ABC 085 Editorial (Japanese)

問題・解説: evima

2018 年 1 月 7 日

For International Readers: English editorial starts on page 5.

A: Already 2018

この問題を解くには、以下の手順を踏む必要があります。

0. （言語によっては不要）文字列変数 S を宣言する。
1. 標準入力から S を文字列として受け取る。
2. S の 4 文字目を '8' に置き換える（直接的または間接的に）。
3. 得た文字列を標準出力に出力する。

ステップ 2. に関しては、言語によって事情が異なります。C++ のような言語では単に `S[3] = '8';` などとすれば済みますが、Python のような言語では文字列中の文字を直接書き換えることが許されないこともあります。このような場合、文字列を一旦別のデータ構造（リスト）に変換する手もありますが、S の末尾の 2 文字を `"2018/01/"` と連結するなど、別の方針も考えられます。

C++, Python での実装例:

```cpp
#include <iostream>
#include <string>
using namespace std;

int main(){
    string S;
    cin >> S;
    S[3] = '8';
    cout << S << endl;
}
```

```python
print("2018/01/" + input()[-2:])
```

B: Kagami Mochi

問題文は遠回しな言い方をしていますが、要するに「$d_1, d_2, \ldots, d_N$ には何種類の異なる値があるか？」と聞いています。

方針 1: 「自分でやる」

配列 `flag` を用意し、はじめ `flag[1] = flag[2] = ... = flag[100] = 0` とします。d_1 から d_N までのそれぞれの値 `d` に対して `flag[d] = 1` とすると、`flag` の要素の和が答えとなります。

C++ での実装例:

```cpp
#include <iostream>
using namespace std;
int N, flag[101];   // Global variables are initially all 0.
                    // Also note that we need [101], not [100]!
int main(){
    cin >> N;
    for(int i = 0; i < N; ++i){
        int d;
        cin >> d;
        flag[d] = 1;
    }
    int ans = 0;
    for(int i = 0; i < N; ++i){
        ans += flag[i];
    }
    cout << ans << endl;
}
```

方針 2: 「set を使う」

多くの言語には重複を省いた集合を表すデータ構造が実装されており、たいてい set か似たような名前を持ちます。$d_1, d_2, \ldots, d_N$ を入れた set の要素数がそのまま答えとなります。

Python での実装例（少し遊んでいます）:

```python
print(len(set(input() for i in range(int(input())))))
```

C: Otoshidama

袋の中身を「10000 円札 x 枚、5000 円札 y 枚、1000 円札 z 枚」とします。当然ですが重要な点として、x, y, z はどれも 0 以上 N 以下の整数でなければなりません。

もっとも単純な方針は、三重ループで x, y, z の値の決め方をすべて試し、$x+y+z=N, 10000x+5000y+1000z=Y$ が成立するか確かめるものですが、$N \leq 2000$、実行制限時間 2 秒という設定で $\mathrm{O}(N^3)$ 時間の解法は分が悪いです（間に合わせることも不可能ではありません）。しかし、x と y の値が決定されると z の値が $z=N-x-y$ に限られることを利用すると、三重ループではなく二重ループですべての可能性を列挙することができます（ただし、z が負になるケースを取り除く必要があります）。これらのすべての可能性に対して $10000x+5000y+1000z=Y$ が成り立つか調べることで、問題を $\mathrm{O}(N^2)$ 時間で解くことができます。

なお、$\mathrm{O}(N)$ 時間や $\mathrm{O}(1)$ 時間の解法も考えられますが、コンテスト中に追求する必要はないでしょう。

D: Katana Thrower

問題文のルールでは、刀を投げてしまうとその刀を振ることはできなくなります。しかし、すでに投げてしまった刀も振ることができるというルールに変更しても問題の答えは変わりません。これを証明します。以下、問題文のルールを「旧ルール」、ここで定義したルールを「新ルール」と呼びます。

新ルールで T 回攻撃して合計 D ダメージを与えることが可能であると仮定します。そのような攻撃手順の一つで、それぞれの刀 i $(1 \leq i \leq N)$ を x_i 回振って y_i 回投げるとします $(0 \leq x_i, 0 \leq y_i \leq 1)$。このとき、旧ルールでも刀 1 を x_1 回振ってから y_1 回投げ、刀 2 を x_2 回振ってから y_2 回投げ、...、刀 N を x_N 回振ってから y_N 回投げることで、同じく T 回の攻撃で合計 D ダメージを与えることが可能です。よって、新ルールで T 回の攻撃で魔物を退治できるとき、旧ルールでも T 回の攻撃で魔物を退治できます。逆に、旧ルールで T 回の攻撃で魔物を退治できるとき、新ルールでもまったく同じ攻撃手順で T 回の攻撃で魔物を退治できます。以上より、新ルールを採用しても問題の答えは変わりません。

以後、新ルールの下で問題を考えます。新ルールでは、刀を振ることと投げることは直接関連せず、我々は攻撃力 a_i の「振り刀」N 本と攻撃力 b_i の「投げ刀」N 本、合計 $2N$ 本の武器を持っていると考えることができます。(攻撃力 a の振り刀は投げることはできないが何度でも振ることができ、そのたびに a ダメージを与えます。攻撃力 b の投げ刀は振ることはできないが一度だけ投げることができ、投げると b ダメージを与えます。) このときの最適な攻撃手順は以下の通りです。

0. $a_1, \ldots, a_N$ のうち最大の値を $a_{\max}$ とし、攻撃力が $a_{\max}$ 以上の投げ刀を「強い投げ刀」とする。
1. 強い投げ刀を攻撃力が高い順に投げていく。
2. 強い投げ刀を使い果たしたら、攻撃力 $a_{\max}$ の振り刀を振り続ける。

この戦略は、はじめに $b_1, \ldots, b_N$ を降順にソートしておき (言語の標準ライブラリを使うべきです)、ステップ 2. で割り算を使うことで、全体で $\mathrm{O}(N \log(N))$ 時間でシミュレートすることができます。

ABC 085 Editorial (English)

Problems and editorial by: evima

January 7, 2018

A: Already 2018

To solve this problem, you need to go through the procedure below:

0. (Unnecessary for some languages) Declare a string variable S.
1. Receive S as a string from Standard Input.
2. Replace the fourth character in S with `'8'` (directly or indirectly).
3. Print the obtained string to Standard Output.

