6.1 Tiền xử lý và sắp xếp.

Đầu tiên, gom ba chỉ số của từng chiến binh thành cặp khóa $(a_i, b_i)$ và giá trị $c_i$. Sắp xếp toàn bộ danh sách chiến binh theo trật tự từ điển của bộ $(a_i, b_i)$. Sau bước này, mọi phần tử đứng sau trong danh sách đều có tuổi không nhỏ hơn phần tử đứng trước, vì vậy khi duyệt từ trái sang phải chúng ta chỉ cần kiểm tra thêm điều kiện về *intelligence*.

Sau khi sắp xếp, yêu cầu "mọi cặp chiến binh trong đội hình đều có thể xếp thứ tự tăng dần theo tuổi và trí tuệ" biến thành bài toán tìm *chuỗi con* trong dãy đã sắp xếp sao cho cột *intelligence* không giảm. Nói cách khác, ta đang tìm một chuỗi tăng theo *intelligence* (dưới điều kiện đã cố định thứ tự theo *age*) mà tích *power* của chuỗi đó là lớn nhất. Toàn bộ chiến lược tiếp theo sẽ khai thác cấu trúc "chuỗi tăng" này.

## 6.2 Áp dụng cho Subtask 1 ($n \leq 20$).

Quy mô nhỏ cho phép duyệt toàn bộ các cách chọn chiến binh khả dĩ:

- Làm việc trên danh sách đã sắp xếp để đảm bảo điều kiện về *age*.
- Duyệt theo kiểu vét cạn: thử mọi tập con (tối đa $2^n$ trường hợp) hoặc dùng quay lui/DFS. Khi chọn thêm một chiến binh, chỉ giữ lại lựa chọn nếu chỉ số *intelligence* của người mới không nhỏ hơn người cuối cùng trong chuỗi hiện tại.
- Với mỗi chuỗi hợp lệ, tính tích *power*. Cập nhật kết quả lớn nhất bắt gặp.

- Có thể cải thiện bằng cách vừa duyệt vừa nhân giá trị *power*, và khi gặp tích tạm thời $= 0$ hoặc vượt ngưỡng lớn nhất hiện tại thì cắt tỉa nhánh.

Nhờ ràng buộc $n \le 20$, chi phí thời gian $O(2^n)$ (hoặc $O(n \cdot 2^n)$ nếu tính cả việc nhân tích trong mỗi nhánh) vẫn chấp nhận được. Cụ thể, mỗi chiến binh được quyết định "chọn" hoặc "bỏ", nên cây tìm kiếm có tối đa $2^n$ lá; mỗi lá ứng với một tập con và việc kiểm tra điều kiện cùng nhân tích chỉ mất thêm hệ số tuyến tính theo kích thước chuỗi. Về bộ nhớ, ta chỉ cần theo dõi chuỗi đang xây dựng trong ngăn xếp đệ quy (độ sâu tối đa $n$) và vài biến tích luỹ, do đó dung lượng sử dụng tỉ lệ với $n$, tức $O(n)$.

## 6.3 Bàn về Subtask 2 ($n \le 1000$).

Khi $n$ lên tới 1000, chiến lược vét cạn của subtask 1 không còn hiệu dụng: không gian lựa chọn $\mathcal{P}(\{1, \ldots, n\})$ có kích thước $2^n$ là bất khả thi. Tuy nhiên, sau bước sắp xếp chúng ta đã chuyển bài toán về chuỗi

$$\mathcal{S} = \big((a_1, b_1, c_1), (a_2, b_2, c_2), \ldots, (a_n, b_n, c_n)\big)$$

với $a_1 \le a_2 \le \cdots \le a_n$ và nếu $a_i = a_{i+1}$ thì $b_i \le b_{i+1}$. Mục tiêu là chọn một dãy con $\mathcal{T}$ của $\mathcal{S}$ sao cho $b$ không giảm và giá trị $\prod_{(a_k, b_k, c_k) \in \mathcal{T}} c_k$ là lớn nhất. Đây là biến thể có trọng số (theo phép nhân) của bài toán "longest non-decreasing subsequence" quen thuộc.

## 6.4 Ý tưởng DP hai nhánh.

Đối với mỗi vị trí $i$, ta muốn biết những tích tốt nhất khi dừng lại ở phần tử thứ $i$. Do $c_i$ có thể âm, việc chỉ lưu "tích lớn nhất" là không đủ: một tích âm nhỏ nhất (tức trị tuyệt đối lớn nhất) có thể trở thành đáp án tối ưu khi nhân với một giá trị âm khác ở phía sau. Vì vậy ta lưu đồng thời hai đại lượng

$$\mathrm{MaxProd}_i = \max \left\{ \prod_{k=1}^{\ell} c_{t_k} \;\middle|\; \begin{array}{l} t_1 < \cdots < t_\ell = i, \\ b_{t_k} \le b_{t_{k+1}} \text{ for all } k \end{array} \right\},$$

$$\mathrm{MinProd}_i = \min \left\{ \prod_{k=1}^{\ell} c_{t_k} \;\middle|\; \begin{array}{l} t_1 < \cdots < t_\ell = i, \\ b_{t_k} \le b_{t_{k+1}} \text{ for all } k \end{array} \right\}.$$

Hai biến này tương ứng với mảng `max_product` và `min_product` trong phần cài đặt Python ở dưới. Chúng cho phép ta theo dõi đồng thời cả hai khả năng "tích lớn nhất" và "tích nhỏ nhất" để kịp thời lật dấu khi gặp hệ số âm.

