Lời giải [IT003.P21.CTTN.1] Assigment2 (Advance và Basic)

Bảo Quý Định Tân - 24520028

Ngày 18 tháng 3 năm 2025

Mục lục

1 Advance

1.1 MaxMinSum

Ta để ý mỗi phần tử trong mảng sẽ có lúc được làm **max** trong một bộ k số nào đó, hoặc làm **min** trong bộ k số khác. Tiếp cận theo hướng này, sắp xếp lại mảng theo thứ tự tăng dần. Với phần tử i:

- Số bộ k mà a_i là **max**: $\binom{i-1}{k-1}$
- Số bộ k mà a_i là **min**: $\binom{n-i}{k-1}$

Ta có thể tính nhanh C_n^k bằng cách chuẩn bị trước mảng $fact[i] = i!\ \%\ mod$ và $iFact[i] = \dfrac{1}{i!}\ \%\ mod$.

1.2 khangtd.Login1 và khangtd.Login2

Có nhiều cách để cài đặt bài này, với giới hạn của đề bài, ta có thể sử dụng cấu trúc dữ liệu **map** trong **c++** để cài đặt dễ dàng.

1.3 khangtd.DetectVirus và khangtd.DetectVirus2

Ở phiên bản dễ, ta hoàn toàn có thể trâu $O(n^2)$. Với phiên bản giới hạn to hơn, ta có thể dùng nhiều cách khác để tối ưu như **z-function** hay **kmp**. Như trong code mình cài đặt là **kmp**.

1.4 Linear Search 4

Gọi $numDiff$ là số lượng số khác nhau. Ta nhận xét được ngay rằng nếu $numDiff > 2 * k$ thì đáp án chắc chắn là No. Ta tham lam, bỏ một vài nhóm số khác nhau vào đội một cho đến khi đủ k nhóm, những nhóm còn lại sẽ bỏ vào đội hai. Ở đây ta thấy số nhóm khác nhau trong đội hai sẽ $\leq k$. Nên ta cần sử dụng một vài nhóm trọng đội một tách ra để tăng thêm số lượng nhóm khác nhau trong đội hai. Gọi $numAdd$ là số lượng nhóm có kích thước > 1 trong k nhóm đội một. Nếu $numDiff + numAdd \geq 2 * k$ thì đáp án sẽ là Yes.

1.5 Vượt mức Pickleball v2

Để tìm được phần tử trung vị, ở bài toán này, ta để ý giới hạn $0 \leq a_i \leq 200$ nên ta có thể tạo một mảng $cnt[]$ để đếm số lần xuất hiện của d số trước và sử dụng mảng này để tìm trung vị.

1.6 Bốn ông điên

Ở bài toán này, ta sẽ có hai hướng là sắp xếp tăng dần và giảm dần. Ta sẽ lần lượt thực hiện cả hai rồi lấy kết quả tốt hơn. Để sắp xếp tăng dần, một hướng đi sẽ là bắt đầu từ phần tử đầu tiên, ta tìm ra phần tử nhỏ nhất trong $n-1$ phần tử còn lại rồi swap chúng nếu cần, tiếp tục với phần tử thứ hai, tìm trong $n-2$ phần tử còn lại rồi swap chúng nếu cần, cứ như vậy đến hết mảng. Sắp xếp giảm dần cũng tương tự.

1.7 Huấn luyện chuột

Ở bài toán này, ta để ý công thức của ta giống như một bài toán chia kẹo thông thường: $a_1 + a_2 + \ldots + a_n = n$ với $0 \leq a_i \leq n$ mà điều kiện $a_i \leq n$ luôn thoả. Đáp án của bài toán sẽ là $\binom{2\times n-1}{n-1}$. Ta để ý điều kiện của số $mod\ p$ với p là một số nguyên tố $\leq 10^5$:

- Nếu $p > 2 \times n - 1$ ta có thể tính nghịch đảo module như bình thường.
- Nhưng khi n lớn hơn, ta cần sử dụng đến định lý Lucas để tính.

2 Basic

2.1 Task

Có được vị trí p q của người đầu tiên, ta có thể tính ra số đề của người này, gọi là x. Ta muốn người kia cũng có chung mà số đề tức $x' \equiv x\ (mod\ k)$ thì sẽ có hai số gần nhất là $x-k$ và $x+k$. Ta sẽ ưu tiên $x-k$ trước theo đề yêu cầu, nếu thoả mãn sẽ lấy, còn không sẽ lấy $x+k$. Khi đã xác định được x' ta chỉ cần tính ra vị trí của người kia.

2.2 Point3D và Point2D

Đây là một dạng bài tự cài đặt hàm **Sort**, có nhiều thuật toán để lựa chọn thì mình sử dụng **Merge Sort**.

2.3 an.512.Trộn 2 mảng

Bài này thực ra là một bước nhỏ trong **Merge Sort**. Ta sẽ xét song song từng phần tử của mảng A và B, phần tử nào bé hơn ta sẽ lấy bỏ vào mảng đáp án cho đến khi một trong hai mảng hết. Mảng còn lại chưa hết thì ta bỏ tất cả những số còn lại vào mảng đáp án.

2.4 Find MEX

Ở bài này ta không được sử dụng hàm **Sort** và một vài cấu trúc dữ liệu có sẵn như **set, multiset, map,** ... Một hướng để làm bài này là ta cần tự code lại hàm **sort** để sắp xếp lại mảng, xong dựa trên đó để tìm Mex.

2.5 VU33-MaxStr

Tính chất để một **số** $\vdots$ 3 là **tổng các chữ số** của số đó $\vdots$ 3. Còn việc xuất sao để số to nhất, ta chỉ đơn giản là xuất các chữ số theo thứ tự giảm dần từ 9 về 0. Trước đó, nếu số này:

- $\vdots$ 3: ta chỉ việc xuất đáp án
- $\equiv 1 \ (mod\ 3)$: ta ưu tiên bỏ ít số nhất có thể, đầu tiên là muốn bỏ một chữ số $\equiv 1 \ (mod\ 3)$. Nếu không thể, ta bỏ hai chữ số $\equiv 2 \ (mod\ 3)$
- $\equiv 2 \ (mod\ 3)$: ta ưu tiên bỏ một chữ số $\equiv 2 \ (mod\ 3)$. Nếu không thể, ta bỏ hai chữ số $\equiv 1 \ (mod\ 3)$.

Tất nhiên khi bỏ chữ số, ta ưu tiên bỏ các chữ số nhỏ trước.

2.6 VQ44-FLOWERS

Bài này thuật toán cũng giống bài **VQ44-FLOWERS** trong **Assignment 1** khác ở chỗ ta phải tự cài đặt hàm **Sort** để thực hiện.

2.7 Kiểm Kê

ở bài toán này, ta cũng phải tự cài đặt hàm **Sort** để sắp xếp mảng tăng dần. Ta duyệt từng phần tử từ đầu tới cuối, nếu nó là phần tử đầu hoặc có giá trị khác với phần tử kề trước, ta sẽ tính là một lần có thêm một phần tử khác nhau mới.

2.8 MergeSort, InsertionSort, BubbleSort, SelectionSort

Đơn giản chỉ là những bài tập cài đặt và visualize các phương pháp sắp xếp.