Lời giải nhóm 10

---

1. Mô tả thuật toán

Subtask 1 ($n \leq 20$)

- **Thuật toán sử dụng: Thử-Sai (Brute Force) bằng phương pháp Quay lui (Backtracking) / Đệ quy**.
- **Lý do:** Với $n \leq 20$, số lượng chiến binh là rất nhỏ. Tổng số tập con có thể chọn là 2^n. Với $n = 20$, $2^{20} \approx 1$ triệu, là một số lượng phép toán đủ nhỏ để máy tính thực hiện trong thời gian cho phép.
- **Các bước thực hiện chi tiết:**
  1. **Sắp xếp:** Đầu tiên, sắp xếp n chiến binh theo `Age` tăng dần. Nếu `Age` bằng nhau, sắp xếp tiếp theo `Intelligence` tăng dần.
  2. **Hàm đệ quy:** Xây dựng một hàm đệ quy `try(index, last_selected_warrior, current_product)`:
     - `index`: Chỉ số của chiến binh đang xét (từ 0 đến n).
     - `last_selected_warrior`: Chiến binh cuối cùng được thêm vào nhóm (dùng để kiểm tra điều kiện).
     - `current_product`: Tích `Power` của nhóm hiện tại.
  3. **Lưu kết quả:** Dùng một biến toàn cục `max_product` (khởi tạo bằng giá trị âm vô cùng, hoặc một giá trị rất nhỏ) để lưu tích lớn nhất tìm được. Mỗi khi hàm đệ quy chạy, cập nhật `max_product = max(max_product, current_product)`.
  4. **Điều kiện dừng:** Khi `index == n` (đã xét hết các chiến binh), ta dừng nhánh đệ quy này.
  5. **Hai lựa chọn:** Tại mỗi chiến binh `i` (index hiện tại):
     - **Lựa chọn 1 (Không chọn `i`):** Gọi đệ quy `try(i + 1, last_selected_warrior, current_product)`.
     - **Lựa chọn 2 (Chọn `i`):** Chỉ thực hiện nếu `i` thỏa mãn điều kiện9: `warrior[i].intelligence >= last_selected_warrior.intelligence`. (Điều kiện `Age` đã được đảm bảo do ta sắp xếp từ bước 1).
       - Nếu chọn được, gọi đệ quy `try(i + 1, warrior[i], current_product * warrior[i].power)`.

Subtask 2 ($n \leq 1000$)

- **Áp dụng thuật toán Subtask 1: Không thể áp dụng**.
- **Giải thích:** Với $n = 1000$, độ phức tạp $O(2^n)$ (như của Subtask 1) sẽ là 2^{1000}, một con số khổng lồ không thể tưởng tượng được. Máy tính sẽ mất hàng tỷ tỷ năm để chạy và chắc chắn sẽ bị Chạy quá thời gian (Time Limit Exceeded).
- **Phương pháp khác: Quy hoạch động (Dynamic Programming - DP)**.
- **Giải thích (Tại sao DP?):**
  - Đây là bài toán tìm một **dãy con** (subsequence) thỏa mãn một điều kiện (giống bài toán Dãy con tăng dài nhất - LIS).
  - Bài toán có "cấu trúc con tối ưu": Tích lớn nhất của một dãy kết thúc tại chiến binh `i` có thể được xây dựng dựa trên các tích lớn nhất (hoặc nhỏ nhất) của các dãy kết thúc tại chiến binh `j` (với $j < i$ và thỏa mãn điều kiện).
  - **Thách thức:** Mục tiêu là **tích** (product) và `Power` có thể là số **âm**. Một tích âm nhỏ nhất nhân với một số âm sẽ trở thành tích dương lớn nhất. Do đó, tại mỗi bước, chúng ta phải theo dõi cả **tích dương lớn nhất** và **tích âm nhỏ nhất**.
- **Các bước thực hiện (Thuật toán DP):**
  1. **Lưu trữ:** Đọc `n` chiến binh vào một danh sách (ví dụ: `warriors`), mỗi phần tử là một `tuple (age, intelligence, power)`.
  2. **Sắp xếp:** Sắp xếp danh sách `warriors` theo `age` (khóa 1) và `intelligence` (khóa 2) đều tăng dần.
  3. **Khởi tạo DP:** Tạo 2 mảng `dp_max` và `dp_min` có kích thước n:
     - `dp_max[i]`: Lưu tích `Power` **lớn nhất** (dương nhất) của một dãy hợp lệ kết thúc tại chiến binh `i`.
     - `dp_min[i]`: Lưu tích `Power` **nhỏ nhất** (âm nhất) của một dãy hợp lệ kết thúc tại chiến binh `i`.
  4. **Lặp (Xây dựng bảng DP):**
     - Lặp `i` từ 0 đến $n-1$ (xét từng chiến binh `i` làm điểm kết thúc):
       - Lấy `current_power = warriors[i].power`.
       - Khởi tạo `dp_max[i] = current_power` và `dp_min[i] = current_power` (trường hợp dãy chỉ có mình chiến binh `i`).
       - Lặp `j` từ 0 đến $i-1$ (xét tất cả chiến binh `j` đứng trước `i`):
         - **Kiểm tra điều kiện:** Nếu `warriors[j].intelligence <= warriors[i].intelligence` (điều kiện `Age` đã được đảm bảo do sắp xếp).
         - **Tính toán các tích ứng viên:**
           - `prod1 = dp_max[j] * current_power`
           - `prod2 = dp_min[j] * current_power`
         - **Cập nhật:**

- `dp_max[i] = max(dp_max[i], prod1, prod2)` (Tìm giá trị dương lớn nhất có thể)
- `dp_min[i] = min(dp_min[i], prod1, prod2)` (Tìm giá trị âm nhỏ nhất có thể)

5. **Tìm kết quả:** Kết quả cuối cùng là giá trị lớn nhất trong toàn bộ mảng `dp_max`. (Lưu ý: Phải lấy `max` của *cả* mảng, vì dãy có tích lớn nhất không nhất thiết phải kết thúc ở chiến binh cuối cùng).

