ABC058 / ARC071 解説

writer: nuip

2017 年 4 月 8 日

For International Readers: English editorial starts from page 7.

A : $\iota \perp l$

問題文にかかれているとおり、$b-a=c-b$ であるかどうか判定すればよいです。例えばこの処理は C++では次のように書けます。

```
#include <iostream>
using namespace std;
int main(){
  int a, b, c;
  cin >> a >> b >> c;
  cout << ( b-a == c-b ? "YES" : "NO" ) << endl;
}
```

B : ∴∵∴∵

E の文字と O の文字を交互に並べるとパスワードが復元できます。$|E|-|O|=1$ のケースに気をつけてください。

```
#include <iostream>
using namespace std;
int main(){
  string E, O;
  cin >> E >> O;
  for(int i=0; i < E.size(); i++){
    cout << E[i];
```

```
    if(i < O.size()) cout << O[i];
  }
  return 0;
}
```

C：怪文書 / Dubious Document

最長の文字列を作るにはできるだけ多く文字を使ったほうがいいです。答えとなる文字列に使える a の数を c_a とします。また、$c_b, \ldots, c_z$ についても同様に定義します。

すぬけ君が文字列を作る時に a を k 個使うためには、どの文字列にも a が k 個出てきている必要があります。よって、c_a は、「a が最も少ない文字列」に出てくる a の個数となります。同じことが他の b から z までの文字についても言えます。

辞書順で最小の文字列にするためには、a から順番に並べればよいので、答えは

$$\underbrace{\mathrm{a}\mathrm{a}\ldots\mathrm{a}}_{c_a\text{個}}\underbrace{\mathrm{b}\mathrm{b}\ldots\mathrm{b}}_{c_b\text{個}}\ldots\underbrace{\mathrm{z}\mathrm{z}\ldots\mathrm{z}}_{c_z\text{個}}$$

となります。

D：井井井 /

答えを数式で表すと以下のようになります。

$$\sum_{1\leq i<j\leq n}\sum_{1\leq k<l\leq m}(x_j-x_i)(y_l-y_k)$$

これをそのまま計算すると計算量が $O(n^2m^2)$ で間に合いません。

この式は次のように因数分解することができます。

$$\left(\sum_{1\leq i<j\leq n}(x_j-x_i)\right)\left(\sum_{1\leq k<l\leq m}(y_l-y_k)\right)$$

これで x 座標と y 座標を別々に扱うことができるようになりました。しかもどちらも同じ形をしているので、x 座標の式 $\sum_{1\leq i<j\leq n}(x_j-x_i)$ をどう計算するかだけ考えれば良いです。

この式もこのままでは計算に $O(n^2)$ かかってしまいます。式をよく見ると、n 個の変数 $x_1, \ldots, x_n$ を足したり引いたりしてるだけなので、$x_1, \ldots, x_n$ がそれぞれ何回足されて何回引かれているかさえ分かればこの値は求まります。x_k が足される回数は、ループの中で $j = k$ である回数と同じです。これは k より小さい添字の個数と同じなので $k-1$ 回だとわかります。同様に x_k が引かれる回数は、k より大きい添字の個数と同じなので $n-k$ 回だとわかります。以上のことから、次の等式が成り立ちます。

$$\sum_{1 \le i < j \le n} (x_j - x_i) = \sum_{1 \le k \le n} ((k-1)x_k - (n-k)x_k)$$

この式の右辺の形式だと計算量が $O(n)$ となるため、制限時間内に答えを出すことができます。よって、この式を使って x, y 両方について計算し、掛け算すれば答えが $O(n+m)$ で求まります。

E : TrBBnsformBBtion

操作 1 や 2 を $\mathrm{A} \xrightarrow{1} \mathrm{BB}$ や $\mathrm{AAAA} \xrightarrow{2} \mathrm{A}$ のように書くことにします。

まず、どちらの操作も（空文字列を作らない限りは）逆の操作が可能です。

$\mathrm{A} \xrightarrow{1} \mathrm{BB}$ の逆の操作は次のように作れます。

$$\mathrm{BB} \xrightarrow{1} \mathrm{AAB} \xrightarrow{1} \mathrm{AAAA} \xrightarrow{2} \mathrm{A}$$

