

ABC 141 解説

beet, drafear, kort0n, sheyasutaka

2019 年 9 月 15 日

For International Readers: English editorial starts on page 9.

A: Weather Prediction

与えられた文字列に対し、対応する文字列を出力する問題です。

条件文を 3 つ列挙することで解くことができます。

以下は C++ における実装例です。

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main(){
5     string s;
6     cin>>s;
7     if(s=="Sunny") cout<<"Cloudy"<<endl;
8     if(s=="Cloudy") cout<<"Rainy"<<endl;
9     if(s=="Rainy") cout<<"Sunny"<<endl;
10    return 0;
11 }
```

B: Tap Dance

条件を「奇数文字目が L でない、偶数文字目が R でない」と言い換えることで単純に書くことができます。以下に実装例を示します。

Listing 1 C での実装例

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void){
5     char s[100 + 1];
6
7     scanf("%s", s);
8
9     int n = strlen(s); // 文字列の長さを得る s
10    for (int i = 0; i < n; i++) {
11        if (i % 2 == 0) {
12            // 奇数文字目
13            if (s[i] == 'L') {
14                puts("No");
15                return 0;
16            }
17        } else {
18            // 偶数文字目
19            if (s[i] == 'R') {
20                puts("No");
21                return 0;
22            }
23        }
24    }
25
26    puts("Yes");
27    return 0;
28 }
```

Listing 2 Python3 での実装例

```
1 import re
2
3 s = input()
4 p = re.compile(r'^([^L][^R])*[^L]?$')
5 if p.match(s):
```

```
6  print("Yes")
7  else:
8  print("No")
```

C: Attack Survival

次の 2 つの操作は同等です.

- 正解者以外の $N - 1$ 人のポイントを 1 減らす.
- N 人全員のポイントを 1 減らしてから, 正解者のポイントだけ 1 増やす.

したがって, ポイントの変動は以下のように言い換えられます.

1. あらかじめ全員のポイントを Q 減らしておく (つまり, 開始時点のポイントは $K - Q$).
2. Q 回の正解それぞれに対して, 正解者のポイントを 1 増やす.

あとはこれをそのまま実装すればよいです.

D: Powerful Discount Tickets

X を 2^Y で割った切り捨て $\lfloor \frac{X}{2^Y} \rfloor$ は

$$\lfloor \frac{X}{2^Y} \rfloor = \lfloor \frac{\lfloor \frac{X}{2^{Y-1}} \rfloor}{2} \rfloor$$

これは、任意の正整数 X, b_1, b_2 について、

$$\lfloor \frac{\lfloor \frac{X}{b_1} \rfloor}{b_2} \rfloor = \lfloor \frac{X}{b_1 b_2} \rfloor$$

であるためです。まず、これを示します。

Proof. X を b_1 で割った商を q_1 、余りを r_1 とすると $\lfloor \frac{X}{b_1} \rfloor = q_1$ であり、

$$X = b_1 q_1 + r_1 \quad (0 \leq r_1 \leq b_1 - 1)$$

が成り立ちます。逆にこれが成り立つような非負整数 q_1, r_1 は一意なので、そのような q_1, r_1 に対して $\lfloor \frac{X}{b_1} \rfloor = q_1$ です。次に、 $\lfloor \frac{X}{b_1} \rfloor = q_1$ を b_2 で割った商を q_2 、余りを r_2 とすると、 $\lfloor \frac{\lfloor \frac{X}{b_1} \rfloor}{b_2} \rfloor = q_2$ であり

