ABC 171 解説

tozangezan, kyopro_friends, sheyasutaka, gazelle, evima, Kmcode

2020 年 6 月 21 日

For International Readers: English editorial will be published in a few days.

A: αlphabet

(解説: evima)

プログラミングの学習を始めたばかりで何から手をつけるべきかわからない方は、まずは「practice contest」(`https://atcoder.jp/contests/practice/`) の問題 A「はじめてのあっとこーだー」をお試しください。言語ごとに解答例が掲載されています。

今回の問題を解くためには、以下の 4 つの要素が必要です。

1. 標準入力からの文字の読み込み
2. 文字 α が大文字か小文字かの判定
3. 上記の判定結果に基づく処理の分岐
4. 標準出力への文字の出力

このうち要素 1, 4 は「はじめての〜」でカバーされたとして (その問題では正確には文字ではなく文字列の入出力が扱われていますが、文字 1 個も長さ 1 の文字列と考えられます)、残りの要素 2, 3 について述べます。

2. 現代の多くの言語には、この判定を行う機能が `isUpperCase`, `islower` などといった名前であらかじめ実装されています。検索エンジンで「[言語名] 大文字 小文字 判定」などと検索することでそのような機能、またはその代わりとなる手法を発見できるはずです。(AtCoder では数十の言語が利用可能であり、ここでそれらについて網羅的に述べるにはこの PDF は狭すぎます。)

3. ほとんどの言語で「if 文」または類似の構文が利用可能であり、また多くの言語に「条件演算子」または類似の要素が存在します。上記と同様に、「[言語名] if」「[言語名] 条件」などと検索することで、その言語で利用可能な構文などの情報が手に入るはずです。

Python での実装例:

```python
a = input()
print('A' if a.islower() else 'a')
```

Ruby での実装例 (2. で述べた `isUpperCase` のような標準機能が存在しない言語の例として):

```ruby
a = gets.chomp
if a == a.upcase
    puts 'A'
else
    puts 'a'
end
```

B: Mix Juice

(解説: evima)

まず、入力されるデータの量が文字 1 個で固定であった問題 A と異なり、今回は価格が入力される果物の数 N が 1 以上 1000 以下と一定ではありません。これに対応するための一般的なキーワードは「for 文」「配列」であり、検索エンジンで「[言語名] for」などと検索することで言語ごとの情報が得られるはずです。ただし、言語によってはこれらのキーワードからややかけ離れた構文や要素も有用であることがあります。

さて、この問題で N 種類の果物から購入する K 種類を選ぶ際は、もちろん価格が低い方から K 種類を選ぶべきです。すなわち、N 個の整数 $p_1, \ldots, p_N$ のうち最も小さいもの、2 番目に小さいもの、...、K 番目に小さいものの和がこの問題の答えで、これをプログラムにどのように求めさせるかが課題となります。結論としては、多くのプログラムに「ソート」、すなわち複数の値を小さい順に並べ替える機能があらかじめ実装されており、これを用いて N 個の整数を小さい順に並べ替えてから先頭の K 個を選択するのが最も簡単です。やはり「[言語名] ソート」などで情報が手に入ります。

C++ での実装例:

```cpp
#include <algorithm>    // sort
#include <iostream>
using namespace std;

int N, K, p[1010];

int main(){
    cin >> N >> K;
    for(int i = 0; i < N; ++i) cin >> p[i];
    sort(p, p + N);
    int ans = 0;
    for(int i = 0; i < K; ++i) ans += p[i];
    cout << ans << endl;
}
```

Python での実装例 (「for 文」からややかけ離れた要素が有用であるような言語の例として):

```
N, K = map(int, input().split())
p = list(map(int, input().split()))
print(sum(sorted(p)[:K]))
```

C: One Quadrillion and One Dalmatians

(解説: evima)

まず、10^{15} という数は現代の我々の計算機にとっても大きく、何らかの処理をこの程度の回数行って実行時間を 2 秒以内に収めることは不可能です。

素直な方針は、問題文で「(以下省略)」となっている部分を計算し、まず番号 N の犬の名前の長さを求めることでしょう。以下、「番号 N の犬の名前」を「名前 N」と書きます。$i = 1, 2, \ldots$ のそれぞれに対し、長さが i であるような名前は 26^i 個存在します。よって、名前 $1, \ldots, 26$ の長さは 1、名前 $26 + 1, \ldots, 26 + 26^2$ の長さは 2、名前 $26 + 26^2 + 1, \ldots, 26 + 26^2 + 26^3$ の長さは 3、$\ldots$ です。$26 + 26^2 + 26^3 + \ldots + 26^{11} = 3817158266467286 > 10^{15} + 1$ より、長さ 11 での計算までに名前 N の長さが判明します。これにより、名前 N の長さが l であると判明したとします。

