ABC066 / ARC077 解説

writer: nuip

2017 年 7 月 1 日

For International Readers: English editorial starts from page 8.

A : ringring

$a+b$ と $b+c$ と $a+c$ の中で一番小さいものを出力する問題です。以下の実装例では、かわりに a, b, c の中で最も大きいものを求めて、$a+b+c$ から引いています。

```
#include<stdio.h>

int main(){
  int a, b, c;
  scanf("%d␣%d␣%d", &a, &b, &c);
  int max=a;
  if(b>max) max=b;
  if(c>max) max=c;
  printf("%d\n", a+b+c-max);
  return 0;
}
```

B : ss

偶文字列かどうかを調べるためには、文字列の前半と後半が全く同じかどうかを調べれば良いです。

偶文字列の長さは必ず偶数長なので、偶数の文字列についてのみ調べれば十分です。最長のものを探しているので、長さ $|S|-2$ の文字列、長さ $|S|-4$

の文字列、…という順に調べて、条件に合うものが見つかった時にその長さを出力すると答えが求まります。

```c
#include<stdio.h>
#include<string.h>

int main(){
  char str[222];
  scanf("%s", str);
  int n=strlen(str);
  for(int i=n-2; i; i-=2){
    if(strncmp(str, str+i/2, i/2)==0){
      printf("%d\n" ,i);
      return 0;
    }
  }
  return 0;
}
```

C : pushpush

毎回逆向きの並び替えをしていると時間がかかりすぎてしまうので ($O(n^2)$)、これを避ける方法を考えます。入力例を参考に考えると、数列の最初と最後に交互に a_i を追加していくと答えが求まることが分かります。また、最後に追加した項は最初に来るはずなので、

1. i と n の偶奇が一致していれば、数列の前に a_i を追加する。
2. i と n の偶奇が一致していなければ、数列の後ろに a_i を追加する。

という操作をすれば良いことが分かります。

この操作を効率よくするためにはどうすれば良いでしょうか?C++を使っている場合は、stl::deque という標準ライブラリの機能を使うと簡単に出来ます。そうでない場合は、大きめの配列を確保して、真ん中あたりから順番に使っていくと実装できます。この方針で実装した C 言語の実装例は以下のとおりです。

```c
#include<stdio.h>

int array[512345];
```

```
4  int a[212345];
5
6  int main(){
7    int n;
8    scanf("%d", &n);
9    for(int i=0; i<n; ++i) scanf("%d", &a[i]);
10   int left=212345, right=left+1;
11   for(int i=0; i<n; ++i){
12     if(i%2 == (n-1)%2){
13       array[left--]=a[i];
14     }else{
15       array[right++]=a[i];
16     }
17   }
18   for(int i=left+1; i<right; ++i){
19     printf("%d␣",array[i]);
20   }
21   return 0;
22 }
```

D : 11

数の種類が n 種で、項が $n+1$ 項あるので、必ず同じ数の項が複数存在します[1]。また、すべての数が 1 度以上現れるという制約から、同じ数である項はちょうど 1 組であることが分かります。

同じ数があるので、単純に $n+1$ 個から k 個を選ぶ組合せ ${}_{n+1}C_k$ を計算するだけでは答えが求められません。そこで、重複を除く方法を考えてみます。

同じ数が出てくるのはちょうど 1 組だけなので、${}_{n+1}C_k$ から重複してるものの数を引けば答えが求まります。重複している数列はどのような数列でしょうか？

取り出した部分列を見てどこから取り出した部分列であるかを考えたとき、1 通りに定まらない場合、その数列は 2 回数えられていることになります。次のような例を考えてみます。

$$2\mathbf{3}1\mathbf{4}5\mathbf{1}67$$

[1] 参考: 鳩ノ巣原理

重複している数である 1 は太字で書いてあります。もし部分列で 1 を一度も選んでいなければ、簡単に復元できます。

$$256 \to \underline{2}3\mathbf{1}45\underline{\mathbf{1}}\underline{6}7$$

部分列で 1 を 2 つとも選んでいる場合も、やはり簡単に復元できます。

$$211 \to \underline{2}3\underline{\mathbf{1}}45\underline{\mathbf{1}}67$$

また、1 を選んでいても、2 つの 1 と 1 の間の数 4, 5 を選んでいれば、これらの数と 1 との位置関係から、使われている 1 がどちらの 1 であるか特定できます。