$$q_1 = b_2 q_2 + r_2 \quad (0 \leq r_2 \leq b_2 - 1)$$

が成り立ちます。したがって、

$$X = b_1 q_1 + r_1 = (b_1 b_2) q_2 + (r_1 + b_1 r_2)$$

であり、

$$0 \leq r_1 + b_1 r_2 \leq (b_1 - 1) + b_1(b_2 - 1) = b_1 b_2 - 1$$

なので $\lfloor \frac{X}{b_1 b_2} \rfloor = q_2$ です。 □

したがって、品物を順番に購入し、割引券を品物ごとにまとめて使う代わりに、「割引券を1枚使うたびに1つの品物を選んで値段を半額(小数点以下切り捨て)にする」として構いません。よって、最も値段が高い品物に割引券を1枚使うことを M 回行えば良いです。なぜなら、まだ割引券が余っているのに今最も高いもの(品物 X とする)に割引券を今後使わないとすると、最後に割引券を使ったものの代わりに品物 X に割引券を使ったほうがお得だからです。しかし、これを愚直に実装すると、時間計算量が $O(NM)$ となり間に合いません。そこで優先度付きキュー (Priority Queue) の出番です。優先度付きキューを使えば

- 値の追加
- 値の削除
- 最大値の取得

がそれぞれ $O(\log |Q|)$, $O(\log |Q|)$, $O(1)$ でできます。ここで、 $|Q|$ は優先度付きキューに入っている値の数です。これを使うことで、1 回の操作 (最高額の品物を選んで半額にする) を $O(\log N)$ の時間計算量でできるため、全体としては $O(M \log N)$ となり間に合います。

E: Who Says a Pun?

Z-Algorithm と呼ばれるアルゴリズムを使うと、長さ N の文字列 S について、 $i = 2, 3, \dots, N$ での S の i 文字目以降から成る文字列と S の最長共通接頭辞の長さを $O(N)$ で求めることが出来ます。

これにより、 $l_1 = 1$ に制限した場合の答えは、 $O(N)$ で求めることが出来ます。

あとは、同様の処理を $S[2 : N], S[3 : N], \dots, S[N : N]$ についても行うことにより、問題の答えを得ることが出来ます。全体での時間計算量は $O(N^2)$ です。

別解として、ローリングハッシュを用いて文字列比較を $O(1)$ で処理すると、二分法及び平衡二分探索木を用いたハッシュ値の既出判定を行うことにより、 $O(N(\log N)^2)$ で答えを求めることが出来ます。ハッシュ値の衝突には十分に注意してください。

F: Xor Sum 3


$(A_1 \oplus A_2 \oplus \dots \oplus A_N) \leq (A_1) + (A_2 \oplus \dots \oplus A_N)$ から、全て同じ色で塗ることを許しても答えに影響しないことがわかります。

赤く塗った数の XOR, 青く塗った数の XOR をそれぞれ $X_{\text{あおくない}}, X_{\text{あおい}}$ とおきます。明らかに $X_{\text{あおくない}} \oplus X_{\text{あおい}} = A_1 \oplus \dots \oplus A_N$ であり、この値は定数です。この値を 2 進法表記したとき、0 となる桁を「興味深いビット」と呼ぶことにします。

$X_{\text{あおくない}}, X_{\text{あおい}}$ の i ビット目は、

- i ビット目が興味深くないとき、 $(0, 1)$ か $(1, 0)$, つまり和は一定
- i ビット目が興味深いとき、 $(0, 0)$ か $(1, 1)$

です。 $2^i > 2^{i-1} + 2^{i-2} + \dots + 2^0$ より、興味深いビットを上位から貪欲に $(1, 1)$ としていくのが最適とわかり、次のように問題を言い換えられます。



A_i の各要素の、興味深くないビットを無視した数列 A'_i を考え、この数列に対する $X'_{\text{あおくない}} + X'_{\text{あおい}}$ を最大化する。興味深くないビットを全て立てた整数をこれに加えれば、元の問題の答えとなる。

上の問題で $X'_{\text{あおくない}} = X'_{\text{あおい}}$ なので、結局 $X'_{\text{あおい}}$ を最大化すればよいです。つまり、以下の問題が解けたらよいです。

A'_1, A'_2, \dots, A'_N の中のいくつかを青く塗り、その XOR を最大化する。

これは以下のようにして解けます:

1. 数列 A'_i を $N \times 60$ 行列とみなし、その行標準形を求める。ここで、列の数 60 は A'_i のビット数を表す。
2. 上の行から貪欲に $X'_{\text{あおい}} \leftarrow X'_{\text{あおい}} \oplus (i \text{ 行目})$ とする。

以上よりこの問題は解かれ、ビット演算を利用して簡単に実装できます ([C での実装例](#))。

ABC 141 Editorial

beet, drafear, kort0n, sheyasutaka

September 15, 2019

A: Weather Prediction

This is a problem of outputting the corresponding string to the given string.