すると、長さが l であるような 26^l 個の名前の中で、名前 l が辞書順で $N - (26 + 26^2 + \ldots + 26^{l-1})$ (この値を k とします) 番目の名前であることもわかります。残る問題は「長さが l であるような文字列の中で、辞書順で k 番目のものを求めよ」というものです。

この問題の解き方がよくわからなければ、アルファベットが $\mathtt{a}, \mathtt{b}, \ldots, \mathtt{j}$ の 10 文字であったとして考えるとわかりやすいでしょう。例えば、この状況では長さ 3 の名前は $\mathtt{aaa}, \mathtt{aab}, \ldots, \mathtt{aaj}, \mathtt{aba}, \ldots, \mathtt{jjj}$ の $10^3 = 1000$ 個存在します。このうち、例えば辞書順で 246 番目のものは $\mathtt{bde}$ です。このように、$k - 1$ の十進表記での一の位が辞書順で k 番目の名前の最後の文字に、十の位が名前の最後から 2 番目の文字に、$\ldots$、10^i の位が名前の最後から i 番目の文字に対応します ($\mathtt{a}, \mathtt{b}, \ldots, \mathtt{j}$ が $0, 1, \ldots, 9$ に対応しています)。

アルファベットが 26 文字の場合も同様に、$k - 1$ の「二十六進表記」における 26^i の位が名前の最後から i 番目の文字に対応します。これらの位の値の求め方の例は次の通りです。(この手法に関しても、動作原理がよくわからなければ十進表記の場合で考えるとわかりやすいでしょう。)

- 変数 x を $k - 1$ で初期化し、x が 0 となるまで以下の操作を繰り返す。
- x を 26 で割った商が q、余りが r であるとする。この操作が i 回目に実行されたときの r が $k - 1$ の下から i 番目の桁である。x を q で置き換える。

Python での実装例:

```python
N = int(input())
ans = ''
for i in range(1, 99):
    if N <= 26 ** i:
        N -= 1
        for j in range(i):
            ans += chr(ord('a') + N % 26)
            N //= 26
        break
    else:
        N -= 26 ** i
print(ans[::-1])    # reversed
```

(付録) より簡潔な実装例:

```python
N = int(input())
ans = ''
while N > 0:
    N -= 1
    ans += chr(ord('a') + N % 26)
    N //= 26
print(ans[::-1])
```

D: Replacing

(解説: evima)

問題文に書かれている指示にそのまま従うと、最悪の場合にのべ 100 億個の要素を書き換えることになります。この場合、C++ などの高速に動作する言語を使うとして少なくとも 10 秒程度の実行時間が欲しいところですが、この問題の実行制限時間は 2 秒であり、間に合いません。

そこで、この問題で出力することを要求されているものが全要素の和のみであることに着目します。「値が B であるような要素をすべて C に置き換える」という操作 (これを操作 X とします) を行うと、全要素の和は $(C - B) \times$ (操作が行われる直前に値が B であった要素の個数) だけ増加します。よって、数列 A そのものを操作する代わりに、A に値 $1, \ldots, 100000$ が出現する個数 $num_1, \ldots, num_{100000}$ を管理するのが合理的です。操作 X により、全要素の和のみならず各要素の値そのものも変化しますが、この変化も num_B と num_C を操作することにより表現することができます。よって、操作を行い始める前に一度だけ時間をかけて全要素の和と $num_1, \ldots, num_{100000}$ を求めておけば、各操作を定数時間で処理することができます。

E: Red Scarf

(解説: evima)

以下、演算子としての XOR を $\oplus$ と表します。

問題文中の XOR の定義から、3 つ以上の整数の XOR の計算は自由な順序で行えること、例えば任意の 3 つの整数 a, b, c に対して $a \oplus b \oplus c = b \oplus c \oplus a = (a \oplus b) \oplus c = c \oplus (a \oplus b)$ であることがわかります。また、任意の整数 a に対して $a \oplus a = 0$ であることもわかります。以上を組み合わせると、例えば任意の 2 つの整数 a, b に対して $a \oplus b \oplus a = (a \oplus a) \oplus b = 0 \oplus b = b$ であることがいえます。

