SOLUTION DAY 8/11

Câu 1: SỐ SIÊU NGUYÊN TỐ

Ý tưởng là sử dụng thuật toán loang: Từ những số siêu nguyên tố có k chữ số, ta loang ra các số siêu nguyên tố có $k+1$ chữ số... Để việc cài đặt được thuận lợi, ta sử dụng phương pháp loang theo lớp:

Khởi tạo danh sách A gồm một số không có chữ số nào (ta sẽ nói cách biểu diễn số này sau)

Khởi tạo danh sách B rỗng

Mỗi bước loang:

- Xét mọi số $x \in A$, thử nối thêm các chữ số $1 \dots 9$ vào trước biểu diễn thập phân của x để được số y, nếu $y \leq R$ và y nguyên tố thì đẩy y vào B, thêm nữa nếu $y \geq L$ thì đếm kết quả.
- Đảo nội dung danh sách A và danh sách B, làm rỗng B và lặp lại nếu như $A \neq \emptyset$

Việc nối thêm chữ số d vào trước số x có k chữ số để được số y có thể viết bởi công thức:

$$y = d.10^k + x$$

Vì vậy ta chỉ cần duy trì giá trị $P = 10^k$ tại mỗi bước loang, ban đầu $P = 1$.

Sau mỗi bước loang thì giá trị này được nhân lên 10 (do sắp phải xử lý danh sách các số siêu nguyên tố có nhiều hơn 1 chữ số so với bước trước). Cũng từ công thức này, số không có chữ số nào có thể coi là số 0.

Để có thể đảo nội dung 2 danh sách A, B trong thời gian $O(1)$, A và B được khai báo là 2 vector.

```
Result = 0;
P = 1;
A.push_back(0);
while (!A.empty())
{
    for (x: A)
        for (d = 1...9)
        {
            y = d * P + x;
            if (y <= R && y nguyên tố)
            {
                B.push_back(y);
                if (y >= L) ++Result;
            }
        }
    A.swap(B);
    P *= 10;
}
```

Mẹo cuối cùng đủ để có được 100% số điểm là dùng thuật toán kiểm tra tính nguyên tố đủ tốt, do thuật toán này phải thực hiện rất nhiều lần.

- Cách chuẩn là dùng sàng để tạo danh sách L các số nguyên tố $\leq \sqrt{R}$, khi đó để kiểm tra y ta chỉ cần duyệt các số $\in L$ và $\leq \sqrt{y}$ xem có số nào là ước của y hay không...
- Cách xác suất nhanh hơn nhiều, là dùng thuật toán Miller-Rabin với độ phức tạp $O(\log y)$ khoảng 3-5 lần trên các hệ số khác nhau (tự đọc thêm). (Không nên dùng Fermat nhỏ vì có rất nhiều trường hợp sai)

Tóm lại, vì số các số siêu nguyên tố không nhiều, yêu cầu của bài chỉ cần phương pháp nối chữ số đủ nhanh và thuật toán kiểm tra tính nguyên tố đủ mạnh.

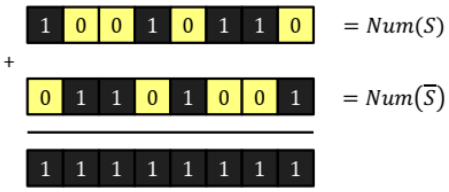
Câu 2: SỐ DƯ

Ký hiệu $Num(S)$ là giá trị số nguyên biểu diễn bởi dãy nhị phân S

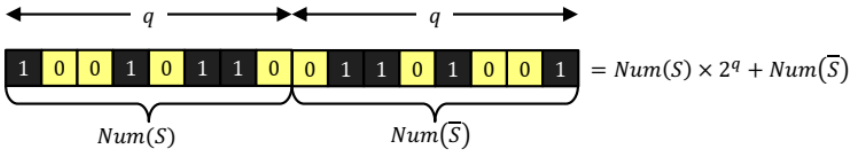
Nhận xét 1: Xâu S_i có độ dài 2^i, tức là độ dài gấp đôi so với S_{i-1}

Xét S là xâu nhị phân độ dài q, gọi $\overline{S}$ là xâu đảo bit của S khi đó:

Nhận xét 2: $Num(S) + Num(\overline{S}) = 2^q - 1$. Đơn giản là vì phép cộng $Num(S) + Num(\overline{S})$ cho ta số có biểu diễn nhị phân bằng $\underbrace{11\ldots1}_{q}$:



