MẬT KHẨU NGÂN HÀNG

Solution:

C++	https://ideone.com/bJxiZg
Java	https://ideone.com/o00sgE
Python	https://ideone.com/76xLSj

Tóm tắt đề:

Cho một chuỗi s là một chuỗi mật khẩu, gồm ba loại ký tự, mỗi loại ký tự sẽ được quy định một số điểm như sau:

- Nếu là chữ cái, dù là in thường hay in hoa, số điểm sẽ là thứ tự của chữ cái đó trong bảng chữ cái la tin. Ví dụ ký tự 'A' có điểm là 1, ký tự 'b' có điểm là 2.
- Nếu là ký tự số, ví dụ trong chuỗi s có các ký tự '2', '3', '4', '5' tính từ trái sang phải, tổng số điểm ta nhận được sẽ là: $2 * 2^0 + 3 * 2^1 + 4 * 2^2 + 5 * 2^3 = 64$.
- Nếu là những ký tự đặc biệt còn lại, số điểm mỗi ký tự là 10 điểm.

Một mật khẩu sẽ "strong" nếu như đầy đủ 3 ký tự ở trên và tổng điểm ít nhất là 100, nếu đủ 3 ký tự nhưng số điểm nhỏ hơn 100 thì mật khẩu chỉ "normal", ngược lại thì sẽ là "weak".

Hướng dẫn giải:

- Bài này thuộc dạng Implementation, tức là ta chỉ cần bám theo yêu cầu của đề là có thể làm được.
- Ta tạo 3 biến hasCharacter, hasDigit và hasSymbol lần lượt trả về giá trị true hoặc false, với ý nghĩa tương ứng là chữ cái, số hay ký tự khác có xuất hiện trong chuỗi hay không?

Ngoài ra ta có một biến score là biến dùng để tính tổng số điểm của chuỗi, một biến count_digit là thứ tự các ký tự số từ trái sang phải được đánh số bắt đầu từ 0.

- Nếu s[i] là chữ cái, ta sẽ bật hasCharacter thành true, nếu s[i] là chữ hoa thì chuyển s[i] thành chữ thường, sau đó ta cộng vào score 1 lượng là s[i] – 'a' + 1.
- Nếu s[i] là ký tự số thì ta cộng vào score một lượng là (s[i] – '0') * 2^{count_digit}, sau đó tăng count_digit lên 1 đơn vị. Đồng thời ta bật hasDigit thành true.
- Nếu s[i] khác với 2 loại trên thì ta bật hasSymbol thành true và cộng vào score 10 đơn vị.
- Nếu như hasCharacter = true và hasDigit = true và hasSymbol = true thì ta kiểm tra xem score < 100 thì in ra normal, ngược lại in ra strong. Trong trường hợp một trong 3 biến trên không = true thì ta in ra weak.

Độ phức tạp: **O(|S| * T)** với |S| là độ dài chuỗi S và T là số lượng bộ test.

STATIC COMPUTATIONAL GRAPH

Solution:

C++	https://ideone.com/dTRPAQ
Java	https://ideone.com/2EBZe0
Python	https://ideone.com/ownRpT

Tóm tắt đề: Bạn có một computational graph có N nút được đánh số từ 0 tới N - 1, chỉ có nút 0 có thể nhận giá trị thực bất kì, các nút khác có giá trị là kết quả của một trong các phép toán:

- max0(x) = giá trị của nút x nếu nó không âm, ngược lại thì 0.
- max(x_1, x_2,... x_k) = giá trị lớn nhất của các nút x_1, x_2,... x_k.
- avg(x_1, x_2,... x_k) = trung bình cộng của các nút x_1, x_2,... x_k.
- sum(a, b) = tổng giá trị của 2 nút a và b.

Hãy cho biết có tồn tại giá trị nào của nút 0 trong đoạn từ -1 đến 1 để giá trị của nút N – 1 bằng Y hay không.

Input

Dòng đầu tiên chứa số nguyên dương N ($N \leq 40$) và số thực Y ($|Y| \leq 10^9$, Y có tối đa 8 chữ số có nghĩa phần thập phân), ngăn cách nhau bởi một khoảng trắng.

