ABC 171 解説

tozangezan, kyopro_friends, sheyasutaka, gazelle, evima, Kmcode

2020 年 6 月 21 日

For International Readers: English editorial starts on page 10.

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:

```
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):

```
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、多くの言語に「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 N 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 $i+1$ 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。

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

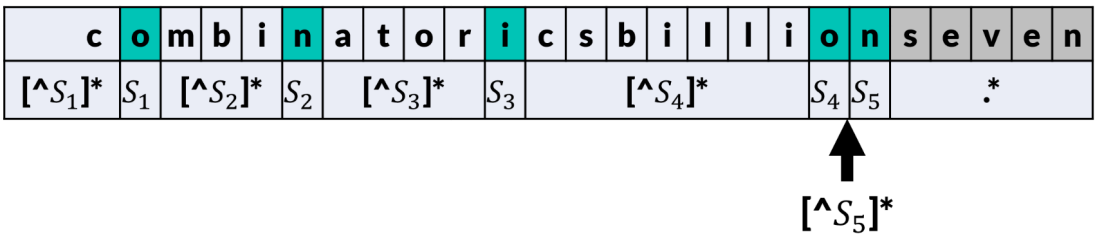
したがって、$a_1 \oplus \ldots \oplus a_N = b_1 \oplus \ldots \oplus b_N$ であることがわかりました。さらに、この値を S とすると、任意の i に対して次が成り立ちます。

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

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 [$S_1 \sim S_{N-1}$ の位置の選び方] × [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．

A: αlphabet

(Editorial: evima)

If you are a beginner of studying programming and have no idea what to start with, then try solving "Welcome to AtCoder", Problem A from the "practice contest" (`https://atcoder.jp/contests/practice/`). There you can find sample codes for each language.

To solve this problem, the following four elements are required.

1. Read a character for standard input
2. Check if character α is uppercase or lowercase
3. Conditional branch according to the result above
4. Output a character to standard output

As elements 1 and 4 are covered by the "Welcome to —" (in which, precisely speaking, it is explained the way of input and output of string, but a single character can considered to be a string of length 1), we will explain the rest, elements 2 and 3.

2. In most modern languages, the functionality of such judgement is implemented out-of-the-box, whose name is such like `isUpperCase`, `islower`. You can find such functionality or alternative methods by searching like "[language name] check case of letter". (Tens of languages are available in AtCoder, so this PDF is too narrow to explain comprehensively about them.)

3. In most languages, "if statement" or similar syntax is available, and also most languages have "conditional operators" or similar one. Just like above, you can obtain some information about the syntax available in the language by searching "[language name] if" or "[language name] conditional branch".

Sample Code in Python:

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

Sample Code in Ruby (as an example of languages which do not have standard functions like `isUpperCase` as mentioned in 2.):

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

B: Mix Juice

(Editorial: evima)

First, unlike Problem A, in which the amount of input data is fixed to a single character, this time, N, the number of fruits whose prices are provided, is not constant. General keywords to deal with them are "for statements" and "arrays," and you will obtain the information for each language by searching like "[language name] for". Note that, however, rather different syntax or elements might be useful sometimes.

In this problem, when choosing K kinds of fruits to buy out of N kinds of fruits, choosing K kinds of cheapest fruits is optimal of course. That is, the answer of this problem is the sum of the smallest, the second smallest, ..., and the K-th smallest ones out of N integers $p_1, \ldots, p_n$, and the task is how to instruct the program to find them. In conclusion, most languages have functionalities of "sorting," that is, the function of arranging multiples values in the ascending order, so the easiest way is to use this function to sort N integers in the ascending order and choosing the first K elements. Still you can obtain information by "[language name] sort".

Sample code in C++:

```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;
}
```

Sample code in Python (as an example of languages in which rather different elements to “for statements” are useful):

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

C: One Quadrillion and One Dalmatians

(Editorial: evima)

First, the number 10^{15} is so large even for our modern machine that it is impossible to perform some operation this number of times within two seconds.

A simple way is to find the length of the name of the dog numbered N by calculating the "and so on"-part found in the problem statement. By "Name X", we denote the name of the dog numbered N. For each $i = 1, 2, \ldots$, there are 26^i names of length i. Therefore, Names $1, \ldots, 26$ have lengths 1, Names $26 + 1, \ldots, 26 + 26^2$ have lengths 2, Names $26 + 26^2 + 1, \ldots, 26 + 26^2 + 26^3$ have lengths 3, 3, ... and so on. Since $26 + 26^2 + 26^3 + \ldots + 26^{11} = 3817158266467286 > 10^{15} + 1$, by the time we calculate the length 11, we can find the length of Name N. We assume that Name N appeared to have length l.

Then we can see that, Name N is the $N - (26 + 26^2 + \ldots + 26^{l-1})$ (let the value be k)-th name among 26^l names of lengths l. Now we want to find "the k-th among the strings of length l."

If yo are not sure how to answer this question, you can simplify it by considering the situation where only 10 alphabets,`a`, `b`, ... and `j`, are available. For example, under that assumption, the number of the name of length 3 is $10^3 = 1000$: `aaa`, `aab`, ..., `aaj`, `aba`, ... and `jjj`. Among them, for instance, the 246-th one is `bde`. Likewise, in the decimal representation of $k - 1$, the ones' place corresponds to the last letter, the tens' place corresponds to the second last letter, and the 10^is' place corresponds to the $i + 1$-th last letter (`a`, `b`, ..., `j` correspond to $0, 1, \ldots, 9$).