$$51 \to 23\mathbf{1}45\underline{\mathbf{1}}67$$

よって、以上のような場合は重複して数えられない数列です。1 を 1 つだけ選んでいて、2 つの 1 と 1 の間の数 4, 5 を選んでいない場合はどちらの 1 を選んでいるかわかりません。これが重複する場合です。

$$\begin{aligned} 31 &\to 2\underline{3}\mathbf{1}45\underline{\mathbf{1}}67 \ ? \\ &\to 2\underline{3\mathbf{1}}45\mathbf{1}67 \ ? \end{aligned}$$

よって、重複している項を $a_l, a_r (l < r)$ とすると、このような部分列の数は、「a_l か a_r の片方を選んで a_l と a_r で囲われていない項から残りの $k-1$ 個を選ぶ場合の数」になります。a_l と a_r で囲われていない項は a_l の左に $l-1$ 項と、a_r の右に $n-r$ 項あるので、求める場合の数は ${}_{l-1+n-r}C_{k-1}$ であることが分かります。

よって、k 行目には、${}_{n+1}C_k - {}_{l-1+n-r}C_{k-1}$ を出力すれば良いです。

${}_nC_k$ の求め方は、ABC042/ARC058 の D 問題の解説を参照して下さい。

`http://arc058.contest.atcoder.jp/data/arc/058/editorial.pdf`

E : guruguru

まず、お気に入りボタンは必ず明るさを切り替える一番最初に押した方が効率的です。なぜなら、それ以前に他のボタンを押していたとしても、お気に入りボタンを押した後の明るさは変わらないためです。

順送りボタンのみを使って明るさ A から B に切り替えるためボタンを押す回数は、$A \leq B$ なら $B-A$ 回で、$A > B$ なら $B+N-A$ 回です。これは、まとめて $(B+N-A)\%N$ と書けます。ただし、$x\%y$ で x を y で割ったあまりを表します。

よって、お気に入りボタンを最初に押した場合のボタンを押す回数は $1+(B+N-X)\%N$ 回で、そうでない場合は $(B+N-A)\%N$ 回です。

また、お気に入りボタンを使うのは、a_i と a_{i+1} 間（a_{i+1} を含む）の状態にお気に入りの明るさ x が存在しているときのみです。

以上を元に、お気に入りが x であるときと $x+1$ である時でボタンを押す回数がどのように変わるか考えてみます。

- $a_{i+1}=x$ である i
  お気に入りボタンを 1 回押すだけで済んでいたのが、順送りボタンを $(a_{i+1}+N-a_i)\%N$ 回押さないといけなくなります。
- a_i と a_{i+1} 間（両端を含まない）に x があるような i
  順送りボタンを押す回数が 1 回減ります。
- それ以外
  変化無しです。

まとめると、ボタンを押す回数は、$a_{i+1}=x$ であるすべての i について $(a_{i+1}+N-a_i)\%N-1$ 回分増えて、a_i と a_{i+1} 間に x があるような i の数だけ減ります。

a_i と a_{i+1} 間に x があるような i の数は、累積和を取る操作を使うと $O(n+m)$ で求めることができます。imos 法という名前で呼ばれることもあります。

$a_{i+1}=x$ である i については、当然全部で n 個しか無いので、いちいち計算して足しても問題ありません。

以上から、$x=0$ である場合の答えを求めた後、$x=i$ の場合を使って $x=i+1$ の場合を順番に求めていくことで、すべての x についてボタンを押す回数が求まるため、最小値も求まります。

F : SS

まず f がどのような関数か考えてみましょう。

A,B,C をそれぞれアルファベット 1 文字として、

ABCABC

という偶文字列を考えます。これに文字を付け加える偶文字列を作るためには、付け加える文字の数は偶数でないといけません（偶文字列の長さは偶数文字なので）。まずは 2 文字付け加えることを考えてみます。

ABCABC??