N - 1 dòng tiếp theo, dòng thứ i ($1 \leq i < N$) mô tả nút thứ i:

- Nếu nút i mô tả hàm số max0(x) thì dòng i có cấu trúc 'max0 x' ($0 \leq x < i$).
- Nếu nút i mô tả hàm số sum(a, b) thì dòng i có cấu trúc 'sum a b' ($0 \leq a, b < i, a \neq b$).
- Nếu nút i mô tả hàm số avg(x_1, x_2,... x_k) hay max(x_1, x_2,... x_k) thì dòng i sẽ bắt đầu bằng chuỗi 'avg' hay 'max' tương ứng, theo sau là số nguyên dương k ($2 \leq k \leq i$), cuối cùng là k số nguyên không âm phân biệt x_1, x_2,... x_k ($0 \leq x_1, x_2, \ldots, x_k < i$). Tất cả các số và chuỗi đều ngăn cách nhau bởi khoảng trắng.

Output

In ra 'YES' nếu thõa mãn điều kiện đề bài, ngược lại in ra 'NO'.

Ví dụ

4 -0.75 max0 0	YES

avg 2 0 1 sum 2 0	

2 -1 max0 0	NO

Giải thích:

Ví dụ 1: nút 0 có thể nhận giá trị -0.5, khi đó nút 1 có giá trị là 0, nút 2 có giá trị là $\frac{-0.5+0}{2} = -0.25$và nút 3 có giá trị là $-0.25 + -0.5 = -0.75$.

Ví dụ 2: với mọi giá trị của nút 0 thì nút 1 luôn có giá trị không âm nên nó không thể có giá trị -1 được.

Hướng dẫn giải:

- Nếu gọi giá trị của nút 0 là x thì ta có một nhận xét là mọi nút trong đồ thị đều biểu diễn các hàm số **liên tục và không giảm theo x** (tức là *khi x tăng 1 khoảng cực nhỏ thì mỗi hàm cũng tăng 1 khoảng cực nhỏ hoặc không tăng*).
- Chứng minh: có thể chứng minh bằng quy nạp một cách đơn giản như sau:
  - Nút 0 rõ ràng là một hàm số liên tục và không giảm theo x (vì nó bằng x luôn rồi).
  - Giả sử các nút từ $0 \rightarrow i - 1$ đều đã là các hàm số liên tục và không giảm theo x: vì các hàm số max0, sum, max và avg đều nhận tham số là giá trị của các nút từ $0 \rightarrow i - 1$ nên hàm chúng đều liên tục và không giảm theo x (nhìn một cách trực quan thì khi x tăng lên một chút thì hàm nào cũng chỉ tăng lên một chút hoặc giữ nguyên giá trị chứ không có giảm), vậy nên dù nút i có mô tả hàm nào thì tính chất này vẫn đúng.
- Từ đó, gọi f_{min} là giá trị nhỏ nhất mà nút $N - 1$ đạt được, đây cũng chính là giá trị của nó khi x = -1, tương tự với f_{max} là giá trị lớn nhất nút $N - 1$ đạt được (tại x = 1). Vì nút $N - 1$ cũng biểu diễn hàm số liên tục và không giảm theo x nên với mọi giá trị từ f_{min} tới f_{max} của nút $N - 1$ thì luôn tồn giá trị của x tương ứng, ngược lại thì không tồn tại giá trị nào. Vậy ta chỉ cần kiểm tra $f_{min} \leq Y \leq f_{max}$, nếu điều kiện này xảy ra thì in 'YES', ngược lại in 'NO'.
- Để tính được giá trị của từng nút, ta có thể sử dụng thuật toán duyệt đồ thị DFS (Depth-First Search), kết hợp tính toán có nhớ (tránh tính giá trị một nút nhiều lần).
  Lưu ý: trong cài đặt cần xử lý số thực khéo léo vì cần độ chính xác khá cao.

Độ phức tạp: Thao tác tốn chi phí nhất là tính f_{min} và f_{max}: ta gọi thuật toán DFS 2 lần. Độ phức tạp cơ bản của DFS là O(V + E) với V là số đỉnh và E là số cạnh trong đồ thị. Trong bài này thì V = N và E có thể đạt tới N^2 (trong trường hợp nút nào cũng mô tả hàm max hoặc avg với tham số là tất cả các nút trước nó). Vậy thao tác này có chi phí $O(N^2)$ về thời gian. Và độ phức tạp tổng cộng của thuật toán này cũng là **$O(N^2)$**.

