Lời giải contest ICT

Le Quy Don Online Judge

Ngày 4 tháng 4 năm 2024

Mục lục I

Mục lục II

Tóm tắt đề bài Quân hậu

Cho một bàn cờ vua có kích thước $n \times n$, các cột được đánh số từ 1 đến n theo chiều từ trái qua phải, các hàng được đánh số từ 1 đến n theo chiều từ trên xuống dưới.

Cho tọa độ quân Hậu ở ô (x, y), hỏi quân Hậu có thể tiến đến ô (u, v) trong một bước hay không ?

Lời Giải

Ta tổng quát các loại đường đi của quân hậu như sau:

- Di chuyển theo chiều ngang: $(x, y + a)$ (với $-y < a \leq n - y$)
- Di chuyển theo chiều dọc: $(x + a, y)$ (với $-x < a \leq n - x$)
- Di chuyển theo chiều ngang: $(x + a, y + b)$ (với $-x < a \leq n - x$ và $-y < b \leq n - y$ và $|a| = |b|$)

Vậy với cách tổng quát trên, bài toán quy về kiểm tra ba điều kiện sau:

- Điểm (u, v) nằm trên cùng hàng với quân Hậu nếu $x = u$.
- Điểm (u, v) nằm trên cùng cột với quân Hậu nếu $y = v$.
- Điểm (u, v) nằm trên đường chéo có quân Hậu: $|x - u| = |y - v|$.

Một trong ba điều kiện trên thỏa mãn thì quân hậu chắc chắn đến được ô (u, v) trong một bước.

Độ phức tạp: $\mathcal{O}(T)$

Tóm tắt đề bài Cặp Số Đảo Ngược

Cho một dãy số nguyên dương A có N phần tử $A_1, A_2, \ldots, A_N$.

Hai cặp số nguyên dương (x, y) được coi là cặp số đảo ngược nếu đảo ngược thứ tự các chữ số của x thì được số y, và nếu đảo ngược thứ tự các chữ số của y thì được số x.

Yêu cầu: Đếm số cặp phần tử trong dãy A là cặp số đảo ngược.

Sử dụng kiểu xâu để lưu dữ liệu

Với bài này, vì các số đầu vào chỉ có độ dài dưới dạng xâu ≤ 10 (chữ số) ta có thể lưu các số nguyên A_i bằng kiểu dữ liệu xâu để dễ dàng xử lý thao tác đảo bằng hàm `reverse()` trong `C++` với độ phức tạp của thao tác đảo là không đáng kể.

Subtask 1: $N \leq 1000$

Với giới hạn N nhỏ, ta có thể duyệt qua từng cặp (i, j) với $1 \leq i < j \leq n$.

Sau đó kiểm tra bằng cách đảo một trong hai xâu A_i hoặc A_j sau đó so sánh với xâu còn lại.

Độ phức tạp: $\mathcal{O}(n^2)$

Subtask 2: $N \leq 10^5$ và $A_i \leq 10^5$

Riêng với subtask này, ta sử dụng kiểu dữ liệu **`int`** để lưu lại mảng A. Sau đó ta gọi mảng `cnt[x]` đóng vai trò như một mảng đếm số lần xuất hiện của một phần tử mang giá trị x trong mảng A.

Có nhận xét: Nếu A_i là một số có số 0 ở hàng đơn vị thì chắc chắn A_i không có số đảo ngược vì số sau khi đảo ngược sẽ có số 0 đứng ở đầu hàng không có nghĩa.

Subtask 2: $N \le 10^5$ và $A_i \le 10^5$

Tiến hành, duyệt i từ $1 \to N$. Với mỗi i có A_i không có số 0 ở hàng đơn vị ở ta thực hiện lần lượt các thao tác sau:

- Gán biến $x = A_i$, $b = 0$ với x là số A_i ban đầu và b là số A_i sau khi đảo ngược.
- Đảo ngược số x bằng cách lặp lại thao tác lấy mod 10 để lấy chữ số ngoài cùng và lưu lại vào biến $b = b \times 10 + x\%10$ và sau đó chia cho 10 để loại bỏ chữ số ngoài cùng của x cho đến khi $x = 0$.
- Sau khi đảo b ta sẽ cộng vào biến kết quả một lượng là `cnt[b]`.
- Cuối cùng ta tăng số lần xuất hiện của phần tử A_i thêm 1 trong mảng cnt.

