AtCoder Grand Contest 003 解説

DEGwer

2016 年 8 月 21 日

A: Wanna go back home

問題概要

二次元の平面上で、原点からスタートし、東西南北のうちの指定された方向に正の距離進む移動を $|S|$ 回行う。この $|S|$ 回の移動のあと、原点に戻ってくるように各移動の距離を設定できるかどうか判定せよ。

解説

北方向に進む移動があり、南方向に進む移動がない場合、最終的な位置は必ず原点より北になるため、原点に戻ってくることはできません。同様に、

- 南方向に進む移動があり、北方向に進む移動がない場合
- 西方向に進む移動があり、東方向に進む移動がない場合
- 東方向に進む移動があり、西方向に進む移動がない場合

も原点に戻ってくることはできません。

それ以外の場合、すなわち、南北方向の移動も、東西方向の移動も、どちらも存在しないかどちらも存在するかのいずれかである場合、原点に戻ってくることができます。具体的には、南北方向の移動どちらもがある場合、北向きの移動と南向きの移動の合計距離を両方 1 にするように移動する (東西方向も同様) 等で原点に戻る手順を構成することができます。

以上より、

- 北方向に進む移動があり、南方向に進む移動がない場合
- 南方向に進む移動があり、北方向に進む移動がない場合
- 西方向に進む移動があり、東方向に進む移動がない場合
- 東方向に進む移動があり、西方向に進む移動がない場合

は NO を出力し、それ以外の場合は YES を出力すればよいです。

B: Simplified mahjong

問題概要

$i(1 \leq i \leq N)$ の書かれたカードが A_i 枚ある。値が連続する、もしくは同じ 2 枚のカードはペアにすることができる。互いに重ならないように最大でペアは何組取れるか。

$N \leq 10^5, A_i \leq 10^9$

解説

まず、$A_i = 0$ となる i がないような場合について考えてみます。このとき、カードの合計枚数を S として、$\lfloor \frac{S}{2} \rfloor$ が答えになることを証明できます。

証明は以下の通りです。

まず、ペアは 2 枚からなる組なので、$\lfloor \frac{S}{2} \rfloor$ 組より多くのペアをとることはできません。逆に、次のようなアルゴリズムで、$\lfloor \frac{S}{2} \rfloor$ 組のペアをとることができます。

以下の手順で、カードをペアにして (ババ抜きのような要領で) 取り除いていくことを考えます。i を 1 から N まで動かし、次の操作をします。

値 i の書かれたカードが B_i 枚残っているとします。値 i のカード同士をペアにして、$\lfloor \frac{B_i}{2} \rfloor$ 組のペアを作り、これらのカードを取り除きます。このとき、値 i の書かれたカードは 0 枚または 1 枚残ります。1 枚が残った場合、

- $i = N$ なら、終了する
- そうでなければ、その 1 枚のカードを、値 $i + 1$ の書かれたカード (これは $A_i \neq 0$ より必ず存在します) とペアにして取り除く

この操作で、高々1 枚の値 N の書かれたカードのみが、ペアにならずに残ります。以上より、$\lfloor \frac{S}{2} \rfloor$ 組のペアが作れることが示されました。

では、問題の、$A_i = 0$ となる i があるかもしれない場合を考えてみます。$A_i = 0$ となるような i の書かれたカードを使うペアはないので、$A_i = 0$ となる i が出てきたところで数列 A を区切り、その区切られた各部分で上の問題を解き、最後にそれらを合計すればいいことが分かります。

以上をまとめると、数列 A_i を 0 が出てくるたびに区切り、その各部分ごとにその部分に属する A_i たちの合計の半分 (小数点以下切り捨て) を求め、すべて足し合わせたものを出力するというアルゴリズムで、この問題を解くことができました。時間計算量は $O(N)$ です。

C: BBuBBBlesort!