THANG MÁY

Solution:

C++	https://ideone.com/Bg4sJ0
Java	https://ideone.com/vOl35N

Tóm tắt đề:

Có một thang máy tốn thời gian X để đi từ tầng trệt lên tầng thượng hoặc từ tầng thượng xuống tầng trệt. Có N vị khách, vị khách thứ i có mặt tại tầng trệt ở thời điểm T_I (T_i tăng dần). Hãy trả lời Q truy vấn, truy vấn i yêu cầu tìm thời điểm sớm nhất để đưa những vị khách có vị trí từ L_i đến R_i lên tầng thượng, nếu thang máy có thể chứa tối đa K_i người.

Input:

Dòng đầu tiên ghi hai số N, X ($1 \le N \le 10^5, 1 \le X \le 10^9$) – số vị khách và thời gian cần để thang máy đi từ tầng trệt lên tầng thượng.

Dòng thứ hai ghi N số $T_1, T_2, \dots, T_N$ ($1 \le T_1 \le T_2 \le \cdots \le T_N \le 10^9$) – thời điểm vị khách thứ i đến tòa nhà Bitexco.

Dòng thứ ba ghi một số nguyên Q ($1 \le Q \le 10^5$) – số truy vấn cần xử lí.

Dòng thứ i trong Q dòng tiếp theo gồm ba số nguyên L_i, R_i, K_i ($1 \le L_i \le R_i \le N$, $1 \le K_i \le R_i - L_i + 1$) – mô tả truy vấn thứ i.

Output:

In ra Q dòng, dòng thứ i gồm một số nguyên là câu trả lời cho truy vấn thứ Q.

Ví dụ:

5 2 1 3 5 12 13 3 1 5 3 1 3 1 2 5 4	15 11 15

Giải thích (Test 1):

Với truy vấn đầu tiên, ta có thể thực hiện việc đưa khách như sau:

- Đợi đến thời điểm $t = 5$, cho các vị khách $1, 2, 3$ vào thang máy và đưa thang lên tầng thượng. Thang sẽ đến tầng thượng ở thời điểm $t = 7$ và trở về tầng trệt ở thời điểm $t = 9$.
- Đợi đến thời điểm $t = 13$, cho hai vị khách $4, 5$ vào thang máy và đưa thang lên tầng thượng. Thang sẽ đến tầng thượng ở thời điểm $t = 15$.

Với truy vấn thứ hai, ta có thể thực hiện việc đưa khách như sau:

- Đợi đến thời điểm $t = 1$, cho vị khách 1 vào thang máy và đưa thang lên tầng thượng. Thang sẽ đến tầng thượng ở thời điểm $t = 3$ và trở về tầng trệt ở thời điểm $t = 5$.
- Ngay tại thời điểm $t = 5$, cho vị khách 3 vào thang máy và đưa thang lên tầng thượng. Thang sẽ đến tầng thượng ở thời điểm $t = 7$ và trở về tầng trệt ở thời điểm $t = 9$.
- Ngay tại thời điểm $t = 9$, cho vị khách 2 vào thang máy và đưa thang lên tầng thượng. Thang sẽ đến tầng thượng ở thời điểm $t = 11$.

Với truy vấn thứ ba, ta có thể đợi đến thời điểm $t = 13$, cho toàn bộ các vị khách vào thang máy và đưa thang lên tầng thượng. Thang sẽ đến tầng thượng ở thời điểm $t = 15$.

Hướng dẫn giải:

Trước hết, ta thấy rằng thời gian đi lên của từng nhóm sẽ phụ thuộc vào người đến trễ nhất ccuar nhóm đó. Như vậy giả sử những người từ L đến R cần di chuyển thì những việc ưu tiên gom nhóm theo thứ tự ngược lại (tức K người đến trễ nhất làm 1 nhóm, K người sớm hơn thành 1 nhóm, …) sẽ lợi hơn vì cùng 1 số lượng nhóm như nhau nhưng với cách gom này, thang may sẽ bắt đầu di chuyển tại thời điểm sớm nhất có thể, thời gian cho các nhóm càng lại sẽ nhiều hơn. Vậy trước tiên ta đảo ngược dãy T – đổi chỗ các cặp vị trí T_i và T_{n-i-1} (dãy T được đánh số từ 0). Khi đó, truy vấn (L, R, K) trở thành $(N - R, N - L, K)$.

Giả sử chỉ có một truy vấn L, R, K. Ta sẽ chặt nhị phân đáp án. Giả sử cần kiểm tra xem có thể đưa tất cả các vị khách từ L đến R lên tầng thượng trước thời điểm M hay không. Khi đó, ta có thể:

- Cho các vị khách từ L đến $L + K - 1$ đi chuyến thang máy xuất phát ở thời điểm $M - X$.
- Cho các vị khách từ $L + K$ đến $L + 2K - 1$ đi chuyến thang máy xuất phát ở thời điểm $M - 3X$.
- …
- Cho các vị khách từ $L + iK$ đến $L + (i + 1)K - 1$ đi chuyến thang máy xuất phát ở thời điểm $M - (2i + 1)X$.