## 6.5 Công thức truy hồi.

Khởi tạo với $\mathrm{MaxProd}_i = \mathrm{MinProd}_i = c_i$ vì chuỗi chỉ chứa riêng chiến binh $i$ luôn hợp lệ. Khi xét một vị trí $i$, với mỗi $j < i$ thoả $b_j \le b_i$, ta có thể nối chiến binh $i$ vào chuỗi tốt nhất hoặc tệ nhất hiện có ở $j$:

$$\widehat{P}_{ij} = \mathrm{MaxProd}_j \cdot c_i, \qquad \widehat{Q}_{ij} = \mathrm{MinProd}_j \cdot c_i.$$

Sau đó cập nhật

$$\mathrm{MaxProd}_i = \max\big(\mathrm{MaxProd}_i,\, \widehat{P}_{ij},\, \widehat{Q}_{ij}\big), \qquad \mathrm{MinProd}_i = \min\big(\mathrm{MinProd}_i,\, \widehat{P}_{ij},\, \widehat{Q}_{ij}\big).$$

Nhờ lưu cả $\mathrm{MinProd}_j$, khi $c_i < 0$ ta có thể chuyển một tích âm lớn (theo trị tuyệt đối) thành tích dương vượt trội tại bước $i$.

## 6.6 Các bước thực hiện chi tiết.

1. Chuẩn hoá đầu vào: xây dựng danh sách $\mathcal{S}$ gồm các bộ $((a_i, b_i), c_i)$.
2. Sắp xếp $\mathcal{S}$ theo thứ tự từ điển $(a_i, b_i)$.
3. Khởi tạo hai mảng `max_product`$[1..n]$ và `min_product`$[1..n]$ với giá trị ban đầu $c_i$.
4. Với từng $i$ từ 1 đến $n$:
   (a) Duyệt tất cả $j < i$. Nếu $b_j \le b_i$, tính hai tích ứng viên $\widehat{P}_{ij}$ và $\widehat{Q}_{ij}$ như trên.
   (b) Lần lượt so sánh $\widehat{P}_{ij}$ và $\widehat{Q}_{ij}$ với giá trị hiện tại ở `max_product`$[i]$ và `min_product`$[i]$, cập nhật chúng: `max_product`$[i] = \max($`max_product`$[i], \widehat{P}_{ij}, \widehat{Q}_{ij})$, `min_product`$[i] = \min($`min_product`$[i], \widehat{P}_{ij}, \widehat{Q}_{ij})$.
5. Kết quả cuối cùng là $\max\limits_{1 \le i \le n} \text{MaxProd}_i$.

Độ phức tạp thời gian là $O(n^2)$ vì vòng lặp chính qua $i$ kết hợp với vòng lặp qua mọi $j < i$ tạo thành đúng $\frac{n(n-1)}{2}$ cặp cần xét; trên mỗi cặp ta chỉ thực hiện hằng số phép tính (hai phép nhân, vài phép so sánh), không có chi phí ẩn khác. Về không gian, ta duy trì hai mảng `max_product` và `min_product` độ dài $n$, cộng thêm các biến tạm như $\widehat{P}_{ij}$, do đó lượng bộ nhớ tăng tuyến tính theo $n$ và được mô tả bởi $O(n)$. Đây là chi phí nhỏ so với giới hạn của subtask 2 nhưng vẫn đủ để tái sử dụng giá trị giữa các bước.

## 6.7 Cài đặt Python cho Subtask 2.

Đoạn mã dưới đây hiện thực hoá các bước trên, đặt tên biến rõ ràng và bổ sung chú thích cho từng dòng (trừ phần nhập dữ liệu theo yêu cầu):

```
import sys

read_line = sys.stdin.readline

n = int(read_line())

warriors = []  # lưu các chiến binh dưới dạng ((age, intelligence), power)
for _ in range(n):
    age, intelligence, power = map(int, read_line().split())
    warriors.append(((age, intelligence), power))

warriors.sort()  # sắp xếp theo (age, intelligence) để cố định thứ tự theo tuổi

max_product = [0] * n  # max_product[i] là tích lớn nhất kết thúc tại chiến binh i
min_product = [0] * n  # min_product[i] là tích nhỏ nhất kết thúc tại i
best_overall = float('-inf')  # đáp án tốt nhất tìm được

for i, ((age_i, intel_i), power_i) in enumerate(warriors):
    max_product[i] = power_i  # khởi tạo với chuỗi chỉ gồm chính chiến binh i
    min_product[i] = power_i  # tương tự cho tích nhỏ nhất
```

```
    for j in range(i):  # xét mọi chiến binh đứng trước để nối chuỗi
        (_, intel_j), _ = warriors[j]  # chỉ cần chỉ số intelligence của chiến binh j

        if intel_j <= intel_i:  # chỉ nối khi dãy intelligence vẫn không giảm
            product_from_max = max_product[j] * power_i  # nối từ tích lớn nhất ở j
            product_from_min = min_product[j] * power_i  # hoặc nối từ tích nhỏ nhất

            if product_from_max > max_product[i]:  # cập nhật tích lớn nhất tại i
                max_product[i] = product_from_max
            if product_from_min > max_product[i]:  # tích âm có thể thành lớn nhất
                max_product[i] = product_from_min

            if product_from_max < min_product[i]:  # cập nhật tích nhỏ nhất tại i
                min_product[i] = product_from_max
            if product_from_min < min_product[i]:  # xét cả trường hợp nối từ min
                min_product[i] = product_from_min

    if max_product[i] > best_overall:  # ghi nhận đáp án tốt nhất sau khi xử lý i
        best_overall = max_product[i]

print(best_overall)  # in ra tích power lớn nhất tìm được
```