問題概要

長さ N の数列があり、全要素は相異なる。これを、連続する 2 要素を反転する操作と、連続する 3 要素を反転する操作でソート列にしたい。前者の操作の回数の最小値を求めよ。

$N \leq 10^5$

解説

連続する 3 要素を反転する操作というのは、1 つ飛ばしの位置にある 2 つの要素を入れ替える操作です。この操作では、数列の要素が全体の何番目の位置にあるかという値の偶奇は、どの要素に対しても変化することはありません。よって、元の列に奇数番目に現れる要素を偶数番目に持っていく操作や、元の列に偶数番目に現れる要素を奇数番目に持っていく操作は、後者の操作のみではできません。

前者の操作では、この偶奇を入れ替えることができます。前者の操作にかかわる要素は 2 つなので、高々2 つの要素に対し、この偶奇を入れ替えることができます。すなわち、その要素が元の列で何番目に現れるかを表す値と、ソート列で何番目に現れるかを表す値の偶奇が異なるような要素の個数の半分の回数の前者の操作は、少なくとも必要となります。

逆に、この回数の前者の操作と、何回かの後者の操作で、元の数列からソート列を得ることができることが証明できます。証明は以下の通りです。

元の列で奇数番目に現れ、ソート列で偶数番目に現れる要素の個数を X とします。明らかに、これは元の列で偶数番目に現れ、ソート列で奇数番目に現れる要素の個数と等しいです。

まず、以下を X 回繰り返します。

元の列で奇数番目に現れ、ソート列で偶数番目に現れる要素を一つとって A、逆に元の列で偶数番目に現れ、ソート列で奇数番目に現れる要素を一つとって B とします。適当な回数の後者の操作で、A を数列の 1 番目に、B を 2 番目に持ってくることができます。ここで前者の操作で A と B を入れ替えます。

そのあと、後者の操作のみを用いて、数列の偶数番目のみからなる数列と、奇数番目からなる数列をそれぞれバブルソートの要領でソートすると、ソート列が得られます。

以上で証明が完了しました。以上をまとめると、元の列で奇数番目に現れ、ソート列で偶数番目に現れる要素の個数を数え、その値を出力すればこの問題を解くことができます。時間計算量は $O(NlogN)$ です。

D: Anticube

問題概要

正整数からなる要素の重複を許す N 要素の集合 $A_1, ..., A_N$ がある。この部分集合で、どの 2 要素の積も立方数にならないようなものの最大のサイズを求めよ。

$N \leq 10^5, 1 \leq$ 集合の要素 $\leq 10^{10}$

解説

各正整数 t に対し、その標準形 $Norm(t)$ を、t が同じ素因数を 3 つ以上含まなくなるまで立方数で割ったものと定めます。例えば、$Norm(144) = 18$ です。

さらに、各正整数 t に対し、その相手 $Pair(t)$ を、t と掛けると立方数になるような整数の標準形 (これは定義より一意) と定めます。

集合の全要素に対して $Norm(t)$ と $Pair(t)$ が求まっていれば、各 $Norm(t) \neq 1$ に対し、標準形が $Norm(t)$ となるような要素数と標準形が $Pair(t)$ となるような要素数を数え、大きいほうをすべて足し合わせたものを答えとすればよいです。(ただし、$Norm(t) = 1$ なる要素が存在する場合、さらに 1 を足します。)

$Norm(t)$ と $Pair(t)$ を各要素に対して求めることを考えます。あらかじめ $(10^{10})^{\frac{1}{3}}$ までの素数を列挙しておけば、$Norm(t)$ に関しては、愚直に試し割りをすることで求めることができます。さらに、この操作で、集合の各要素に含まれる、その要素の 3 乗根以下の大きさの素因数はすべて求めることができます。

さて、$Pair(t)$ を求めることを考えます。集合の各要素に対して、その要素の 3 乗根以下の大きさの素因数はすべて求まっています。そのため、これらの素因数すべてでその要素を割ると、その値は以下の 3 通りに分類できます。

- 素数である
- 素数の 2 乗である
- 相異なる素数 2 つの積である

$Pair(t)$ の値は、2 番目のケースではその値の平方根を、それ以外のケースではその値の 2 乗を求めることで求まります (実装においては、実際に 2 乗を求める必要はありません)。よって、2 番目のケースかどうかを判定すればいいですが、それはあらかじめ $(10^{10})^{\frac{1}{2}}$ までの素数の 2 乗を列挙しておけば判定することができます。

以上をまとめると、

- $(10^{10})^{\frac{1}{2}}$ までの素数を列挙し、
- このデータを使って $Norm(t)$ と $Pair(t)$ を求めて適切に管理し、
- このデータから答えを求める