You can enumerate three conditions to solve it.

The following is an implementation example in C++.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main(){
5     string s;
6     cin>>s;
7     if(s=="Sunny") cout<<"Cloudy"<<endl;
8     if(s=="Cloudy") cout<<"Rainy"<<endl;
9     if(s=="Rainy") cout<<"Sunny"<<endl;
10    return 0;
11 }
```

B: Tap Dance

You can rephrase the conditions as "the character of odd index is not L, and the character of even index is not R" so that implementation will be easy. The following is the implementation example.

Listing 1 Implementation example in C

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void){
5     char s[100 + 1];
6
7     scanf("%s", s);
8
9     int n = strlen(s); // get the length of string s
10    for (int i = 0; i < n; i++) {
11        if (i % 2 == 0) {
12            // character of odd index
13            if (s[i] == 'L') {
14                puts("No");
15                return 0;
16            }
17        } else {
18            // character of even index
19            if (s[i] == 'R') {
20                puts("No");
21                return 0;
22            }
23        }
24    }
25
26    puts("Yes");
27    return 0;
28 }
```

Listing 2 Implementation example in Python

```
1 import re
2
3 s = input()
4 p = re.compile(r'^([^L][^R])*[^L]?$')
```

```
5 if p.match(s):  
6     print("Yes")  
7 else:  
8     print("No")
```

C: Attack Survival

The following two operations are equivalent:

- Decrease the score of $N - 1$ players, except for the player who gave the correct answer, by 1.
- Decrease the score of all N people by 1, and then increase the score of the player who gave the correct answer by 1.

Therefore, the fluctuation of the scores can be rephrased as follows:

1. Decrease the scores of all the participants by Q beforehand (so the initial scores are all $K - Q$).
2. For each of Q correct answers, increase the score of the player who gave the answer by 1.

You can implement this straightforward.

D: Powerful Discount Tickets

The rounded down quotient of X divided by 2^Y , $\lfloor \frac{X}{2^Y} \rfloor$, is

$$\lfloor \frac{X}{2^Y} \rfloor = \lfloor \frac{\lfloor \frac{X}{2^{Y-1}} \rfloor}{2} \rfloor$$

This is because, for any positive integer X, b_1, b_2 ,

$$\lfloor \frac{\lfloor \frac{X}{b_1} \rfloor}{b_2} \rfloor = \lfloor \frac{X}{b_1 b_2} \rfloor$$

holds. Here is the proof.

Proof. Let q_1 be the quotient of X divided by b_1 , and r_1 be the remainder, then $\lfloor \frac{X}{b_1} \rfloor = q_1$ holds, and

$$X = b_1 q_1 + r_1 \quad (0 \leq r_1 \leq b_1 - 1)$$

holds. Conversely, the pair of non-negative integer q_1, r_1 such that satisfies the equation above is unique, so for such q_1, r_1 , $\lfloor \frac{X}{b_1} \rfloor = q_1$ holds. Next, Let q_2 be the quotient of $\lfloor \frac{X}{b_1} \rfloor = q_1$ divided by b_2 , and r_2 be the remainder, then it holds that $\lfloor \frac{\lfloor \frac{X}{b_1} \rfloor}{b_2} \rfloor = q_2$ and