Độ phức tạp: $\mathcal{O}(N)$.

Subtask 3: $N \leq 10^5$ và $A_i \leq 10^9$

Với subtask này, ta quay về sử dụng kiểu dữ liệu xâu để dễ dàng xử lý.

Sử dụng ý tưởng của Subtask 2, ta cần một mảng đếm `cnt[x]` nhưng vì x lưu dưới dạng xâu nên ta sử dụng cấu trúc dữ liệu `map` và dùng hàm `reverse()` để thuận tiện cài đặt bước đảo xâu.

Độ phức tạp: $\mathcal{O}(N \times \log_2(N))$

Tóm tắt đề bài Siêu giai thừa

Cho hàm $sf(N) = 1! \times 2! \times \ldots \times N!$ và số N.

Yêu cầu: Hãy tính số chữ số 0 tận cùng của N siêu giai thừa.

Tính chất của thừa số nguyên tố

Ta nhận thấy trong một dãy tích các số, số số 0 ngoài cùng được tạo ra $=$ min của số lượng thừa số nguyên tố 2 và thừa số nguyên tố 5 tạo ra khi ta phân tích mỗi phần tử trong dãy thành các số nguyên tố.

Điều này đúng, vì để tạo ra một số 0 ngoài cùng, ta đem số đó đi nhân với 10 tức là 2×5.

Trong giai thừa, dễ dàng nhận thấy số lượng thừa số nguyên tố 2 xuất hiện nhiều hơn số lượng thừa số nguyên tố 5 nên ta có thể tối giản bài toán thành đếm số lượng thừa số nguyên tố 5 khi phân tích sf thành 1 dãy tích các thừa số nguyên tố.

Subtask 1: $N \leq 10^3$

Với giới hạn N nhỏ, ta có thể cài đặt hàm `int factorial(int x)` tính số lượng thừa số nguyên tố 5 của $x!$ như sau:

Bằng cách duyệt từ i từ $1 \to N$. Với mỗi i, ta tiến hành phân tích thừa số 5 của i bằng cách gán $x = i$ sau đó chia liên tục x cho 5 cho đến khi x không chia hết cho 5 nữa và với mỗi lần chia được như vậy thì ta cộng vào biến `cnt` thêm 1 (có thêm 1 thừa số 5 được tìm thấy).

Sau đó trả về `cnt`. Độ phức tạp của hàm `factorial(x)`: $\mathcal{O}(N \times \log_5(N))$.

Subtask 1: $N \leq 10^3$

Sau khi cài đặt hàm `int factorial(int x)` ta có thể giải Subtask này như sau:

Duyệt i từ $1 \to N$. Với mỗi i, ta gọi hàm `factorial(i)` và lấy kết quả trả về của hàm cộng vào biến kết quả.

Tổng độ phức tạp: $\mathcal{O}(N^2 \times \log_5(N))$.

Subtask 2: $N \leq 10^6$

Với subtask này, hàm `factorial(x)` sẽ không được sử dụng vì nó cho một độ phức tạp rất lớn và vòng for i cuối cùng là bắt buộc.

Ta có nhận xét, giả sử ta gọi tổng số lượng thừa số 5 khi phân tích của $x!$ là y. Ta thấy rằng, tổng số lượng thừa số 5 của giai thừa $(x+1)! = y+$ số lượng thừa số 5 khi phân tích của $x+1$. Áp dụng nhận xét này, Subtask 2 có thể được giải như sau:

Gán $cnt = 0$ là biến đếm số thừa số 5.

Duyệt i từ $1 \to N$, với mỗi i ta tiến hành phân tích thừa số nguyên tố 5 như hàm `factorial(x)` nhưng lần này ta chỉ phân tích mỗi i đang xét hiện tại, với mỗi thừa số 5 như vậy ta tăng biến `cnt` lên 1 đơn vị. Sau khi phân tích xong, ta cộng `cnt` vào biến kết quả.