Nhận xét 3: Xâu nối $S + \overline{S}$ là biểu diễn nhị phân của giá trị số $Num(S) \times 2^q + Num(\overline{S})$



Đến đây ta có thể viết một chương trình tính $Num(S_n)$ như sau: Khởi tạo hai biến $x = 1$ và $\overline{x} = 0$ ứng với $Num(S_0)$ và $Num(\overline{S_0})$. Lặp n lần với $i = 1,2,\ldots,n$, mỗi bước lặp ta tính lại $x = Num(S_i)$ và $\overline{x} = Num(\overline{S_i})$ theo công thức:

$$q = 2^{i-1} \ (= \text{độ dài xâu } S_{i-1})$$
$$x_{\text{mới}} = x_{\text{cũ}} \times 2^q + \overline{x}_{\text{cũ}}$$
$$\overline{x}_{\text{mới}} = 2^q - 1 - x_{\text{mới}}$$

Tất cả các phép tính ở trên đều được thực hiện trong modulo m.

Để cải thiện thêm một chút về tốc độ và tính đơn giản, ta thêm vào những nhận xét sau:

Nhận xét 4: Từ $\overline{x} = 2^q - 1 - x$, giá trị $x \times 2^q + \overline{x}$ có thể viết thành $x \times 2^q + 2^q - 1 - x = (x + 1)(2^q - 1)$, ta không cần duy trì và cập nhật giá trị $\overline{x}$ nữa.

Nhận xét 5: Tại mỗi bước lặp, ta cần biết giá trị 2^q với q là độ dài xâu ở bước trước. Vì q tăng gấp đôi sau mỗi lần lặp, giá trị 2^q được bình phương lên so với giá trị của nó ở bước lặp trước, như vậy ta có thể tính được giá trị 2^q ở mỗi bước lặp trong thời gian $O(1)$.

```
x = 1 % m; //Num(S[0]) % m
p = 2 % m; //2^(độ dài S[0]) % m
for (i = 1; i <= n; ++i)
{
    x = (x + 1) * (p - 1) % m; //x = Num(S[i]) % m
    p = p * p % m; //Bình phương p lên
}
if (x < 0) x += m; //Trường hợp x bị âm, có thể bỏ dòng này nếu thay p - 1 trong biểu thức trên bởi p + m - 1
return x;
```

Độ phức tạp $O(n)$.

Câu 3: GIẢI CỨU

Thuật toán $O(n! \times n)$ khá tầm thường thông qua phép duyệt mọi hoán vị và kiểm tra số con chó thoát được theo một hoán vị trên tháp, tuy nhiên giải pháp này không cho ta một ý tưởng nào để nghĩ ra thuật toán tốt hơn.

Trước hết ta đặt vấn đề: Nếu ta biết chính xác trong phương án tối ưu những con chó nào sẽ phải ở lại, thì những con chó khác sẽ thoát ra theo thứ tự nào.

Không khó để chỉ ra rằng các con chó phải ở lại sẽ đứng ở đáy tháp và đứng theo thứ tự nào cũng được, ta coi như chúng chịu trách nhiệm làm giảm độ sâu của hố và không quan tâm tới chúng nữa, còn các con chó thoát ra sẽ theo thứ tự tăng dần của $b[.]$; Thật vậy, con chó có $b[.]$ lớn nhất chắc chắn phải thoát ra cuối cùng, bởi nếu nó không thoát được thì tất cả những con khác cũng không thoát được nếu đứng vào vị trí của nó. Từ đó suy ra ĐPCM.

Ý tưởng đầu tiên chính là duyệt mọi tập con của tập $\{1,2,..,n\}$ để thử mọi cách chọn tập các con chó phải ở lại. Với mỗi phép thử, xếp các con chó không nằm trong tập theo thứ tự tăng dần của $b[.]$ rồi kiểm tra, ghi nhận phương án tối ưu.

Các con chó có thể xếp tăng dần theo $b[.]$ ngay từ đầu, có tất cả 2^n tập con phải duyệt và với mỗi tập con, phép kiểm tra khả năng thoát mất thời gian $O(n)$. Ta có thuật toán $O(2^n.n)$

Thuật toán tốt hơn cũng dựa vào chính nhận xét trên: Ngay từ đầu ta xếp các con chó tăng dần theo $b[.]$ từ đỉnh tháp xuống đáy tháp.

$$b_1 \le b_2 \le \cdots \le b_n$$

Để tiện trình bày, ta gọi $d(i)$ là khoảng cách từ chân con chó i tới miệng hố, theo cách sắp xếp ban đầu này thì:

$$d(i) = h - \sum_{j=i+1}^{n} a_j$$

$d(i)$ có thể âm trong trường hợp con chó i đứng cao hơn miệng hố.