これが偶文字列であるなら、赤い文字列と青い文字列は同じである必要があります。よって、“AB” と “BC” は同じ文字である必要があります。

同じように、 $2k$ 文字付け加えることを考えてみます。

$$S_1S_2\dots S_nS_1S_2\dots S_k\ S_{k+1}\dots S_nX_1X_2\dots X_{2k}$$

この場合、S の最初の $n-k$ 文字と、S の $k+1$ 文字目以降とが一致している必要があることが分かります。これは、S が周期 k の文字列であるということを意味します。$X_1X_2\dots X_{2k}$ は自由に決められるので、S が周期 k の文字列である場合、必ず $2k$ 文字付け加えて偶文字列が作れます[2]。

以上から、$f(S)$ を求めるには、S の最小周期 k を求めればいいことがわかりました。

ここで、同じ文字列 S を n 回繰り返してできる文字列を $S\times n$ と書くことにします。例えば、 “abc”$\times 3$ は”abcabcabc” となります。

ここからは、問題を偶文字列 $S\times 2$ のままでなく、「半分」にした文字列 S の状態で考えることにします（そうした方が扱いやすいからです）。半分にした状態の文字列を扱うために、関数 g を導入します。関数 g を

$$g(S)\times 2=f(S\times 2)$$

で定めます。右辺は必ず偶文字列であるため、このような $g(S)$ は必ず存在します。$g(S)$ は $f(S)$ の接頭辞になるため、$f^{10^{100}}(S)$ のかわりに $g^{10^{100}}(S)$ を考えても同じ結果が得られます。

g がどのような関数であるかを考えます。S の最小周期を x として、S の長さ x の接頭辞 を T とします。このとき、正の整数 n と S は T の接頭辞である文字列 T' を使って

$$S=T\times n+T'$$

と書けます。

もし x が $|S|$ の約数である場合は単純で、$S=T\times n$ と書けるので、

$$f(S\times 2)=f(T\times 2n)=T\times(2n+2)$$

となります。よって、$g(T\times n)=T\times(n+1)$ です。

そうでない場合、先程の結果を使うと、$g(S)$ の長さは $|S|+x$ であるはずなので、

$$g(S)=T\times n+T'+T$$

[2] $(X_1X_2\dots X_n)=(S_{n-k+1}S_{n-k+2}\dots S_nS_1S_2\dots S_k)$ とすれば良いです。

と分かります。x の最小性から、この文字列$T \times n + T' + T$ の周期は $|T \times n + T'|$ です。よって、もう一度 g でこの文字列を写すと、

$$\begin{aligned} g^2(S) = g(T \times n + T' + T) &= (T \times n + T' + T) + (T \times n + T') \\ &= g(S) + S \end{aligned}$$

となります。これはどの偶文字列 S に対しても成り立つため、$g^{n+2}(S) = g^{n+1}(S) + g^n(S)$ であることが分かります。

あとは、この式をもとに、各文字 c について、$g^n(S)$ に現れる回数を計算し、i 番目までに c が現れる回数を再帰的に数えれば良いです。

ABC066 / ARC077 Editorial

writer: nuip

July 1st, 2017

A : ringring

```c
#include<stdio.h>

int main(){
  int a, b, c;
  scanf("%d␣%d␣%d", &a, &b, &c);
  int max=a;
  if(b>max) max=b;
  if(c>max) max=c;
  printf("%d\n", a+b+c-max);
  return 0;
}
```

B : ss

```c
#include<stdio.h>
#include<string.h>

int main(){
  char str[222];
  scanf("%s", str);
  int n=strlen(str);
  for(int i=n-2; i; i-=2){
    if(strncmp(str, str+i/2, i/2)==0){
      printf("%d\n" ,i);
      return 0;
    }
  }
  return 0;
}
```

C : pushpush

You can observe that a_i are appended to the beginning or the end, alternately. The last element (a_n) will be appended to the beginning.

Therefore, we start from an empty sequence, and in the order $a_1, \ldots, a_n$,

1. If i and n have the same parity, append a_i to the beginning of the sequence.
2. If i and n have different parities, append a_i to the end of the sequence.

This can be implemented in $O(n)$ with a deque.