Tổng độ phức tạp: $\mathcal{O}(N \times \log_5(N))$

Subtask 3: $N \leq 10^6$

Để giải được subtask này. Ta làm như sau:

Ta xét các số 5^k với k được duyệt từ $1 \to \log_5(n)$. Ta nhận thấy tổng số 5 được tạo ra bởi mỗi 5^k cho $sf(n)$ sẽ được tính như sau:

- $p = n/5^k$
- Tổng số 5 do 5^k tạo ra cho $sf(n) = \dfrac{(p-1) \times p}{2} \times 5^k + (n\%5^k) * p$

Gọi hàm tính trên là $f(x)$. Vậy với mỗi 5^k ta chỉ cần cộng $f(5^k)$ vào biến kết quả với k duyệt từ $1 \to \log_5(N)$

Độ phức tạp: $\mathcal{O}(\log_5(N))$

Tóm tắt đề bài Xâu ICT

Xâu ICT là một xâu kí tự chỉ chứa ba loại kí tự `i`, `c` hoặc `t`. Cho một xâu kí tự S có độ dài N, chỉ chứa các kí tự chữ in thường từ `a` đến `z`.

Yêu cầu: Hãy tìm một đoạn xâu con liên tiếp dài nhất là xâu ICT.

Subtask 1: $N \leq 1000$

Với subtask này, ta chỉ cần duyệt l từ $1 \to N$ để chọn điểm bắt đầu của một đoạn, sau đó ta cố định l và chạy r từ $l \to N$ và kiểm tra xem kí tự thứ r có thuộc một trong 3 kí tự i, c hoặc t hay không. Nếu không thì ta ngừng lại và đem lấy max của biến kết quả hiện tại với $r - l + 1$.

Độ phức tạp: $\mathcal{O}(N^2)$

Subtask 2: $N \leq 10^6$

Với subtask 1, ta nhận thấy rằng việc duyệt tiếp $l = l + x$ với l sau khi gán $\leq r$ là vô nghĩa vì xâu mới chắc chắn sẽ không có độ dài lớn hơn xâu $r - l + 1$ ban đầu. Vì vậy, ta có thể tối giản cách duyệt trên bằng duyệt $l = r + 1$.

Với cách làm trên, mỗi phần tử chỉ bị duyệt qua nhiều nhất một lần nên độ phức tạp của cách làm này là: $\mathcal{O}(N)$

Tóm tắt đề bài Bộ ba số Pytago

Cho một dãy số nguyên dương A có N phần tử $A_1, A_2, \ldots, A_N$.

Một bộ ba số Pythago nguyên dương a, b, c sao cho $a^2 + b^2 = c^2$ hoặc $a^2 + c^2 = b^2$ hoặc $b^2 + c^2 = a^2$.

Yêu cầu: Hãy tìm số lượng bộ ba phần tử của dãy A là một bộ ba số Pythago.

Subtask 1: $N \leq 100$

Với subtask này, ta chỉ đơn giản duyệt i, j, k với $k > j > i$. Và kiểm tra xem A_i, A_j, A_k có tạo ra một bộ ba số Pythago hay không.

Độ phức tạp: $\mathcal{O}(N^3)$

Subtask 2: $N \leq 5000$

Ở đây, vì các số $A_i \leq 1000$ nên tổng $a^2 + b^2$ bất kì chỉ có thể đạt mức tối đa là 2×1000^2. Nên ta có thể dùng mảng đếm để giải bài toán này.

Gọi mảng đếm `cnt[x]` là số lần xuất hiện của phần tử x. Với mảng `cnt[x]`, ta lưu các số lần xuất hiện của các phần tử A_i^2 đại diện cho c^2 trong công thức $a^2 + b^2 = c^2$.

Vì các số a, b, c lần lượt đôi một phân biệt. Nên ta có thể tính trước mảng `cnt[x]` mà không bị tính lặp.

Việc còn lại ta chỉ cần duyệt i và j tuần tự ($j > i$) và cộng $\texttt{cnt}[A_i^2 + A_j^2]$ vào biến kết quả.

Độ phức tạp: $\mathcal{O}(N^2)$

Subtask 3: $N \leq 10^6$ và $A_i \leq 100$

Quay lại công thức: $a^2 + b^2 = c^2$

Tương tự như cách làm ở subtask 2, ta vẫn sẽ có mảng `cnt[x]` đã nêu ở Subtask 2.

