ABC 063 / ARC 075 Editorial (Japanese)

問題・解説: evima

2017 年 6 月 3 日

For international readers: English Editorial starts at page 6.

A: Restricted

問題の要求に素直に従うことが求められています。以下に、C++ での実装の例を示します。

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

int main(){
    int A, B;
    cin >> A >> B;
    if(A + B >= 10){
        cout << "error" << endl;
    }else{
        cout << A + B << endl;
    }
}
```

B: Varied

いくつかの方針が考えられます。

- ループを二重に組み合わせ、$1 \leq i < j \leq |S|$ であるような添字のペア (i, j) すべてについて S_i と S_j が異なるか確認する（$|S|$ が大きいと遅くなりますが、今回は $|S| \leq 26$ であり問題ありません[*1]）
- S に含まれる文字をアルファベット順にソートし、同じ文字が二つ隣り合わないか確認する
- 26 種類の文字それぞれについて S での出現回数を数え、どの文字の出現回数も 1 以下であるか確認する

以下に、最初の方針の C++ での実装の例を示します。

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

int main(){
    string S;
    cin >> S;
    int N = S.size();
    string ans = "yes";
    for(int i = 0; i < N; ++i){
        for(int j = i + 1; j < N; ++j){
            if(S[i] == S[j]){
                ans = "no";
            }
        }
    }
    cout << ans << endl;
}
```

[*1] この問題の制約に反しますが、仮に $|S| = 10^5$ であったとすると、実行制限時間の 2 秒に間に合わないでしょう。余談ですが、鳩の巣原理より $|S| > 26$ のときに直ちに no と出力すれば、そのような長さの入力に対しても実行が間に合います。

C: Bugged

方針 1: 「正攻法」

すべての問題の配点の合計を S とします。

- S が 10 の倍数でない場合、S が画面に表示される最大の成績です。
- S が 10 の倍数である場合、画面に 0 でない成績を表示させるためには一問以上の問題で不正解する必要があります。もし、配点が 10 の倍数でないような問題が一問以上存在するなら、それらのうち配点が最も低い一問のみで不正解することで、画面に表示される最大の成績を得られます。配点が 10 の倍数でないような問題が一問も存在しない場合は、画面に 0 が表示されることを防げません。

方針 2: 「オーバーキル」

動的計画法を用いて、「解いた問題の配点の合計」として可能性のあるすべての値を時間計算量 $\mathrm{O}(N\sum s_i)$ で列挙することができます（詳細は省きます)。これは、答えを求めるのに十分過ぎる情報です。

D: Widespread

「残りの体力が最も多い魔物を中心に爆発を起こすことを繰り返す」のは最適な戦略ではありますが、この戦略を素直にシミュレートしてしまうと、例えば体力 10^9 の魔物が 10^5 体存在して $A = 2, B = 1$ の場合に長大な時間を要します。

整数 T に対し、 $\mathit{enough}(T)$ を「T 回以内の爆発ですべての魔物を消すことは可能か？」という問いの答え (Yes または No) とします。このとき、求めたい答えは $\mathit{enough}(T) = \mathrm{Yes}$ であるような T の最小値です。enough は単調性を持つ ($\mathit{enough}(X) = \mathrm{Yes}$ であれば $X \leq Y$ のとき $\mathit{enough}(Y) = \mathrm{Yes}$) ため、この最小値を二分探索[*2]で求めることができます。

定められた T の値に対して $\mathit{enough}(T)$ を判定するには、爆発を起こすことを以下のように捉え直すとよいでしょう:「すべての魔物の体力を B ずつ減らし、魔物を一体選んでその体力をさらに $A - B$ 減らす」。
すると、T 回の爆発を起こすことは、すべての魔物の体力を $B \times T$ ずつ減らし、その上で T 回にわたって一体の魔物の体力を $A - B$ 減らすこと（以下、この行為を「追加ダメージを与える」と表現します）と同等です。すべての魔物を消すには、体力が $B \times T$ より高い魔物 i それぞれに対して追加ダメージを $\lceil (h_i - B \times T)/(A - B) \rceil$ 回与える必要があり（$\lceil x \rceil$ は x 以上の最小の整数を表します)、この回数の合計が T 以下であれば $\mathit{enough}(T) = \mathrm{Yes}$、合計が T を超える場合は $\mathit{enough}(T) = \mathrm{No}$ と判定できます。