---

2. Cài đặt thuật toán (Subtask 2)

Python

```
import sys

def solve():


    # Đọc số lượng chiến binh n
    try:
        n = int(sys.stdin.readline())
        if n == 0:
            print(0) # Trường hợp không có chiến binh nào
            return

        warriors = [] # Danh sách lưu các chiến binh

        # Đọc thông tin của n chiến binh
        for _ in range(n):
            # Đọc age, intelligence, power
            a, b, c = map(int, sys.stdin.readline().split())
            warriors.append((a, b, c)) # Lưu dưới dạng tuple

        # Bước 1: Sắp xếp các chiến binh
        # Sắp xếp theo Age tăng dần (khóa 1), sau đó theo Intelligence tăng
dần (khóa 2)
        warriors.sort(key=lambda x: (x[0], x[1]))

        # Bước 2: Khởi tạo mảng DP
        # dp_max[i] = Tích power LỚN NHẤT của dãy hợp lệ kết thúc tại i
        # dp_min[i] = Tích power NHỎ NHẤT của dãy hợp lệ kết thúc tại i
        dp_max = [0] * n
        dp_min = [0] * n
```

```
        # Khởi tạo kết quả cuối cùng với một giá trị rất nhỏ
        # (Vì kết quả có thể là số âm)
        max_product = -float('inf')

        # Bước 3: Xây dựng bảng quy hoạch động
        for i in range(n):
            # Lấy power của chiến binh hiện tại
            current_power = warriors[i][2]

            # Khởi tạo: Tối thiểu, dãy chỉ bao gồm chính chiến binh i
            dp_max[i] = current_power
            dp_min[i] = current_power

            # Lặp qua các chiến binh j đứng trước i
            for j in range(i):
                # Kiểm tra điều kiện:
                # 1. warriors[j][0] <= warriors[i][0] (Luôn đúng do đã sắp
xếp)
                # 2. warriors[j][1] <= warriors[i][1] (chỉ số intelligence)
                if warriors[j][1] <= warriors[i][1]:

                    # Tính 2 ứng viên tích mới khi kết hợp i với dãy kết thúc
tại j
                    # Ứng viên 1: Lấy tích max tại j * power của i
                    prod1 = dp_max[j] * current_power
                    # Ứng viên 2: Lấy tích min tại j * power của i
                    prod2 = dp_min[j] * current_power

                    # Cập nhật dp_max[i] (tìm giá trị dương lớn nhất)
                    dp_max[i] = max(dp_max[i], prod1, prod2)
                    # Cập nhật dp_min[i] (tìm giá trị âm nhỏ nhất)
                    dp_min[i] = min(dp_min[i], prod1, prod2)

            # Cập nhật kết quả chung sau mỗi lần tính xong dp[i]
            max_product = max(max_product, dp_max[i])

        # Bước 4: In kết quả
        print(max_product)

    except EOFError:
        pass
    except Exception as e:
        # Xử lý các lỗi có thể xảy ra (ví dụ: đầu vào trống)
        pass
```

3. Phân tích độ phức tạp

Subtask 1 (Brute Force / Quay lui)

- **Độ phức tạp thời gian (Time Complexity):** $O(n \log n + 2^n)$
  - $O(n \log n)$ cho bước sắp xếp ban đầu.
  - $O(2^n)$ cho việc quay lui, vì tại mỗi chiến binh ta có 2 lựa chọn (chọn hoặc không chọn).
  - Tổng cộng: $O(2^n)$ (vì 2^n lớn hơn $n \log n$).
- **Độ phức tạp không gian (Space Complexity):** $O(n)$
  - $O(n)$ để lưu danh sách n chiến binh.
  - $O(n)$ cho độ sâu tối đa của ngăn xếp (stack) đệ quy.

Subtask 2 (Dynamic Programming)

- **Độ phức tạp thời gian (Time Complexity):** $O(n^2)$
  - $O(n \log n)$ cho bước sắp xếp `warriors.sort()`.
  - $O(n^2)$ cho hai vòng lặp lồng nhau (vòng lặp `i` bên ngoài và vòng lặp `j` bên trong) để xây dựng bảng DP.
  - Tổng cộng: $O(n \log n + n^2) = O(n^2)$ (vì n^2 lớn hơn $n \log n$ khi n lớn).
- **Độ phức tạp không gian (Space Complexity):** $O(n)$
  - $O(n)$ để lưu danh sách `warriors`.
  - $O(n)$ cho mảng `dp_max`.
  - $O(n)$ cho mảng `dp_min`.
  - Tổng cộng: $O(n)$.