という操作で、この問題を解くことができます。最初の操作が篩法によって $O(\sqrt{(MAXA_i)}loglog(MAXA_i))$ で、2 番目の操作が (X 以下の素数の個数が $O(\frac{X}{\log X})$ であることを利用すると)$O(N\frac{(MAXA_i)^{\frac{1}{3}}}{\log(MAXA_i)})$ で、最後の操作が適切な実装で $O(NlogN)$ でできるので、十分高速です。

E: Sequential operations on sequence

問題概要

数列 $1, 2, ..., N$ がある。これに、現在の数列を無限回繰り返したものの先頭 A_i 文字をとる操作を Q 回施す。最終的にできる数列に $1, 2, ..., N$ がそれぞれいくつずつ現れるかを求めよ。

$N, Q \leq 10^5, A_i \leq 10^{18}$

解説

まず、$A_i \geq A_{i+1}$ なら、A_i はなかったことにしても問題ありません。なぜなら、先頭 A_i 文字をとった後に先頭 A_{i+1} 文字をとる操作は、単に先頭 A_{i+1} 文字をとる操作と同じだからです。

この事実より、A_i たちを、適切な項の削除により、単調増加となるようにした数列 B を代わりに考えます。B の長さを L とします。この適切な項の削除は、stack 等を用いることで線形時間で行うことができます。

操作を後ろから見ます。すると、最終的にできる数列に現れる各値の個数は、次のようなアルゴリズムで求めることができます。

- はじめに、A_Q 個の 1 を並べる。
- 操作列 B を後ろから見て、以下の操作を順に行う。
  - 操作列の今見ている箇所の値が X だとする。新しい列を、X 項からなり、i 項目は添え字が $mod X$ で i と等しいような元の列の要素の合計とした列とする。
- 最後に現れる数列を出力する。

数列を「のばす」操作でどの項がどこにくるのかを考えれば、このアルゴリズムの正当性は簡単に示すことができます。

このアルゴリズムを高速化することを考えます。まず、各操作を (逆順に) ひとつずつ処理する代わりに、以下のような順番で操作を行います。

- 配列 a, t を用意する。
- $t[L] = 1$ とする。
- 操作列 B を後ろから見て、以下の操作を順に行う。
  - 操作列の今見ている箇所が B_i だとする。
  - $k = B_i$ とする。
  - 操作列 B を $i-1$ から逆順に見て、次の操作を行う。

    - 今見ている箇所が B_j だとする。
    - $k \geq B_j$ なら、$t[j]$ に $t[i] \times \lfloor \frac{k}{B_j} \rfloor$ を加算し、k を $kmodB_j$ とする。
  - $a[1]$ から $a[k]$ までに $t[i]$ を加算する。

- 配列 a の中身を出力する。

このアルゴリズムにおいて、$t[i]$ の値は、i 番目までの操作を行ったときにできる数列が、最終的な数列に何回完全に現れるかを表します。k の値は、その「完全に現れる数列」の部分を除いた残りの長さを表します。

さらにこのアルゴリズムを高速化します。まず、k の値が変化するのは、$k \geq B_j$ のときのみです。この操作で k の値は半分以下となるので、この操作は高々$O(\log A_Q)$ 回しか行われません。よって、$k \geq B_j$ となる (後ろから見て) 最初の要素を二分探索等で求めることで、このシミュレーションをひとつの i に対して $O(\log Q \log A_Q)$ で行うことができます。

さらに、$a[1]$ から $a[k]$ までに $t[i]$ を加算する操作は、累積和のテクニックを用いて全体で線形時間で行うことができます。

以上より、上記のアルゴリズムの高速化で、$O(Q \log Q \log A_Q + N)$ 時間でこの問題を解くことができました。

F: Fraction of fractal

問題概要

$H * W$ のグリッドがあり、各マスは黒か白のいずれか。黒のマスは 4 方向に連結。レベル 0 のフラクタルとは黒いマスひとつで、レベル $k+1$ のフラクタルとは、レベル k のフラクタルをグリッドの黒いマスすべてに置き、それ以外の場所を白で埋めたものとする。レベル K のフラクタルの連結成分の個数を求めよ。