Điều kiện để con chó i thoát được là các con chó phía trên nó thoát được, đồng thời $d(i) \le b[i]$

Cách duy trì các $d(i)$ như thế nào ta sẽ nói sau trong phần kỹ thuật.

Ban đầu ta có một tháp, xét lần lượt các con chó từ 1 tới n (từ đỉnh tháp xuống), nếu gặp một con chó i không thoát được có nghĩa là tập các con chó $\{1,2,\dots,i\}$ không thể thoát hết trong phương án tối ưu, chắc chắn ta phải bỏ lại ít nhất một con trong tập này, (những) con chó bị bỏ lại sẽ được chuyển xuống đáy tháp.

Nếu con chó j nào đó bị chuyển xuống đáy tháp $(1 \le j \le i)$ thì:

- Những con phía trên nó $(1 \dots j-1)$ có $d(.)$ không đổi, khả năng thoát không bị ảnh hưởng
- Những con chó phía dưới nó $(j+1 \dots n)$ có $d(.)$ giảm đi $a[j]$, cơ hội thoát được tăng lên

Ta xét phép chọn tham lam: Chọn con chó j $(1 \le j \le i)$ có $a[j]$ lớn nhất để chuyển xuống đáy tháp. Sau phép chuyển như vậy $d(1 \dots j-1)$ không đổi, $d(j+1 \dots i-1)$ tăng lên, tức là những con chó $1,2,\dots,j-1,j+1,\dots i-1$ vẫn thoát được. Ngoài ra nếu $j \ne i$ thì con chó i chắc chắn sẽ thoát được.

Thật vậy, trước khi chuyển con chó j xuống đáy thì con chó $i-1$ thoát được, tức là:

$$\begin{aligned} & d(i-1) \le b_{i-1} \\ \Rightarrow\ & d(i) - a_i \le b_{i-1} \text{ (do con } i-1 \text{ đứng trên lưng con } i) \\ \Rightarrow\ & d(i) - a_i \le b_i \text{ (do dãy b tăng dần)} \\ \Rightarrow\ & d(i) - a_j \le b_i \text{ (do } a_j \text{ lớn nhất)} \end{aligned}$$

Vậy sau khi chuyển con chó j xuống đáy thì $d(i)$ giảm đi a_j và trở nên $\le b_i$, tức là con chó i thoát được.

Ý tưởng của thuật toán tham lam là vì các con chó vai trò như nhau, khi bắt buộc phải bỏ lại một con chó, ta sẽ bỏ lại con chó để có lợi nhất cho những con đứng dưới nó, những con phía trên nó dĩ nhiên vẫn thoát được như thường.

Bạn có thể chứng minh tính đúng đắn của thuật toán một cách chặt chẽ bằng cách giả sử phương án tối ưu chọn bỏ lại con j' thay vì con j thì việc cho con j thay thế cho con j' sẽ được một phương án không tệ hơn…

Thuật toán $O(n^2)$ đến đây trở nên đơn giản, ta không đi sâu vào phần cài đặt nữa.

Thuật toán $O(n \log n)$:

Để khéo léo duy trì các $d(i)$, trước hết ta tính $Q = \sum_{i=1}^{n} a_i$ là tổng các độ cao đến lưng của tất cả các con chó

Khi xét tới con chó i, trước hết ta đặt $Q -= a_i$ là tổng độ cao đến lưng của những con chó đứng dưới con chó i, khi đó ta có $d(i) = h - Q$.

Khi chuyển một con chó j xuống đáy, ta chỉ việc tăng Q lên a_j (một cách khác là giảm h đi a_j)

Như vậy ta không cần duy trì đại lượng $d(i)$ nữa mà tại mỗi bước lặp ta sẽ viết $h - Q$ thay vì $d(i)$.

Để chọn nhanh $a[.]$ lớn nhất, mỗi khi xét một con chó i, ta đẩy a_i vào một hàng đợi ưu tiên, nếu con chó i không thoát được, ta lấy phần tử lớn nhất trong hàng đợi ưu tiên ra, giá trị lấy ra sẽ được cộng thêm vào Q (hoặc trừ đi từ h).