CHIA ĐỘI

Kì thi Hackathon CSP sắp tới, Ban tổ chức muốn chia các thí sinh thành các đội chơi theo nguyện vọng của chính các thí sinh. Biết rằng, có n thí sinh tham gia cuộc thi. Thí sinh thứ i mong muốn đội của mình không ít hơn a_i người.

Để cuộc thi thêm nhiều gay cấn, Ban tổ chức mong muốn chia n thí sinh thành nhiều nhất các đội sao cho:

- Mỗi thí sinh chỉ thuộc một đội.
- Số lượng thí sinh thuộc mỗi đội không nhỏ hơn nguyện vọng của từng thí sinh thuộc đội đó
- Số lượng đội là nhiều nhất có thể. Nếu có nhiều phương án chia thành nhiều đội nhất, hãy chọn cách chia để số thí sinh thuộc đội nhiều nhất là ít nhất có thể.

Yêu cầu: Biết nguyện vọng của từng thí sinh, hãy cho biết Ban tổ chức có thể chia các bạn thí sinh thành tối đa bao nhiêu đội.

Dữ liệu: Vào từ file văn bản TEAMS.INP gồm:

- Dòng đầu ghi số nguyên dương n ($1 \le n \le 10^6$).
- Dòng tiếp theo ghi n số nguyên dương $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) là nguyện vọng của từng thí sinh

Kết quả: Ghi ra file văn bản TEAMS.OUT gồm:

- Dòng đầu tiên ghi số nguyên dương k – số đội lớn nhất có thể chia được thỏa mãn nguyện vọng của tất cả các thí sinh
- k dòng tiếp theo, mỗi dòng cho biết một đội: Đầu dòng là số lượng thí sinh trong đội, tiếp theo là các chỉ số của các thí sinh trong đội.

Ví dụ:

TEAMS.INP	TEAMS.OUT
5 2 1 2 2 3	2 2 4 2 3 5 1 3

Bộ test chia làm 2 subtasks

Subtask 1 (50% số điểm): $n \le 5000$

Subtask 2 (50% số điểm): Không có ràng buộc bổ sung

Thuật toán

Bằng một phép sắp xếp, ta có thể coi dãy $a_{1\dots n}$ được sắp giảm dần:

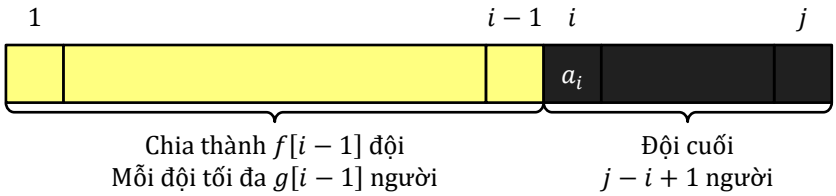
$$a_1 \ge a_2 \ge \cdots \ge a_n$$

Chú ý rằng các giá trị trong mảng $a[1 \dots n]$ đều là số nguyên $\in \{1 \dots n\}$ nên để sắp xếp ta có thể sử dụng thuật toán đếm phân phối mất thời gian $O(n)$.

Nhận xét 1: Trong các phương án chia đội tối ưu, tồn tại phương án mà mỗi nhóm gồm những thí sinh liên tiếp. Thật vậy, với một cách chia đội bất kỳ, xét đội chứa thí sinh 1, nếu đội này có thí sinh $i+1$ mà không có thí sinh i, ta có thể yêu cầu hai thí sinh i và $i+1$ đổi đội, khi ấy thí sinh i được đưa vào đội có ít nhất a_1 người còn thí sinh $i+1$ được đưa vào đội có ít nhất a_i người. Do $a_1 \ge a_i$ và $a_i \ge a_{i+1}$ nên yêu cầu của mọi thí sinh vẫn được thỏa mãn. Cứ làm như vậy cho tới khi đội có thí sinh 1 gồm các thí sinh liên tiếp (chẳng hạn từ 1 tới k), ta tiếp tục phương pháp tương tự với đội có thí sinh $k+1$... Như vậy ta chỉ cần quan tâm tới phương án chia đội mà mỗi đội gồm những thí sinh liên tiếp.

Xét dãy thí sinh từ 1 tới j, gọi $f[j]$ là số đội cực đại có thể thành lập từ những thí sinh này và $g[j]$ là giới hạn cực tiểu sao cho có thể thành lập $f[j]$ đội mà số thí sinh trong mỗi đội không vượt quá $g[j]$ (Nếu không có cách chia đội cho các thí sinh này coi như $f[j] = -\infty$ và $g[j]$ vô nghĩa). Ta cần xây dựng công thức truy hồi tính các $f[j]$ và $g[j]$ với $\forall j: a_1 < j \leq n$.