$H, W \leq 1000, K \leq 10^{18}$

解説

一番上の行と一番下の行がともに黒マスであるような列と、一番左の列と一番右の列がともに黒マスであるような行が両方存在したとします。このとき、フラクタルをグリッドに並べていく操作で、すべての黒マスが連結となることが帰納的にわかります。よってこのときは、1 を出力すればよいです。

そうでない場合、一番上の行と一番下の行がともに黒マスであるような列は存在しないとして一般性を失いません。このとき、フラクタルのランクが 1 増えることで、連結成分の個数がどうなるかを考えます。

仮定より、黒マス同士がフラクタルのランクを上げることで新しく縦につながることはありません。もし横にもつながることがなければ、連結成分の個数は、グリッドの黒マスの個数倍になります。つなりこのとき、求める連結成分の個数は $(\text{グリッドの黒マスの個数})^{K-1}$ です。

そこで、それ以外の場合を考えます。もとのフラクタルが横につながることで、いくつの連結成分が減少するかを数えることにします。

この減少は、グリッドで黒マスが横に連続している部分に対応する箇所で起こります。このような箇所 1 箇所につき、連結成分はいくつ減少するでしょうか？この減少は、もとのフラクタルで、同じ行の一番左のマスと一番右のマスが両方黒であるような行を持つような黒マスの連結成分の組の個数分の回数起こります。では、この値はどう計算すればいいでしょうか？

この値は、フラクタルのランクが 1 上がることで、元のグリッドで一番左のマスと一番右のマスがともに黒であるような行の個数倍になります。これは、黒マスが縦につながらないことからわかります。

以上をまとめると、求めたいものは、以下のベクトルの第一成分です。ただし、a はグリッドの黒マスの個数、b はグリッドの黒マス同士が横に連続する場所の個数、c はグリッドの右端のマスと左端のマスが両方黒であるような行の個数とします。

$$\begin{pmatrix} a & -b \\ 0 & c \end{pmatrix}^{K-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{1}$$

この値は二分累乗法で求めることができるので、$O(HW + logK)$ でこの問題を解くことができます。

AtCoder Grand Contest 003 Editorial

DEGwer

August 21st, 2016

A: Wanna go back home

Summary of Problem Statement

You start from the origin on a plane and moves $|S|$ times. In each move you move to the given direction (east, west, south, or north) by a positive distance. Determine if you can be at the origin after these moves.

Solution

If you move to the north and you don't move to the south, your position after the $|S|$ movements will be to the north of the origin, so you can't return to the origin. Similarly,

- When you move to the south and you don't move to the north
- When you move to the west and you don't move to the east
- When you move to the east and you don't move to the west

you can't return to the origin.

Otherwise you can return to the origin. When you move to both south and north, you can set the moving distances such that you move to the north by 1 in total and to the south by 1 in total. We can set the moving distance for east/west directions similarly.

Thus, if

- you move to the north and you don't move to the south, or
- you move to the south and you don't move to the north, or
- you move to the west and you don't move to the east, or
- you move to the east and you don't move to the west

print 'NO', and otherwise print 'YES'.

B: Simplified mahjong

Summary of Problem Statement

There are A_i cards with an integer $i (1 \leq i \leq N)$. You can make a pair with two cards if the integers on them are the same or adjacent. Compute the maximum number of disjoint pairs you can make.

$N \leq 10^5, A_i \leq 10^9$

Solution

First, assume that there is no i such that $A_i = 0$. In this case, we can prove that the answer is always $\lfloor \frac{S}{2} \rfloor$, where S is the total number of cards.

The proof is the following.

It is obvious that we can't make more than $\lfloor \frac{S}{2} \rfloor$ pairs. On the other hand, we can construct $\lfloor \frac{S}{2} \rfloor$ pairs in the following algorithm:

Let $x_1, \ldots, x_S$ be all the cards sorted in non-decreasing order. Then, for each i, $x_{i+1} - x_i \leq 1$ (Otherwise, there is no card with the integer $x_i + 1$, which is a contradiction). Thus, we can construct $\lfloor \frac{S}{2} \rfloor$ pairs $(x_1, x_2), (x_3, x_4), \ldots$. This way we can make $\lfloor \frac{S}{2} \rfloor$ pairs.