以上により、一つの T の値に対する $\mathit{enough}(T)$ の判定を時間計算量 $\mathrm{O}(N)$ で行うことができ、この解法の全体の時間計算量は $\mathrm{O}(N \log(h_{\max}/B))$ となります。

[*2] 二分探索自体の説明は省きます。例えば、日本語版 Wikipedia の当該記事に詳細な記述があります。

E: Meaningful Mean

問題文中の表記から a の添え字の範囲をずらし、この解説では $a = \{a_0, a_1, \ldots, a_{N-1}\}$ とします。

a の空でない連続する部分列 $\{a_l, a_{l+1}, \ldots, a_{r-1}\}$ $(0 \leq l < r \leq N)$（これも問題文中の表記と異なり、a_r が含まれないことに注意してください）の平均が K 以上であることは、次のように言い換えられます。

$$\sum_{i=l}^{r-1} a_i \geq (r-l)K \quad \Leftrightarrow \quad \sum_{i=0}^{r-1} a_i - \sum_{i=0}^{l-1} a_i \geq rK - lK \quad \Leftrightarrow \quad \sum_{i=0}^{r-1} a_i - rK \geq \sum_{i=0}^{l-1} a_i - lK$$

ここで $b_j = \displaystyle\sum_{i=0}^{j-1} a_i - jK$ $(0 \leq j \leq N)$ とおくと、この条件は単に $b_r \geq b_l$ と書けます。b_j $(0 \leq j \leq N)$ すべての値は容易に $\mathrm{O}(N)$ 時間で求めることができ、残るは $b_r \geq b_l$ であるようなペア (l, r) $(0 \leq l < r \leq N)$ の個数を数えることのみです。これには、何らかのデータ構造を用いることが必要となるでしょう。以下に、最も簡単と思われる方法を述べます。

まず、$N+1$ 個の値 $b_0, b_1, \ldots, b_N$ を、大小関係を保ったまま 0 から N の範囲の値に圧縮し、その値を $c_0, c_1, \ldots, c_N$ とします（ソートと二分探索を用います）。そして、Binary Indexed Tree[*3]を用いて $0, 1, \ldots, N$ のそれぞれの値の出現回数を保持します。これにより、$i = 0, 1, \ldots, N$ のそれぞれに対し、$c_0, c_1, \ldots, c_{i-1}$ に c_i 以下の値が何回出現したかをそれぞれ $\mathrm{O}(\log(N))$ 時間で計算することができ、合計 $\mathrm{O}(N \log(N))$ 時間で求めるべきペアの個数を計算することができます。

[*3] Binary Indexed Tree 自体の詳細な説明は省きますが、このデータ構造は長さ n の数列を特殊な形式で保持し、二種類の操作「数列の指定された位置の要素に指定された値を加算する」「数列の先頭から指定された位置までの要素の和を求める」をそれぞれ時間計算量 $\mathrm{O}(\log(n))$ で行うことができます。より詳細な情報は、例えば英語版 Wikipedia の記事 “Fenwick Tree” にあります。

F: Mirrored

与えられた式を $rev(N) - N = D$ とみなします。N の十進表記における桁数を L とし、$N = \sum_{i=0}^{L-1} 10^i n_i$ $(0 \leq n_i \leq 9, n_{L-1} \neq 0)$ と表記すると、$rev(N) - N$ は次のように変形できます。

$$rev(N) - N = \sum_{i=0}^{L-1} 10^{L-1-i} n_i - \sum_{i=0}^{L-1} 10^i n_i = \sum_{i=0}^{L-1} (10^{L-1-i} - 10^i) n_i = \sum_{i=0}^{\lfloor L/2 \rfloor - 1} (10^{L-1-i} - 10^i)(n_i - n_{L-1-i})$$