Nhưng ở đây, ta không thể duyệt i, j nữa. Thay vào đó, ta gọi `dict[x]` là số lượng phần tử x đã xuất hiện khi ta duyệt tới i với i được duyệt từ $1 \to N$. Mục đích của việc tạo mảng `dict[x]` như sau:

Ta thấy rằng, $A_i \leq 100$ nên thay vì duyệt i, j thì với mỗi i ta duyệt b chạy từ $1- > 100$ nếu giá trị b đã xuất hiện trước đó thì `dict[b]` > 0 khi đó $a = A_i, b = b, c = a^2 + b^2$. Như vậy, việc còn lại chỉ là cộng $dict[b] * cnt[c]$ (số cách chọn cặp) vào biến kết quả sau đó tăng giá trị `dict`$[A_i]$ lên một đơn vị.

Độ phức tạp: $\mathcal{O}(N \times 100)$

Subtask 4: $N \leq 10^6$ và $A_i \leq 1000$

Với Subtask này, ta không thể duyệt i được nữa. Nhưng với Subtask 3, ta có thể cải tiến nó như sau. Vì a, b, c phân biệt, nên ở đây. Thay vì duyệt qua N phần tử A_i, thì ta chỉ cần duyệt qua 1000 số (vì $A_{\text{max}} = 1000$). Cách làm như sau:

Với mảng `dict[x]`, ta duyệt qua N phần tử và cập nhật $\texttt{dict}[A_i]$ thêm 1 đơn vị.

Với mảng `dict`, ta duyệt a từ $1 \to 1000$ và duyệt b từ $a \to 1000$, khi đó ta có bộ 3 số a, b và $c = a^2 + b^2$. Khi đó số cách chọn bộ ba a, b, c đã cho sẵn sẽ bằng `cnt[x]` $\times$ `dict[a]` $\times$ `dict[b]` và ta cộng lượng này vào biến kết quả.

Độ phức tạp: $\mathcal{O}(1000^2)$.

Tóm tắt đề bài Xâu con bằng nhau

Cho hai xâu s và t.

Yêu cầu: Xét tất cả các xâu con khác rỗng của s và các xâu con khác rỗng của t, đếm xem có bao nhiêu cặp xâu bằng nhau

Subtask 1: $n \leq 10$

Với $n \leq 10$, ta có thể sinh nhị phân cả hai xâu s và t. Sau đó, tính số cặp các xâu con giống nhau.

Độ phức tạp: $\mathcal{O}(2^n)$

Subtask 2: $n \leq 20$

Với $n \leq 20$, ta có thể sinh nhị phân xâu s sau đó lưu các xâu vào trong cấu trúc dữ liệu *map* rồi ta tiến hình sinh xâu t, với mỗi x là xâu con của t ta cộng `map[x]` vào biến kết quả.

Độ phức tạp: $\mathcal{O}(2^n \times \log_2(2^n))$

Subtask 3: $n \leq 7000$

Với giới hạn này, ta có thể dùng ý tưởng của bài toán "Xâu con chung lớn nhất để giải". Cụ thể:

Gọi `dp[i][j]` là số lượng xâu con chung khi xét các kí tự từ 1 đến i của sâu s, và các kí tự từ 1 đến j của xâu t.

- Trường hợp 1 (không lấy cặp i, j):
  `dp[i][j] = dp[i][j-1] + dp[i-1][j] - dp[i-1][j-1].`
- Trường hợp 2 (lấy cặp i, j (chỉ khi `s[i] = t[j]`)):
  `dp[i][j] += dp[i - 1][j - 1] + 1.`

Độ phức tạp: $\mathcal{O}(n^2)$

Subtask 4: $n \leq 10^4$

Nhận thấy rằng việc giữ chiều i sẽ khiến độ phức tạp bộ nhớ của bài toán rất lớn có thể làm chậm chương trình hoặc bị *MLE*.