$$q_1 = b_2 q_2 + r_2 \quad (0 \leq r_2 \leq b_2 - 1)$$

holds. Therefore,

$$X = b_1 q_1 + r_1 = (b_1 b_2) q_2 + (r_1 + b_1 r_2)$$

holds, and

$$0 \leq r_1 + b_1 r_2 \leq (b_1 - 1) + b_1(b_2 - 1) = b_1 b_2 - 1$$

holds, so $\lfloor \frac{X}{b_1 b_2} \rfloor = q_2$. □

Therefore, instead of buying items one by one and using discount tickets at once for each item, you can assume that "every time you use a discount ticket, choose one item and halve its price (rounded down to the nearest integer)." Therefore, all you have to do is performing operations of choosing the item with the highest price and use a discount ticket to it M times. This is because, assume that you decided not to apply a discount ticket to the current most expensive item (let it be item X), then the total price would be cheaper if you apply the discount ticket to item X rather than to the item you applied it the last time. However, if you naively implement it, it will need a total of $O(NM)$ time, so it won't finish in time. Time for using Priority Queue. If you use priority queue, you can

- add value
- remove value

- get the maximum value

in a $O(\log |Q|)$, $O(\log |Q|)$, $O(1)$ time each, where $|Q|$ is the number of values stored in the priority queue. By making use of it, you can perform each operation (of choosing the most expensive item and halve its price) in a $O(\log N)$ time, so it takes a total of $O(M \log N)$ time, so you can make it in time.

E: Who Says a Pun?

Given a string S of length N , for each $i = 2, 3, \dots, N$, you can find the longest common prefix of S and the substring after the i -th character of S in a $O(N)$ time by applying the algorithm called Z-Algorithm.

Therefore, if you fix $l_1 = 1$, you can find the answer in a $O(N)$ time.

All that left is do the same operations for $S[2 : N], S[3 : N], \dots, S[N : N]$ and you can find the answer for the original problem. It needs a total of $O(N^2)$ time.

As an another solution, if each string comparison are performed in a $O(1)$ time with rolling hash, by performing binary search on self-balancing binary search tree to check existence of hash value, you can find the answer in a total of $O(N(\log N)^2)$ time. Be very careful of hash collisions.

F: Xor Sum 3


Since $(A_1 \oplus A_2 \oplus \dots \oplus A_N) \leq (A_1) + (A_2 \oplus \dots \oplus A_N)$ holds, it can be assumed that you may paint all integers in the same color.

Let $X_{\text{notblue}}, X_{\text{blue}}$ be the XOR of the integers painted in red and the XOR of the integers painted in blue, respectively. Obviously, $X_{\text{notblue}} \oplus X_{\text{blue}} = A_1 \oplus \dots \oplus A_N$ holds, and this value is constant. Write the value in binary notation, and let's call its digits with value 0 as "interesting."

The i -th bits of $X_{\text{notblue}}, X_{\text{blue}}$ are

- $(0, 1)$ or $(1, 0)$ if i -th bit is not interesting, in which case the sum is constant, and
- $(0, 0)$ or $(1, 1)$ if i -th bit is interesting.

Since $2^i > 2^{i-1} + 2^{i-2} + \dots + 2^0$, it appears that it is optimal to greedily letting interesting bit $(1, 1)$ from the higher to the lower, so the problem can be rephrased as follows:



Let A'_i be a sequence, each element's non-interesting bits being removed, and maximize $X'_{\text{notblue}} + X'_{\text{blue}}$. If you add the integer whose all the non-interesting bits are set to 1 to it, that would be the answer for the original problem.

In the problem above $X'_{\text{notblue}} = X'_{\text{blue}}$ holds, so maximizing X'_{blue} will be sufficient. Eventually, solving the following problem is sufficient:

Paint some of A'_1, A'_2, \dots, A'_N in blue, and maximize their XOR.

This can be solved as follows:

1. Regard the sequence A'_i as a $N \times 60$ matrix, and find its row canonical form. Here, 60 denotes the number of bits of A'_i .
2. From the top row to the bottom, let $X'_{\text{blue}} \leftarrow X'_{\text{blue}} \oplus (i\text{-throw})$.

Here the problem was solved, and it can be easily implemented with bit operations ([imple-
mentation example in C](#))