Nếu ta biết được trong phương án tối ưu, đội cuối cùng chứa các thí sinh $i \dots j$ thì vấn đề trở thành chia tối ưu trên dãy các thí sinh từ 1 tới $i - 1$.



Tức là:

$$\begin{cases} f[j] = f[i-1] + 1 \\ g[j] = \max(g[i-1], j - i + 1) \end{cases}$$

Vậy để tính $f[j]$, ta xét mọi chỉ số i thỏa mãn $1 \leq i \leq j$ và $a[i] \leq j - i + 1$ (gọi chỉ số i này là điểm cắt). Với mỗi điểm cắt i thì $f[j]$ được cực đại hóa theo $f[i-1] + 1$. Để tính $g[j]$, ta cũng xét mọi điểm cắt $i \leq j$ như trên, bổ sung thêm điều kiện $f[j] = f[i-1] + 1$ rồi cực tiểu hóa $g[j]$ theo $\max(g[i-1], j - i + 1)$.

Cơ sở: $f[0] = 0$; $g[0] = 0$; $f[a_1] = 1$; $g[a_1] = a_1$ còn $\forall j: 0 < j < a_1$ thì $f[j] = -\infty$. Công thức truy hồi được dùng để tính lần lượt các $f[j]$ và $g[j]$ với $j \in \{a_1 + 1 \dots n\}$. Thuật toán $O(n^2)$ không có gì khó khăn.

Thuật toán $O(n)$ đòi hỏi có thêm những nhận xét tinh tế. Ta sẽ tách riêng hai công đoạn tính hàm mục tiêu f và hàm mục tiêu g.

Tính f:

Nhận xét 2: với $\forall j > a_1$ thì:

$$f[j-1] \leq f[j] \leq f[j-1] + 1$$

Tức là nếu bổ sung thêm thí sinh j vào tập thí sinh $\{1,2, \dots, j-1\}$, số đội trong phương án tối ưu không giảm đi nhưng cùng lắm chỉ tăng lên 1.

Thật vậy, với một phương án chia đội cho những thí sinh từ 1 tới $j - 1$, ta chỉ cần bổ sung thí sinh j vào đội cuối là được một phương án chia đội cho những thí sinh từ 1 tới j, suy ra $f[j] \geq f[j-1]$.

Ngược lại, với một phương án chia ≥ 2 đội cho những thí sinh từ 1 tới j, ta chỉ cần nhập 2 đội cuối lại và loại bỏ thí sinh j là được một phương án chia đội cho những thí sinh từ 1 tới $j - 1$, suy ra $f[j-1] \geq f[j] - 1$.

Bây giờ làm thế nào để kiểm tra $f[j]$ có phải bằng $f[j-1] + 1$ hay không?

Xét phương án tối ưu chia đội cho những thí sinh $1 \dots j$, giả sử đội cuối gồm những thí sinh $i \dots j$, khi đó có điều kiện $a_i \leq j - i + 1$. Ta sẽ chỉ ra rằng nếu $f[j] = f[j-1] + 1$ thì bắt buộc $a_i = j - i + 1$, bởi nếu $a_i < j - i + 1$ ta chỉ cần loại bỏ thí sinh j khỏi đội cuối là được phương án chia đội cho $j - 1$ thí sinh đầu tiên với số đội bằng $f[j]$ (lớn hơn $f[j-1]$). Mâu thuẫn.

Điều kiện $a_i = j - i + 1$ có thể viết thành $a_i + i - 1 = j$. Thuật toán $O(n)$ tính hàm mục tiêu f như sau: Với mỗi chỉ số i, ta đưa nó vào danh sách $L[a_i + i - 1]$. Để tính $f[j]$, trước tiên ta đặt $f[j] = f[j-1]$, sau đó thay vì phải xét mọi điểm cắt i đứng trước chỉ số j, ta chỉ cần xét mọi điểm cắt i trong danh sách $L[j]$ mà thôi. Tổng kích thước các danh sách $L[.]$ là n phần tử và mỗi danh sách chỉ phải duyệt qua 1 lần khi tính hàm mục tiêu f nên độ phức tạp tính toán là $O(n)$.