Similarly, when 26 kinds of alphabets are used, in the notation of base-26, the 26^is' place corresponds to the $i + 1$-th last letter. You can find each digit by the following procedures for example:

- Initialize variable x with $k - 1$, and repeat until x becomes 0.
- Let q and r be the quotient and remainder of x divided by 26, respectively. The i-th result of r in this operation is the i-th last digit of $k - 1$. Replace x with q.

Sample code in Python:

```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
```

(Appendix) a simpler implementation example:

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

D: Replacing

(Editorial: evima)

If you naively follow the directions in the problem statement, you will have to replace as many as 10 billion elements in total. In this case, even if we use fast languages like C++, we want about 10-second execution time, but the time limit for this problem is 2 seconds, so it won't finish in time.

Note that we are only asked to output the sum of all elements. After performing the operation of "replacing every element whose value is B with C" (we denote this operation by X), the total sum of elements increases by $(C - B) \times$ (the number of elements whose values are B before the opration). Therefore, instead of manipulating sequence A itself, it is more reasonable to manage the number of appearances of $1, \ldots, 100000$ in A. By the operation X, not only the sum of all elements but also the value of each element itself also changes, but this changes can also be represented by operating num_B and num_C. Therefore, by taking time for calculating the sum of all the elements and $num_1, \ldots, num_{100000}$ only once before start to perform the operations, each operation can be performed in a constant time.

E: Red Scarf

(Editorial: evima)

Hereinafter $\oplus$ denotes XOR as an operator.

By the definition of XOR in the problem statement, we can see that calculation of XOR of 3 or more integers can be done in an arbitrary order; for example, for any three integers a, b, c, it holds that $a \oplus b \oplus c = b \oplus c \oplus a = (a \oplus b) \oplus c = c \oplus (a \oplus b)$. We can also see that, $a \oplus a = 0$ for all integer a. Together, we can claim that $a \oplus b \oplus a = (a \oplus a) \oplus b = 0 \oplus b = b$.

Now let's move on to the original problem. Let b_i b the integer of i-th cat. Here, $a_i = b_1 \oplus \ldots \oplus b_{i-1} \oplus b_{i+1} \oplus \ldots \oplus b_N$. It's a bit kind of sudden, but we will calculate the XOR of the given N values $a_1, \ldots, a_N$. This value can be transformed as followed:

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

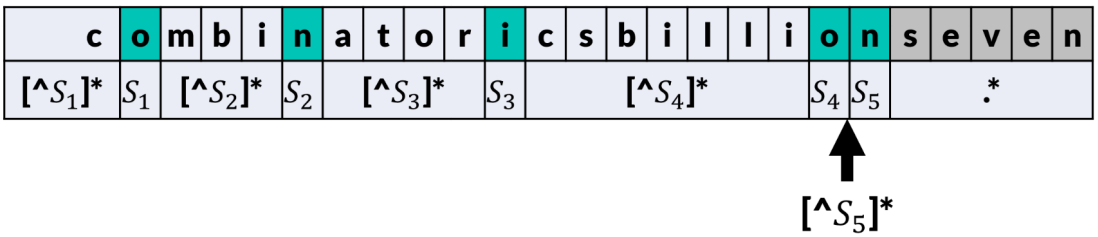
Therefore, we could see that $a_1 \oplus \ldots \oplus a_N = b_1 \oplus \ldots \oplus b_N$. Moreover, let this value be B, then for all i the following equations hold:

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

Therefore, for each i it appears that $b_i = S \oplus a_i$, so the problem has been solved. There was a constraints in the problem statement that "there exists a combination of integers on the scarfs that is consistent with the given information," and the "Output" section states that "if there are multiple possible solutions, you may print any of them;" however, by this solution, for any combination of N integers $a_1, \ldots, a_N$, there always exists a unique combination of integers on the scarfs $b_1, \ldots, b_N$ that satisfies it.

F: Strivore

Each string that satisfies the given conditions is of length $|S| + K$, which has the following form:



Here, "[^S_i]*" denotes a string consisting of English alphabets other than S_i of 0 or more length, and ".*" denotes a string consisting of any English alphabets of 0 or more length. Note that there are 26 kinds of English alphabets.

Assume that the positions of $S_1 \sim S_N$ among $|S| + K$ letters are fixed. Then, the coices of alphabets at the positions other than those of S_I are independent of each other, so the number of such combinations can be calculated by simple multiplications: there are $25^{K-(\text{the number of letters before } S_N)} \times 26^{(\text{the number of letters after } S_N)}$ combinations.

Also, when the positions S_N is fixed, the number of combinations of positions of $S_1 \sim S_{N-1}$ is ${}_{|S|+K-(\text{The number of letters after } S_N)-1}C_{N-1}$ (search for "binomial coefficient" for the meaning of this formula.

There, try all possible positions of S_N, and for each of them, calculate [the choices of positions of $S_1 \sim S_{N-1}$] $\times$ [the choices of alphabets other than S_i], then the sum is the desired answer.

After a proper precalculation, this problem can be solved in a total of $\Theta(|S| + K)$.