A と B が逆の場合でも、この操作の A と B を入れ替えればよいです。

$$\mathrm{AA} \xrightarrow{1} \mathrm{BBA} \xrightarrow{1} \mathrm{BBBB} \xrightarrow{2} \mathrm{B}$$

操作 2 の逆の操作は、消した AAA や BBB の隣の文字を利用して行えます。たとえば隣の文字が A であれば、次のようにします。

$$\mathrm{A} \xrightarrow{1} \mathrm{BB} \xrightarrow{1} \mathrm{AAB} \xrightarrow{1} \mathrm{AAAA}$$
$$\mathrm{A} \xrightarrow{1} \mathrm{BB} \xrightarrow{1} \mathrm{AAB} \xrightarrow{1} \mathrm{ABBB}$$
$$\mathrm{A} \xrightarrow{1} \mathrm{BB} \xrightarrow{1} \mathrm{BAA} \xrightarrow{1} \mathrm{BBBA}$$

隣の文字が B の場合でも、この操作の A と B を入れ替えればよいです。

ここで、文字列を「文字列 T に変換できるもの」というグループに分類することを考えましょう。すると、操作が双方向に行えるので、「文字列 X を Y に変換できるか」と「文字列 X と文字列 Y は同じグループか」が同じ意味になります。

ではグループにはどのようなものが考えられるでしょうか？簡単にするために、A だけが使われるように変換することにします。また、できるだけ短くすることを目指します。すると、どの文字列も次の 3 つのうちどれかになることがわかります。

- A に変換できるもの
- AA に変換できるもの
- AAA に変換できるもの

逆に、これら 3 つは互いに変換可能ではありません。これを確かめるために、文字列について「各文字について A を 1、B を 2 として合計して 3 で割ったあまり」を考えます。例えば、文字列 A は 1、AA は 2、AAA は 0 です。この値は操作 1 でも操作 2 でも変わらないので、これら 3 つの文字列は互いに変換不可能です。

以上のことから、文字列 X と Y のこの値が等しいこと（同じグループであること）と、X を Y に変換できることは同値です。そのため、各クエリに対してその部分文字列の値を計算して 3 で割ったあまりが等しいかを判定すれば良いことになります。これは、事前に累積和を計算しておくことで、各クエリに対して $O(1)$ で判定できます。

F : Infinite Sequence

2 つめの条件がややこしいので、まずこれについて考えましょう。1 である項については 2 つめの条件は何も関係ありません。では 2 以上の項についてはどうなるでしょうか？数列のどこかに x, y というふうに 2 以上の数 x, y が連続して出てきた場合、x についての条件から x, y, y のように y がもう 1 つあとに続いているはずです。さらに、1 つめの y についての条件から x, y, y, y のようにさらにもう 1 つ y がついていることが分かります。同様の議論が 2 つめ以降の y についても可能なので、ずっと y が続くということが分かります。つまり、2 以上の数が連続すると、それ以降の項は 1 通りに定まってしまいます。

次に 1 つめの条件について考えてみましょう。第 n 項からはずっと同じ数が続いているということですが、この「同じ数の繰り返し」が始まったのはどこからでしょうか？その数が a_1 からずっと続いているかもしれない (A) ですし、その前に別の数である項がある (B) かもしれません。

今後、第 n 項の数を x で表して、第 n 項に下線をひいて表すことにします。

(A) $x, x, \ldots, x, \underline{x}, x, \ldots$

(B) $?, ?, \ldots, ?, t, x, x, \ldots, x, \underline{x}, x, \ldots$

(A) の場合は単純なので、(B) の場合を考えてみましょう。x の繰り返しの直前にある数を t と呼ぶことにします ($t \neq x$)。ではその t のさらに前にはどんな数が出てきているでしょうか？t が最初の項から続いている場合 (B1) とそうで無い場合 (B2) があります。

(B1) $t, t, \ldots, t, x, x, \ldots, x, \underline{x}, x, \ldots$

(B2) $?, ?, \ldots, ?, s, t, t, \ldots, t, x, x, \ldots, x, \underline{x}, x, \ldots$

t の前の数を s と呼ぶことにします。いま t がいくつか続いているように書きましたが、前に見たとおり、$t > 1$ だと t が連続しているはずはありません。$t > 1$ ということにして場合分けしておきましょう。