Khi đó, thời điểm mà vị khách thứ i bước vào thang máy là $M - \left(2\left\lfloor\frac{i-L}{K}\right\rfloor + 1\right)X$.

Để có thể thực hiện các chuyến thang máy trên, thời điểm có mặt của các vị khách thứ i không được trễ hơn thời điểm mà vị khách thứ i bước vào thang máy, hay:

$$M - \left(2\left\lfloor\frac{i-L}{K}\right\rfloor + 1\right)X \geq T_i \Leftrightarrow M \geq T_i + \left(2\left\lfloor\frac{i-L}{K}\right\rfloor + 1\right)X \ (\forall L \leq i \leq R)$$

Do T_i giảm dần nên điều kiện trên tương đương:

$$M \geq T_{L+iK} + (2i+1)X \ \left(\forall 0 \leq i \leq \frac{R-L}{K}\right)$$

Do đó, thời điểm sớm nhất cần tìm sẽ là $\max(T_{L+iK} + (2i+1)X)$ với $0 \leq i \leq \frac{R-L}{K}$.

Trở lại bài toán với Q truy vấn. Ta chia thành hai loại truy vấn: truy vấn có $K \leq C$ và truy vấn có $K > C$ (với C là một con số nguyên bất kì).

Với truy vấn có $K > C$, ta có thể tính trực tiếp đáp án bằng một dòng for trong $O\left(\frac{N}{C}\right)$.

Với truy vấn có $K \leq C$:

- Gọi $r = L \bmod K$. Gọi A_r là mảng bao gồm các phần tử của mảng T ở vị trí i sao cho $i \equiv r \ (mod \ K)$. Khi đó, thời điểm sớm nhất cần tìm là $\max(A_r[i] + (2i+1)X)$ với $0 \leq i \leq \frac{R-L}{K}$ (với $A_r[i]$ là phần tử thứ i của mảng A_r).
- Gọi $P_r[i] = A_r[i] + (2i+1)X$. Khi đó, thời điểm sớm nhất cần tìm là $\max(P_r[i]) - 2\left\lfloor\frac{L}{K}\right\rfloor X$ với $\left\lfloor\frac{L}{K}\right\rfloor \leq i \leq \left\lfloor\frac{R}{K}\right\rfloor$. Đây là bài toán tìm giá trị lớn nhất trên một đoạn con liên tiếp, bài toán kinh điển về cây phân đoạn.
- Với mỗi giá trị K từ 1 đến C. Ta xây dựng sẵn các mảng P_r cho các giá trị r từ 0 đến $K-1$ (và cây phân đoạn tương ứng với mảng đó). Khi đó, mỗi truy vấn (L, R, K) sẽ là một truy vấn trên cây phân đoạn tương ứng.

Vấn đề còn lại là xác định giá trị C. Với thuật toán trên:

- Về bộ nhớ: Cây phân đoạn cho một mảng độ dài n có bộ nhớ từ $2n$ đến $4n$ tùy vào cách cài đặt (cách cài đặt của người ra đề tốn bộ nhớ $4n$). Tổng độ dài của các mảng P_r với một giá trị K nào đó là N, nên tổng độ dài các mảng P_r với các giá trị K nhỏ hơn C là NC, nên tổng bộ nhớ của các cây phân đoạn là $4NC$.
- Về thời gian chạy: Tổng thời gian để xây dựng các cây phân đoạn là $O(NC)$. Thời gian thực hiện với mỗi truy vấn là $O(\log N)$ nếu $K \leq C$ hoặc $O\left(\frac{N}{C}\right)$ nếu $K > C$. Do đó, tổng độ phức tạp thuật toán là $O\left(NC + Q\left(\frac{N}{C} + \log N\right)\right)$.

Với giới hạn bộ nhớ 512MB, ta có thể chọn giá trị C khoảng 100. Khi đó, thời gian chạy trong trường hợp xấu nhất ước tính khoảng 100 triệu phép toán, và bộ nhớ cần dùng ước tính là khoảng 200MB.

Độ phức tạp: $\boldsymbol{O\left(NC + Q\left(\frac{N}{C} + \log N\right)\right)}$.