では、今回の問題の内容に移ります。i 番の猫の整数を b_i とします。このとき、$a_i = b_1 \oplus \ldots \oplus b_{i-1} \oplus b_{i+1} \oplus \ldots \oplus b_N$ です。やや唐突ですが、与えられた N 個の値 $a_1, \ldots, a_N$ の XOR を計算することにします。この値は、上の段落で述べた性質を用いて、次のように整理することができます。

$$
\begin{aligned}
&a_1 \oplus a_2 \oplus \ldots \oplus a_N = (b_2 \oplus \ldots \oplus b_N) \oplus (b_1 \oplus b_3 \oplus \ldots \oplus b_N) \oplus \ldots \oplus (b_2 \oplus \ldots \oplus b_{N-1}) \\
&= (\underbrace{b_1 \oplus \ldots \oplus b_1}_{N-1 \text{ 個}}) \oplus (\underbrace{b_2 \oplus \ldots \oplus b_2}_{N-1 \text{ 個}}) \oplus \ldots \oplus (\underbrace{b_N \oplus \ldots \oplus b_N}_{N-1 \text{ 個}}) \\
&= b_1 \oplus b_2 \oplus \ldots \oplus b_N \ (\because N \text{ は偶数})
\end{aligned}
$$

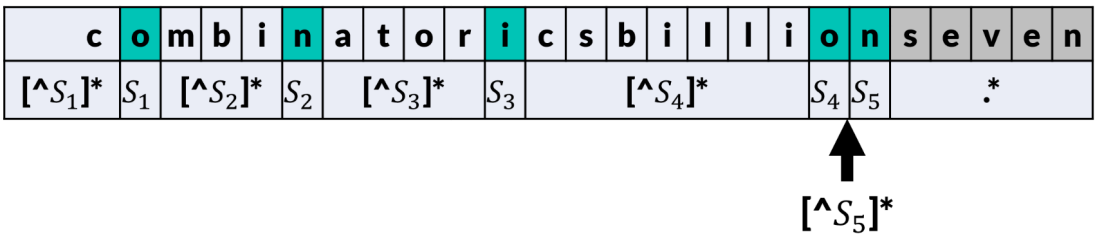
したがって、$a_1 \oplus \ldots \oplus a_N = b_1 \oplus \ldots \oplus b_N$ であることがわかりました。さらに、この値を S とすると、任意の i に対して次が成り立ちます。

$$
\begin{aligned}
&S \oplus a_i = (b_1 \oplus \ldots \oplus b_N) \oplus (b_1 \oplus \ldots \oplus b_{i-1} \oplus b_{i+1} \oplus \ldots \oplus b_N) \\
&= (b_1 \oplus b_1) \oplus \ldots \oplus (b_{i-1} \oplus b_{i-1}) \oplus b_i \oplus (b_{i+1} \oplus b_{i+1}) \oplus \ldots \oplus (b_N \oplus b_N) = b_i
\end{aligned}
$$

よって、任意の i に対して $b_i = S \oplus a_i$ であることがわかり、問題が解けました。問題の制約に「与えられた情報と整合するようなスカーフ上の整数の組合せが存在する」、「出力」セクションに「解が複数存在する場合、どれを出力しても構わない」とありましたが、この解法より、実際には任意の N 個の整数 $a_1, \ldots, a_N$ の組合せに対してそれに整合するようなスカーフ上の整数 $b_1, \ldots, b_N$ の組合せがちょうど一通り存在することがわかります。

F: Strivore

条件を満たす文字列は長さ $|S|+K$ であり，次のような形をしています.



ここで，"[^S_i]*" は S_i でない英小文字からなる長さ 0 以上の文字列を，".*" は任意の英小文字からなる長さ 0 以上の文字列を指します．英小文字の数は 26 であることに注意してください.

$|S|+K$ 文字のうちの $S_1 \sim S_N$ の位置を固定してみたときの，S_i 以外の位置の文字を選ぶ通り数を考えると，文字は互いに影響を及ぼさず独立に選べるので単純に各位置の候補数を掛け合わせればよく，通り数は $25^{K-(S_N\text{以降の文字数})} \times 26^{(S_N\text{以降の文字数})}$ となります.

また，S_N の位置を固定したとき，$S_1 \sim S_{N-1}$ の位置を選ぶ通り数は ${}_{|S|+K-(S_N\text{以降の文字数})-1}C_{N-1}$ となります (「二項係数」で調べるとこの式の意味が分かります).

したがって，S_N の位置を全て試し，各々における [$S_1 \sim S_{N-1}$ の位置] × [S_i 以外の文字の選び方] を足し合わせることで，求める値が得られます.

適切に前処理をして，時間・空間ともに $\Theta(|S|+K)$ で解けます.