ここで $d_i = n_i - n_{L-1-i}$ $(0 \leq i \leq \lfloor L/2 \rfloor - 1)$ とおくと、

$$rev(N) - N = \sum_{i=0}^{\lfloor L/2 \rfloor - 1} (10^{L-1-i} - 10^i) d_i$$

この式の右辺を $f(L, d)$ とおきます。与えられた式を満たす N の個数を求めることは、$f(L, d) = D$ であるような L と -9 以上 9 以下の整数の列 $d_0, d_1, \ldots, d_{\lfloor L/2 \rfloor - 1}$ の組み合わせを列挙することとほぼ同等です(一つの d_i の列に対し、それに対応する N が複数個存在することを考慮する必要がありますが)。ここで、$0 \leq i < \lfloor L/2 \rfloor - 1$ であるような任意の i に対して次の式が成立します。

$$10^{L-1-i} - 10^i > \sum_{j=i+1}^{\lfloor L/2 \rfloor - 1} ((10^{L-1-j} - 10^j) \cdot 9) + 10^{L - \lfloor L/2 \rfloor}$$

(この式の「意味」は次の通りです:「ある i $(\neq \lfloor L/2 \rfloor - 1)$ に対して $d_i = 1$ とすると、$j > i$ であるようなすべての j に対して $d_j = -9$ としても、$d_i = 1$ による $f(L, d)$ への正の方向への影響が残り、その残りは $10^{L - \lfloor L/2 \rfloor}$ より大きい」) この式から、次の二点がいえます。

- D の十進表記における桁数を L_D とすると、$L > 2L_D$ で $f(L, d)$ が正のとき、$f(L, d)$ が D より大きくなることがわかります。したがって、L の値として L_D 以上 $2L_D$ 以下のすべての値を検討すれば十分です。
- L の値を定めたとき、$f(L, d) = D$ であるような整数列 $d_0, d_1, \ldots, d_{\lfloor L/2 \rfloor - 1}$ を列挙するために(そして与えられた式を満たす N の個数を数えるために)、d_0 から順に値を定めていく深さ優先探索を行うことを考えます。d_{i-1} までの値をすでに定めているとき、d_i の値として検討する必要がある候補は、$dif = N - \sum_{j=0}^{i-1} ((10^{L-1-j} - 10^j) \cdot d_j)$ として、$(10^i - 10^{L-1-i}) d_i \leq dif$ であるような最大の値と、$(10^i - 10^{L-1-i}) d_i > dif$ であるような最小の値の高々二つのみであることがわかります。(直感的に述べると、探索の途中で $f(L, d)$ の「途中まで」の値が D からあまり離れてしまうとそれ以降「戻ってくる」ことはできないため、D の「近くから離れる」べきではありません。) したがって、L の値を定めたとき、与えられた式を満たす N であって桁数が L であるものの個数を時間計算量 $\mathrm{O}(2^{\lfloor L/2 \rfloor})$ で求めることができます。

以上の二点より、求めるべき N の個数を時間計算量 $\mathrm{O}(\sum_{i=L_D}^{2L_D} 2^{\lfloor i/2 \rfloor}) = \mathrm{O}(2^{L_D})$ で求めることができます。

ABC 063 / ARC 075 Editorial (English)

Problems and editorial by evima

June 3, 2017

A: Restricted

It is required to do exactly what is told. C++ implementation follows:

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

int main(){
    int A, B;
    cin >> A >> B;
    if(A + B >= 10){
        cout << "error" << endl;
    }else{
        cout << A + B << endl;
    }
}
```

B: Varied

There are several solutions:

- Nest two loops and check if S_i and S_j differs for all pairs of indices (i, j) such that $1 \leq i < j \leq |S|$ (slower if $|S|$ is larger, but fine in this problem since $|S| \leq 26$[*1])
- Sort the characters in S in alphabetical order, and check if no two adjacent letters are the same
- For each of the 26 letters, count the occurrences in S, and check if no letter appears more than once

C++ implementation of the first solution follows:

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

int main(){
    string S;
    cin >> S;
    int N = S.size();
    string ans = "yes";
    for(int i = 0; i < N; ++i){
        for(int j = i + 1; j < N; ++j){
            if(S[i] == S[j]){
                ans = "no";
            }
        }
    }
    cout << ans << endl;
}
```

[*1] The constraints in this problem does not allow this, but if $|S| = 10^5$, it would not fit within the execution time limit of 2 seconds. A little off-topic, but if we immediately print "no" when $|S| > 26$ by pigeonhole principle, it will run in time.