For step 2., situations are different depending on language. In languages such as C++, writing `S[3] = '8';` or similar code is fine, but in some languages such as Python, rewriting a character in a string is prohibited. In this case, a workaround is to convert the string to another data structure (list) temporarily, but there are also other paths, such as appending the last two characters in S to `"2018/01/"`.

Sample implementations in C++ and Python:

```cpp
#include <iostream>
#include <string>
using namespace std;

int main(){
    string S;
    cin >> S;
    S[3] = '8';
    cout << S << endl;
}
```

```python
print("2018/01/" + input()[-2:])
```

B: Kagami Mochi

The statement is beating around the bush, but in short, the problem is asking you: "how many different values are there among $d_1, d_2, \ldots, d_N$?"

Approach 1: Do it yourself

Prepare an array `flag`, and initially set it to `flag[1] = flag[2] = ... = flag[100] = 0`. Then, for each value `d` from d_1 to d_N, set `flag[d] = 1`. Now, the sum of the elements of `flag` is the answer.

Sample implementation in C++:

```cpp
#include <iostream>
using namespace std;
int N, flag[101];   // Global variables are initially all 0.
                    // Also note that we need [101], not [100]!
int main(){
    cin >> N;
    for(int i = 0; i < N; ++i){
        int d;
        cin >> d;
        flag[d] = 1;
    }
    int ans = 0;
    for(int i = 0; i < N; ++i){
        ans += flag[i];
    }
    cout << ans << endl;
}
```

Approach 2: Use set

Many languages implement data structures representing sets without duplication, usually named "set" or similarly. Insert $d_1, d_2, \ldots, d_N$ into a set, and the number of elements itself is the answer.

Sample implementation in Python (I'm playing a bit):

```python
print(len(set(input() for i in range(int(input())))))
```

C: Otoshidama

Let the contents of the envelope be "x 10000-yen bills, y 5000-yen bills and z 1000-yen bills". Naturally but importantly, x, y and z all have to be integers between 0 and N.

The simplest approach is to try all ways to set the values of x, y and z in three nested loops and check if $x + y + z = N, 10000x + 5000y + 1000z = Y$ hold, but under the constraint $N \leq 2000$ and the execution time limit of 2 seconds, odds are against $\mathrm{O}(N^3)$ time solutions (not impossible to make it in time, though). Notice that, however, when we fix the values of x and y, the value of z will be limited to $z = N - x - y$. Then, we can enumerate all possibilities in two nested loops, not three (but we need to remove the case where z is negative). By checking if $10000x + 5000y + 1000z = Y$ holds for each of these possibilities, we can solve the problem in $\mathrm{O}(N^2)$ time.

There are also $\mathrm{O}(N)$ time and $\mathrm{O}(1)$ time solutions, but no need to find them during the contest.

D: Katana Thrower

Under the rules in the statement, when you throw a katana, you will no longer be able to wield that katana. However, the answer does not change if we change the rule to allow wielding a katana that you have already thrown. We will now prove it. Below, we will call the rules in the statement the "old rules", and the rules defined here the "new rules".

Suppose that we can deal a total of D damage in T attacks under the new rules. Assume that, in one such sequence of attacks, each katana i ($1 \leq i \leq N$) is wielded x_i times and thrown y_i times ($0 \leq x_i, 0 \leq y_i \leq 1$). Then, we can also deal a total of D damage in T attacks under the old rules, by wielding katana 1 x_1 times, then throwing katana 1 y_1 times, then wielding katana 2 x_2 times, then throwing katana 2 y_2 times, ..., then wielding katana N x_N times, then throwing katana N y_N times. Thus, if we can vanish the monster in T attacks under the new rules, we can also vanish the monster in T attacks under the old rules. On the other hand, if we can vanish the monster in T attacks under the old rules, we can also vanish the monster in T attacks under the new rules by performing exactly the same sequence of attacks. Therefore, the answer does not change by adopting the new rules.

We will consider the problem under the new rules from now on. Under the new rules, wielding katana and throwing them is not directly related, and we can assume that we have $2N$ weapons, N "wielding katana" of attack power a_i and N "throwing katana" of attack power b_i. (A wielding katana of attack power a_i cannot be thrown but can be wielded any number of times, and deals a damage each time we wield it. A throwing katana of attack power b_i cannot be wielded but can be thrown only once, and deals b damage when we throw it.) The optimal strategy now is:

0. Let $a_{\max}$ be the maximum value among $a_1, \ldots, a_N$, and "strong throwing katana" be the katana whose attack powers are $a_{\max}$ or above.
1. Throw strong throwing katana one at a time in decreasing order of attack power.
2. When we have no more throwing katana, repeat wielding the wielding katana of attack power $a_{\max}$.

This strategy can be simulated in $\mathrm{O}(N \log(N))$ time in total, by sorting $b_1, \ldots, b_N$ in decreasing order (the standard library in your language should be used) in the beginning and use division in step 2.