When $A_i = 0$ for some i, you can't use cards with the integer i, so you can split the sequence at i and for each part you can solve the problem independently (the answer is the sum of these independent problems).

Therefore, you can split the sequence A_i when 0 appears, and for each part compute the half (with the fraction part rounded down) of the sum of A_i, and the answer is the sum of these numbers. The time complexity is $O(N)$.

C: BBuBBBlesort!

Summary of Problem Statement

There is a sequence with N elements (elements are pairwise distinct). You want to sort the sequence using two types of operations: reverse two consecutive elements or three consecutive elements. Minimize the number of former operations.

$N \leq 10^5$

Solution

Reversing the order of 3 consecutive elements is equivalent to swapping two elements at the distance of two. Thus, in this operation, the parity of the position of a particular element never changes. If we want to move an element at an odd position to an even position, we must use the former operation.

In the former operation, we can change the parity of position for two elements. Thus, if we want to change the parity of position for k elements, we need at least $k/2$ operations of the former type.

On the other hand, we can prove that we can sort the sequence using $k/2$ operations of the former type (and some number of perations of the latter type).

Let X be the number of elements that are at odd positions in the original sequence but at even positions in the sorted sequence. This number is the same as the number of elements that are at even positions in the original sequence but at odd positions in the sorted sequence, so $X = k/2$.

First, repeat the following X times:

Let A be an element that is at odd position in the original sequence and at even position in the sorted sequence. Let B be an element that is at even position in the original sequence and at odd position in the sorted sequence. By repeating the latter type of operations, you can move A to the first position and B to the second position. Then you can swap A and B using the former type of operation.

After that, by using only the latter type of the operations, we can sort all elements at even positions like bubble sort. Similarly, we can sort elements at odd positions, and we sort the entire sequence.

In summary, the answer is the number of elements at odd positions that should go to even positions after sorting. The time complexity is $O(NlogN)$.

D: Anticube

Summary of Problem Statement

You are given a multiset $A_1, ..., A_N$. Compute its subset with maximum number of elements such that the product of any two distinct elements is not a cubic number.

$N \leq 10^5, 1 \leq elements \leq 10^{10}$

Solution

For an integer t, define $Norm(t)$ as the small divisor of t such that $t/Norm(t)$ is a cubic number. For example, $Norm(144) = 18$.

Also, define $Pair(t)$ as the unique integer without non-trivial cubic divisor such that $t \times Pair(t)$ is a cubic number. For example, $Pair(18) = 12$.

If we know $Norm(t)$ and $Pair(t)$ for each element t, for each $x \neq 1$ we compute the number of t such that $Norm(t) = x$ and $Pair(t) = x$, and the answer is the sum of maximum of these numbers. (If there is at least one element t such that $Norm(t) = 1$, add 1 to the answer.)

Let's compute $Norm(t)$ and $Pair(t)$ for each element. First, compute all primes smaller than $(10^{10})^{\frac{1}{3}}$. You can compute $Norm(t)$ in a straightforward way. Also, you can compute small divisors of each element.

How can we compute $Pair(t)$? For each element t, we already know all divisors up to $t^{1/3}$. Thus, we can write t as $t = (productofknownprimes) \times s$, where all prime divisors of s are greater than $t^{1/3}$. There are three cases:

- $s = 1$ or s is a prime.
- s is square of a prime.
- s is the product of two distinct primes.

In the first and the third case, $Pair(s) = s^2$, while in the second case $Pair(s) = sqrt(s)$. Thus, we only need to know whether s is a square of a prime. This can be done by pre-compute all primes up to $(10^{10})^{\frac{1}{2}}$ and their squares.

In summary, we can solve the problem in the following way:

- Compute all primes up to $(10^{10})^{\frac{1}{2}}$,
- Compute $Norm(t)$ and $Pair(t)$ for each element t using the precomputed list of small primes, and

- Compute the answer using $Norm(t)$ and $Pair(t)$.

The first step is $O(\sqrt{(MAXA_i)}loglog(MAXA_i))$. The second step is (by prime number theorem: the number of primes up to X is $O(\frac{X}{\log X})$) $O(N\frac{(MAXA_i)^{\frac{1}{3}}}{\log(MAXA_i)})$. The last step is $O(NlogN)$. Each of these steps are sufficiently fast.