C: Bugged

Solution 1: "Straight-forward"

Let S be the total points allocated to all the problems.

- If S is not a multiple of 10, S is the maximum value we seek.
- If S is a multiple of 10, in order to display a value other than 0, we need to incorrectly answer at least one question. If there is at least one question with a score which is not a multiple of 10, incorrectly answer one of the problems with the lowest score allocated among them, and we can obtain the maximum grade that can be correctly displayed. If there is no such question, there is no way to prevent the system from displaying 0.

Solution 2: "Overkill"

By Dynamic Programming, we can enumerate all the possible candidates of "the total score allocated to correctly answered problems" in $O(N \sum s_i)$ time (we will omit the detail). This is more than enough to find the answer.

D: Widespread

It is indeed optimal to repeatedly cause an explosion centered at the monster with the most health remaining, but straight-forward simulation of this strategy will take a long time when, for example, there are 10^5 monsters with 10^9 health each and $A = 2, B = 1$.

For an integer T, let *enough*(T) be the answer to the question "is it possible to vanish all the monster in at most T explosions?": either "yes" or "no". Then, the answer we seek is the minimum value of T such that *enough*(T) = yes. Since *enough* is monotonic (that is, if *enough*(X) = yes, *enough*(Y) = yes when $X \leq Y$), we can perform binary search[*2] to find this minimum.

In order to determine *enough*(T) for a fixed value of T, it would be useful to understand causing an explosion in the following way: "decrease the health of all the monsters by B, then further decrease the health of one selected monster by $A - B$." Causing T explosions is now equivalent to decreasing the healths of all the monsters by $B \times T$, then do the following T times: further decrease the health of one selected monster by $A - B$ (we will call this an "extra attack"). In order to vanish all the monsters, we need to deliver $\lceil (h_i - B \times T)/(A - B) \rceil$ extra attacks to each monster i with a health greater than $B \times T$. If the total required number of extra attacks is at most T, *enough*(T) = yes, and otherwise no.

We can now determine *enough*(T) for a fixed value of T in $\mathrm{O}(N)$ time, which leads to the total time complexity of $\mathrm{O}(N \log(h_{\max}/B))$.

[*2] We will not explain the notion of binary search itself here. Resources can be found in, for example, Wikipedia.

E: Meaningful Mean

We will shift the range of the index of a and let $a = \{a_0, a_1, \ldots, a_{N-1}\}$ in this editorial.

We can transform the condition "the non-empty contiguous subsequence of a, $\{a_l, a_{l+1}, \ldots, a_{r-1}\}$ $(0 \leq l < r \leq N)$ (note that a_r is excluded), has a mean greater than or equal to K" into:

$$\sum_{i=l}^{r-1} a_i \geq (r-l)K \quad \Leftrightarrow \quad \sum_{i=0}^{r-1} a_i - \sum_{i=0}^{l-1} a_i \geq rK - lK \quad \Leftrightarrow \quad \sum_{i=0}^{r-1} a_i - rK \geq \sum_{i=0}^{l-1} a_i - lK$$

Let $b_j = \sum_{i=0}^{j-1} a_i - jK$ $(0 \leq j \leq N)$, and this condition can be simply written as $b_r \geq b_l$. We can easily find all the values of b_j $(0 \leq j \leq N)$ in $\mathrm{O}(N)$ time, and what remains is to count the number of the pairs (l, r) $(0 \leq l < r \leq N)$ such that $b_r \geq b_l$. It would be necessary to utilize some kind of data structure. We will explain a method that we think is easiest.

First, compress the $N+1$ values $b_0, b_1, \ldots, b_N$ into a range between 0 and N, keeping the magnitude relationship (by sorting and binary search), and let the compressed values be $c_0, c_1, \ldots, c_N$. Then, maintain the numbers of occurrences of the values between 0 and N using Binary Indexed Tree[*3]. In this manner, for each of $i = 0, 1, \ldots, N$, we can found the number of occurrences of values less than or equal to c_i among $c_0, c_1, \ldots, c_{i-1}$ in $\mathrm{O}(\log(N))$ time, which enables us to compute the desired number of pairs in a total of $\mathrm{O}(N \log(N))$ time.