Ta cần giảm đi 1 chiều của quy hoạch động. Cụ thể là chiều i, vì trong công thức quy hoạch động nói trên, ta chỉ quan tâm i hiện tại và $i-1$. Các giá trị $i-2, i-3, \ldots, 0$ đều bị bỏ dỡ trong bộ nhớ. Vì thế thay vì $i-1$, ta có thể dùng một biến `st` mang giá trị 0 hoặc 1 khi đó công thức quy hoạch động sẽ chuyển biến như sau:

- Trường hợp 1 (không lấy cặp i, j):
  `dp[st][j]=dp[st][j-1]+dp[st^1][j]-dp[st^1][j-1].`
- Trường hợp 2 (lấy cặp i, j (chỉ khi `s[i] = t[j]`):
  `dp[st][j] += dp[st ^ 1][j - 1] + 1.`

Với `st = (i ^ 1)` và `^` là phép XOR.

Độ phức tạp: $\mathcal{O}(n^2)$

Ngoài ra có thể tối ưu các phép mod để chương trình chạy nhanh hơn.

Tóm tắt đề bài Đoạn con chính phương

Cho một dãy số nguyên dương $a_1, a_2, \ldots, a_n$. Đếm số cặp (l, r) sao cho $1 \leq l \leq r \leq n$ và $a_l \times a_{l+1} \times \ldots \times a_r$ là số chính phương.

Nhận xét

Với mọi số tự nhiên, ta đều có thể phân tích dưới dạng:
$x = p_1^{k_1} \times p_2^{k2} \times \ldots \times p_n^{kn}$ (p là các số nguyên tố, k là các số tự nhiên).

Ta có nhận xét số x là số chính phương khi tất cả các số tự nhiên k đều chia hết cho 2.

Subtask 1: $n \leq 1000$

Ta có thể duyệt đoạn l từ $1 = 1 \to n$ và r từ $r = l \to n$).

Khi đó ta sẽ phân tích thừa số của mỗi $a[r]$ khi đoạn con được mở rộng và tính tổng các số mũ của các thừa số nguyên tố sau khi phân tích.

Với mỗi r có tổng số mũ của các thừa số nguyên tố từ $l \to r$ đều chẵn thì đoạn $[l, r]$ là một đoạn hợp lệ.

Độ phức tạp : $\mathcal{O}(n^2 \times \log(n))$ với $\log(n)$ là độ phức tạp phân tích thừa số nguyên tố bằng sàng Nguyên Tố.

Subtask 2: $n \leq 5 \times 10^5$ và với a_i đủ bé

Gọi `pref[i]` $(1 \leq i \leq n)$ là tích tiền tố của các phần tử từ $1 \rightarrow i$.

Vì chỉ quan tâm đến tính chẵn lẽ của số mũ k của các thừa số nguyên tố nên với mỗi i ta tính `pref[i]` bằng cách phân tích a_i thành các thừa số nguyên tố với số mũ được $\%2$. Ta tính `pref[i]` như sau :

- `pref[i] = pref[i - 1]`
- với p là thừa số nguyên tố khi phân tích a_i, ta nhân `pref[i]` với p nếu `pref[i]` $\%p! = 0$ và chia `pref[i]` với p nếu `pref[i]` $\%p = 0$.

Dễ dàng nhận thấy rằng đoạn $[l, r]$ là một đoạn hợp lệ khi và chỉ khi `pref[r] = pref[l - 1]`.

Đến đây có thể dễ dàng giải được bài toán bằng cách duyệt qua i từ $1 \rightarrow n$ và sử dụng kiểu dữ liệu map để lưu lại lần xuất hiện của `pref[i-1]`. Sau đó với mỗi i, ta chỉ cần cộng `map[pref[i]]` với biến kết quả.

Độ phức tạp: $O(n \times \log(n))$. Tuy nhiên khi a_i quá lớn sẽ dẫn tới tràn

Subtask 3: $n \leq 5 \times 10^5$

Sử dụng ý tưởng của Subtask 2. Tuy nhiên vì a_i quá lớn có thể dẫn đến tràn số khi nhân các thừa số nguyên tố lại với nhau, vì vậy thay vì nhân các thừa số nguyên tố lại với nhau thì ta hash các thừa số nguyên tố ấy kèm với số mũ của chúng.
Và cách làm cũng hệt như Subtask 2 với `pref[i]` bây giờ là một mảng Hash.
Độ phức tạp:$\mathcal{O}(n \times \log_2(n))$.