Tính g:

Dựa vào những lập luận trong thuật toán tính $f[.]$, nếu $f[j] = f[j-1] + 1$ thì để tính $f[j]$ ta chỉ cần xét các điểm cắt $i \in L[j]$ có $f[i-1] = f[j] - 1$ rồi cực tiểu hóa $g[j]$ theo $\max\left(g[i-1], \underbrace{j-i+1}_{=a_i}\right)$

Trường hợp $f[j] = f[j-1]$, xét phương án tối ưu chia các thí sinh $1 \dots j$ thành $f[j]$ đội (tối ưu theo nghĩa cho $g[j]$ nhỏ nhất). Phương án này có thể thuộc một trong hai loại:

Loại 1: Đội cuối là "chặt", có nghĩa là nếu đội cuối gồm các thí sinh $i \dots j$ và ta bỏ thí sinh j ra khỏi đội cuối thì sẽ vi phạm yêu cầu của thí sinh i. Điều này có nghĩa là $a_i = j - i + 1$ hay $i \in L[j]$. Như vậy nếu phương án tối ưu rơi vào loại 1, nó có thể được xác định bằng cách xét mọi điểm cắt $i \in L[j]$, chọn ra điểm cắt i thỏa mãn hai điều kiện:

$$\begin{cases} f[i-1] = f[j] - 1 \\ \max(g[i-1], a_i) \text{ nhỏ nhất có thể} \end{cases}$$

Loại 2: Đội cuối là không "chặt" tức là nếu đội cuối gồm các thí sinh $i \dots j$ và ta bỏ thí sinh j ra khỏi đội cuối thì được một phương án chia đội cho các thí sinh $1 \dots j-1$. Điều này cho thấy $g[j] \geq g[j-1]$.

- Nếu $f[j-1] \times g[j-1] = j-1$, tức là $j-1$ thí sinh đầu tiên được chia tối ưu thành $f[j-1]$ đội với số thành viên bằng nhau và bằng $g[j-1]$. Khi thêm thí sinh j vào, $g[j]$ chắc chắn $> g[j-1]$, vậy ta thêm thí sinh j vào đội cuối và thu được $g[j] = g[j-1] + 1$. Vì $g[j] > g[j-1]$ nên $g[j] = g[j-1] + 1$ đảm bảo là giá trị $g[j]$ tối ưu.
- Nếu không, tức là $j-1$ người đầu tiên được chia tối ưu thành $f[j-1]$ đội với số thành viên không hoàn toàn giống nhau, trong trường hợp này ta có thể thêm người j vào đội có ít thành viên nhất và làm cho $g[j] = g[j-1]$. Vì $g[j] \geq g[j-1]$ nên $g[j] = g[j-1]$ đảm bảo là giá trị $g[j]$ tối ưu. (Việc phân đội cho thí sinh j thế này có thể khiến cho các đội không gồm các thí sinh liên tiếp nhưng ta đã lập luận rằng sẽ có phương án chia đội gồm các thí sinh liên tiếp tương đương)

Việc tìm $g[j]$ tốt nhất trong các phương án loại 1 mất thời gian $O(|L[j]|)$ còn tìm $g[j]$ tốt nhất trong các phương án loại 2 mất thời gian $O(1)$. Độ phức tạp tính toán của thuật toán tính $g[.]$ là $O(n)$

```
«Tạo các danh sách L[1…n]»;
//Tính f
f[0] = 0; f[1…a[1] – 1] = -∞; f[a[1]] = 1;
for (j = a[1] + 1; j <= n; ++j)
{
    f[j] = f[j - 1];
    for (i: L[j])
        «Cực đại hóa f[j] theo f[i – 1] + 1»;
}
//Tính g
for (j = a[1] + 1; j <= n; ++j)
{
    g[j] = -∞;
    if (f[j] > f[j - 1])
    {
        for (∀i ∈ L[j]: f[j] == f[i - 1] + 1)
           «Cực tiểu hóa g[j] theo max(g[i - 1], j - i + 1)»;
    }
    else //f[j] == f[j - 1]
    {
        //Cực tiểu hóa g[j] theo các phương án loại 1
        if (g[j - 1] * f[j - 1] == j - 1)
            g[j] = g[j - 1] + 1;
        else
            g[j] = g[j - 1];
        //Cực tiểu hóa g[j] theo các phương án loại 2
        for (∀i ∈ L[j]: f[j] == f[i - 1] + 1)
            «Cực tiểu hóa g[j] theo max(g[i - 1], j - i + 1))»;
    }
}
```