[*3] We will not explain Binary Indexed Tree in detail here, but here is the summary: this data structure maintains a sequence of length n in a special format, and it can perform each of the following kinds of operations in $\mathrm{O}(\log(n))$ time: "incrementing the element at the specified position by a specified amount" and "finding the sum of the elements from the beginning of the sequence up to the specified position". More information can be found in, for example, Wikipedia (in the name of "Fenwick Tree" in English version).

F: Mirrored

We will see the given formula as $rev(N) - N = D$. Let the number of digits of N in decimal notation as L, and $N = \sum_{i=0}^{L-1} 10^i n_i$ $(0 \leq n_i \leq 9, n_{L-1} \neq 0)$. Then, $rev(N) - N$ can be transformed into:

$$rev(N) - N = \sum_{i=0}^{L-1} 10^{L-1-i} n_i - \sum_{i=0}^{L-1} 10^i n_i = \sum_{i=0}^{L-1} (10^{L-1-i} - 10^i) n_i = \sum_{i=0}^{\lfloor L/2 \rfloor - 1} (10^{L-1-i} - 10^i)(n_i - n_{L-1-i})$$

Here, let $d_i = n_i - n_{L-1-i}$ $(0 \leq i \leq \lfloor L/2 \rfloor - 1)$, which results in:

$$rev(N) - N = \sum_{i=0}^{\lfloor L/2 \rfloor - 1} (10^{L-1-i} - 10^i) d_i$$

Let the right side of this formula be $f(L, d)$. Finding the count of N that satisfy the given formula is almost equivalent to enumerating the pairs of L and a sequence of integers between -9 and 9, $d_0, d_1, \ldots, d_{\lfloor L/2 \rfloor - 1}$ (we also need to take into account that there is more than one N that corresponds to a sequence of d_i, though). Here, for any i such that $0 \leq i < \lfloor L/2 \rfloor - 1$, the following holds:

$$10^{L-1-i} - 10^i > \sum_{j=i+1}^{\lfloor L/2 \rfloor - 1} ((10^{L-1-j} - 10^j) \cdot 9) + 10^{L - \lfloor L/2 \rfloor}$$

(The "meaning" of this is as follows: "Suppose that $d_i = 1$ for some i $(\neq \lfloor L/2 \rfloor - 1)$. Then, even if we set $d_j = -9$ for all j such that $j > i$, the 'positive effect' on $f(L, d)$ by $d_i = 1$ is not completely negated, and more than $10^{L - \lfloor L/2 \rfloor}$ still remains.") Based on this, the following can be observed:

- Let L_D be the number of digits of D in decimal notation. Then, when $L > 2L_D$ and $f(L, d) > 0$, it can be seen that $f(L, d) > D$. Therefore, it is enough to consider the values between L_D and $2L_D$ as the value of L.
- For a fixed value of L, consider enumerating the sequences $d_0, d_1, \ldots, d_{\lfloor L/2 \rfloor - 1}$ such that $f(L, d) = D$ (and finding the count of N that satisfy the given formula), by performing Depth First Search starting from d_0. When the values up to d_{i-1} are already decided, it can be seen that there are at most two candidates of the value of d_i that have to be considered: the maximum value such that $(10^i - 10^{L-1-i}) d_i \leq dif$, and the minimum value such that $(10^i - 10^{L-1-i}) d_i > dif$. (Intuitively, if the "halfway" value of $f(L, d)$ during the search gets too far from D, it is not possible to "get back", and thus it should "stay close" to D.) Therefore, for a fixed value of L, we can find the count of N that satisfy the given formula in $O(2^{\lfloor L/2 \rfloor})$ time.

The above enables us to find the desired count of N in $O(\sum_{i=L_D}^{2L_D} 2^{\lfloor i/2 \rfloor}) = O(2^{L_D})$ time.