(B1-1) $1, 1, \ldots, 1, x, x, \ldots, x, \underline{x}, x, \ldots$

(B1-2) $t, x, x, \ldots, x, \underline{x}, x, \ldots$

(B2-1) $?, ?, \ldots, ?, s, 1, 1, \ldots, 1, x, x, \ldots, x, \underline{x}, x, \ldots$

(B2-2) $?, ?, \ldots, ?, s, t, x, x, \ldots, x, \underline{x}, x, \ldots$

$t \neq x$ であったので、(B1-1) と (B2-1) では $x > 1$ です。まぎらわしいのでこれらについては x' と書いておきます。また、(B2-2) の場合 s としては 1 しかありえません。

(B1-1) $1, 1, \ldots, 1, x', x', \ldots, x', \underline{x'}, x', \ldots$

(B1-2) $t, x, x, \ldots, x, \underline{x}, x, \ldots$

(B2-1) $?, ?, \ldots, ?, s, 1, 1, \ldots, 1, x', x', \ldots, x', \underline{x'}, x', \ldots$

(B2-2) $?, ?, \ldots, ?, 1, t, x, x, \ldots, x, \underline{x}, x, \ldots$

以上で、第 n 項から左に辿っていくと、(B1-1) と (B2-1) と (B2-2) では 1 が出てくることが分かりました。そのようなケースは 1 で終わる有限列に “$x', x', \ldots, x', \underline{x'}, x', \ldots$” か “$t, x, x, \ldots, x, \underline{x}, x, \ldots$” を付けた数列になっています。

以上でどのような数列が条件をみたすのかがわかったので、答えを求めていきましょう。

(A) のケースは n 通りです。(B1-2) のケースは $(n-1)^2$ 通りです[1]。それ以外のケースの答えを求めるには、「1 で終わる長さ i の列で最後の項以外が条件を満たすものの数」が必要になります。これを $dp[i]$ とします。この値は動的計画法で求められます（後述）。(B2-2) のように後ろに "$t, x, x, \ldots, x, \underline{x}, x, \ldots$" が付くケースは $(dp[1] + dp[2] + \ldots + dp[n-2])(n-1)^2$ 通りです。(B1-1) と (B2-1) のように後ろに "$x', x', \ldots, x', \underline{x'}, x', \ldots$" が付くケースについては、$x' > 1$ なので、$(dp[1] + dp[2] + \ldots + dp[n-1])(n-1)$ 通りです。よって、答えは

$$n+(n-1)^2+(dp[1]+dp[2]+\ldots+dp[n-2])(n-1)^2+(dp[1]+dp[2]+\ldots+dp[n-1])(n-1)$$

となります。

さて、あとは $dp[i]$ を求めるだけです。「最後の項以外が条件を満たす、1 で終わる長さ i の列」は、次のどれかです。

- (最後の項以外が条件を満たす、1 で終わる長さ $i-1$ の列) + "1"
- (最後の項以外が条件を満たす、1 で終わる長さ $i-3$ の列) + "$2, 1, 1$"
- (最後の項以外が条件を満たす、1 で終わる長さ $i-4$ の列) + "$3, 1, 1$"

  $\vdots$
- (最後の項以外が条件を満たす 1 で終わる長さ 1 の列) + "$i-2, 1, 1, \ldots, 1$"
- "$i-1, 1, 1, \ldots, 1$"

よって、

$$dp[i] = \begin{cases} 1 & (i = 1) \\ 1 + dp[1] + dp[2] + \ldots + dp[i-1] & (i > 1) \end{cases}$$

と計算できます。$dp[i]$ の累積和を計算しておくことで $dp[1], dp[2], \ldots, dp[n]$ を $O(n)$ で求めることができます。

この $dp[i]$ の計算を行列とベクトルの掛け算で実装すると $O(\log n)$ で解くこともできます。

[1] $t > 1$ に注意

ABC058 / ARC071 Editorial

writer: nuip

April 8th, 2017

A : $\iota \perp l$

Check if $b - a = c - b$.

```
#include <iostream>
using namespace std;
int main(){
  int a, b, c;
  cin >> a >> b >> c;
  cout << ( b-a == c-b ? "YES" : "NO" ) << endl;
}
```