E: Sequential operations on sequence

Summary of Problem Statement

There is a sequence $1, 2, ..., N$. You perform Q operations on this sequence. In the i-th operation, you replace the sequence by the first A_i elements of its infinite repetition. Compute the frequencies of $1, 2, ..., N$ in the final sequence.

$N, Q \leq 10^5, A_i \leq 10^{18}$

Solution

If $A_i \geq A_{i+1}$, we can ignore A_i. This is because the sequence of two operations A_i, A_{i+1} is equivalent to a single operation A_{i+1}.

Therefore, by removing appropriate elements, you can convert the sequence A into an increasing sequence B. Let L be the length of B. This sequence can be computed in linear time using a stack.

We process the sequence of operations in the reverse order.

- First construct a sequence with A_Q elements. All elements are 1.
- We process the sequence of operations B in the reverse order, and perform the following operations:
  - Suppose that we currently process X.
  - We replace the sequence by a new sequence with X elements. The i-th element of the new sequence is the sum of elements in the old sequence whose indices are i in $mod X$.
- Print the final sequence.

Let's improve this algorithm. Instead of processing each operation independently (in the reverse order), we process the operations in the following order:

- We use two arrays: a and t.
- Let $t[L] = 1$.
- We process the sequence of operations B in the reverse order, and perform the following operations:
  - Suppose that we currently process B_i.
  - Let $k = B_i$.

- We process the sequence of operations $B_{i-1}, \ldots, B_1$ and perform the following operations:
  - Suppose that we currently process B_j.
  - If $k \geq B_j$, we add $t[i] \times \lfloor \frac{k}{B_j} \rfloor$ to $t[j]$ and replace k with $k mod B_j$.
- Add $t[i]$ to $a[1], \ldots, a[k]$.

- Print the array a.

In this algorithm, $t[i]$ is the number of occurrences of the sequence after the i-th operation in the final sequence. The value of k is the number of elements in the final sequence except for the $t[i]$ occurrences of the sequence.

We will further improve this algorithm. The value of k changes only when $k \geq B_j$. By this operation, k decreases by at least twice, so this operation happens at most $O(\log A_Q)$ times. Thus, if we compute the maximum j such that $k \geq B_j$ using binary search, we can simulate this for each i in $O(\log Q \log A_Q)$ time.

Furthermore, we can add $t[i]$ to $a[1], \ldots, a[k]$ in $O(1)$ using partial sums.

The algorithm works in $O(Q \log Q \log A_Q + N)$ time.

F: Fraction of fractal

Summary of Problem Statement

There is an $H * W$ grid, and each cell is colored black or white. Black cells are 4-connected. The level-0 fractal is a single black cell. The level-$k+1$ fractal is obtained by arranging level-k fractals at black cells in the grid. Compute the number of connected components in the level-K fractal.

$H, W \leq 1000, K \leq 10^{18}$

Solution

We call a grid *vertically connected* if there is a column whose topmost cell and bottommost cell are both black. Similarly, define *horizontally connected*.

When the grid is *vertically connected* and *horizontally connected*, regardless of k, all black cells are connected. This can be proved by a simple induction. Thus, in this case the answer is 1.

When the grid is not *vertically connected* and not *horizontally connected*, each time when the level increase, the number of connected component is multiplied by the number of black cells in the grid. Thus, the answer is $(the number of black cells)^{K-1}$.

Otherwise, without loss of generality, we can assume that the grid is *horizontally connected* but not *vertically connected*.

For each black cell in the level-$k-1$ fractal, we put a given grid, and this increases the number of connected component by one. However, when two black cells are horizontally adjacent in the level-$k-1$ grid, the two grid patterns corresponding to these two cells will be in the same connected components. Thus, the answer is $x - y$, where x is the number of black cells in the level-$k-1$ fractal, and y is the number of horizontally adjacent pairs of black cells in the level-$k-1$ fractal.

In summary, the answer is the first element of the following vector. Here a is the number of black cells in the grid, b is the number of pairs of horizontally adjacent black cells, and c is the number of rows such that both its leftmost cell and rightmost cell are black.

$$\begin{pmatrix} a & -b \\ 0 & c \end{pmatrix}^{K-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{1}$$

The time complexity is $O(HW + logK)$.