D : 11

There are $\binom{n+1}{k}$ ways to choose k elements from the sequence, but this way we may count the same subsequence multiple times.

For example, consider the following case:

$$23\underline{1}45\underline{1}67$$

Fix a subsequence s of this sequence. How many times will s be counted? We can observe that:

- If s contains two 1s, the positions can be uniquely determined and it will be counted only once.
- If s contains no 1, the positions can be uniquely determined and it will be counted only once.
- If s contains 4 or 5, the positions can be uniquely determined and it will be counted only once.
- Otherwise, we can't determine which 1 we chose; it will be counted twice.

So, we must subtract the number of ways to choose $k-1$ elements from $2, 3, 6, 7$.

In general, if the distance between two elements of the same value is d, the answer is $\binom{n+1}{k} - \binom{n-d}{k-1}$.

E : guruguru

Let $f(s, t, x)$ be the number of buttons we must push to change the brightness from s to t, when the favorite brightness is set to x.

First consider the following slow solution. We have an array c of length m. Initially all values are zeroes. Then, for each i and x, we add $f(a_i, a_{i+1}, x)$ to $c[x]$. The answer is the minimum value in the sequence c.

How can we improve this solution? We want to do this efficiently for a fixed i. Let's fix i, and let $s = a_i, t = a_{i+1}$. Assume that $s < t$ (the other case can be handled similarly).

- If $x \leq s$, $f(s, t, x) = t - s$.
- If $s < x \leq t$, $f(s, t, x) = t - x + 1$.
- If $t < x$, $f(s, t, x) = t - s$.

Notice that in all three cases, the value of f is a linear function of x.

Thus, we can solve this problem by adding $O(N)$ linear functions to given ranges of c. This can be done by two prefix sums: one for linear part and one for constant part.

F : SS

Suppose that initially we have a string of length $2n$.

$$S_1 S_2 \ldots S_n S_1 S_2 \ldots S_n$$

Can we make it an even string by appending $2k$ characters?

$$S_1 S_2 \ldots S_n S_1 S_2 \ldots S_k \; S_{k+1} \ldots S_n X_1 X_2 \ldots X_{2k}$$

From this, you see that the first $n-k$ characters of $S_1 S_2 \ldots S_n$ and the last $n-k$ characters of $S_1 S_2 \ldots S_n$ must be the same. It means that $S_1 S_2 \ldots S_n$ has a period of k.

In general, $f(SS) = STST$, where T is the first k characters of S, where k is the shortest period of S.

Let's define $g(S) = ST$, where T is defined in the same way. We can prove the following:

- If $g(S) = ST$ and $|T|$ is a divisor of $|S|$, $g(ST) = STT$.
- If $g(S) = ST$ and $|T|$ is not a divisor of $|S|$, $g(ST) = STS$.

By repeating this, we get:

- If $g(S) = ST$ and $|T|$ is a divisor of $|S|$, $g^{\infty}(S) = STTTTT....$
- If $g(S) = ST$ and $|T|$ is not a divisor of $|S|$, $g^{i+2}(S) = g^{i+1}(S) + g^{i}(S)$ for each i.

The remaining part is not very hard. Note that in the actual implementation you don't need to consider the first case: you get the same value for $g^{\infty}(ST)$ from the second case anyway.

===

Sketch of the proof for the second case (the first case is easy).

Suppose that

$$S = T \times n + T'$$

where n is a positive integer, $|T|$ is the shortest period of S, and T' is a non-trivial prefix of T.

We want to prove that the shortest period of

$$g(S) = T \times n + T' + T$$

is $|T \times n + T'|$.

Assume that there exists a shorter period $x < |t \times n + T'|$.

- If $x \neq |T'| \pmod{|T|}$ and $x < |t \times n + T'|$, we can prove that T has a period of $gcd(|T|, x - |T'|)$. Thus, S also has a period of $gcd(|T|, x - |T'|)$, and it contradicts the fact that the shortest period of S is $|T|$.
- Otherwise, $x \neq 0 \pmod{|T|}$ and $x \leq |t \times (n-1) + T'|$. In this case we can prove that S has a period of $gcd(|T|, x)$, and again we get a contradiction.