B : ∴∴∴

Arrange characters in E and O alternately.

```
#include <iostream>
using namespace std;
int main(){
  string E, O;
  cin >> E >> O;
  for(int i=0; i < E.size(); i++){
    cout << E[i];
    if(i < O.size()) cout << O[i];
  }
  return 0;
}
```

C : Dubious Document

Let c_a be the maximum integer such that each string contains at least c_a occurrences of the character 'a'. Define $c_b, \ldots, c_z$ similarly.

The answer is

$$\underbrace{\mathrm{aa}\ldots\mathrm{a}}_{c_a \text{occurrences}}\ \underbrace{\mathrm{bb}\ldots\mathrm{b}}_{c_b \text{occurrences}}\ \ldots\ \underbrace{\mathrm{zz}\ldots\mathrm{z}}_{c_z \text{occurrences}}$$

D :

The answer is

$$\sum_{1\le i<j\le n}\sum_{1\le k<l\le m}(x_j-x_i)(y_l-y_k)$$

which is equal to

$$\left(\sum_{1\le i<j\le n}(x_j-x_i)\right)\left(\sum_{1\le k<l\le m}(y_l-y_k)\right)$$

Thus, we can solve the problem for each coordinate independently.

In order to solve the problem for x-coordinates quickly, use the following equation:

$$\sum_{1\le i<j\le n}(x_j-x_i)=\sum_{1\le k\le n}((k-1)x_k-(n-k)x_k)$$

It works in $O(n+m)$.

E : TrBBnsformBBtion

Replace 'A' with '1' and 'B' with '2'. We claim that we can convert a string s to another string t if and only if the sum of digits of s and the sum of digits of t are equivalent in modulo 3.

It is trivial that this is a necessary condition. We want to prove that this is sufficient.

First, the operations are revertable (except for the case where the initial string is empty):

$$\mathrm{BB} \xrightarrow{1} \mathrm{AAB} \xrightarrow{1} \mathrm{AAAA} \xrightarrow{2} \mathrm{A}$$

$$\mathrm{A} \xrightarrow{1} \mathrm{BB} \xrightarrow{1} \mathrm{AAB} \xrightarrow{1} \mathrm{AAAA}$$
$$\mathrm{A} \xrightarrow{1} \mathrm{BB} \xrightarrow{1} \mathrm{AAB} \xrightarrow{1} \mathrm{ABBB}$$
$$\mathrm{A} \xrightarrow{1} \mathrm{BB} \xrightarrow{1} \mathrm{BAA} \xrightarrow{1} \mathrm{BBBA}$$

Using the operations $B \to AA$, we can convert any (non-empty)string to a string that consists only of 'A', so any string can be converted into one of the strings 'A', 'AA', or 'AAA'. These three strings have different values in modulo 3. Thus, this condition is also sufficient.

F : Infinite Sequence

Let N be a fixed integer. Let $dp[k]$ be the number of infinite sequences $a_1, a_2, \ldots$ such that

- For each i, $1 \leq a_i \leq N$ (Note that this N is constant regardless f the value k)
- $a_k = a_{k+1} = \cdots$
- For each (i, j, k), if $i < j < k \leq i + a_i$, $a_j = a_k$.

The problem asks to compute $dp[N]$.
Let's compute $dp[k]$ by case-analysis:

- If the first element is 1, there are $dp[k-1]$ ways to decide the remaining elements.
- If the first element is not 1 (call it c), and the second element is also not 1 (call it d), the sequence must be $c, d, d, d, \ldots$. In total there are $(N-1)^2$ sequences of this form.
- If the first element is $c > 1$ and the second element is 1,
  - If $c + 1 \leq k$, there are $dp[k - c - 1]$ ways to decide the remaining elements.
  - Otherwise, the sequence must be $c, 1, 1, 1, \ldots$.

Thus, $dp[k] = dp[k-1] + (N-1)^2 + dp[k-3] + dp[k-4] + \cdots + dp[1] + (N - k + 2)$.