ABC 173 解説

evima, gazelle, kyopro_friends, satashun, sheyasutaka, ynymxiaolongbao

2020 年 7 月 5 日

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

A: Payment

(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。

「はじめての〜」に正解するソースコードがあれば、今回の問題でも入出力は行えるはずです。以下では、お釣りの金額を求めることに集中します。

支払う千円札の枚数を計算する方針も考えられますが、ここでは価格 N の下三桁に注目します。千円札のみで支払いを行う以上、N の下三桁のみがお釣りの金額に関係します。この下三桁に対応する数値は、多くの言語に存在する剰余演算子 `%` を用いて `N % 1000` と求められます。

この値が 0 であればお釣りは 0 円、そうでなければ 1000 からその値を引いたものがお釣りの金額です。ここでは if 文を使うこともできますが、`1000 - N % 1000` をさらに 1000 で割った余りを計算することでも対処できます。以下は後者を C++ で実装した例です。(説明は次頁へ続きます)

```cpp
#include <iostream>
using namespace std;
int main(){
    int N;
    cin >> N;
    cout << (1000 - N % 1000) % 1000 << endl;
}
```

言語によっては、「負の整数 % 正の整数」が負になりません。そのような言語では、以下の Ruby での実装例のように楽をすることもできます。

```
puts -gets.to_i % 1000
```

B: Judge Status

(解説: evima)

仮にテストケース数 N が 3 などの値に固定されていれば、if 文のみでも処理することができます。しかし、現実には N は 1 以上 10 万以下と一定でない上に大きく、for 文の出番です。

最も単純な方針は、以下の疑似コードの内容を実装することです。

```
N := input
C0, C1, C2, C3 := 0, 0, 0, 0
for i = 0, ..., N-1:
    S := input
    if S == "AC": C0 = C0 + 1
    else if S == "WA": C1 = C1 + 1
    else if S == "TLE": C2 = C2 + 1
    else: C3 = C3 + 1
print C0, C1, C2, C3
```

多くの言語で、この疑似コードを言語に合わせて修正することでこの問題を解くことができます。ただし、文字列を比較する際に言語によっては特別な注意を要します。言語ごとの詳細な情報は、検索エンジンで「[言語名] for 文」「[言語名] 文字列 比較」などと検索すれば手に入るはずです。以下に、疑似コードを Java のソースコードに変換したものを挙げます。

```java
import java.util.*;
public class Main {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int C0 = 0, C1 = 0, C2 = 0, C3 = 0;
        for(int i = 0; i < N; ++i){
            String S = sc.nextLine();
            if(S.equals("AC")) ++C0;
            else if(S.equals("WA")) ++C1;
            else if(S.equals("TLE")) ++C2;
            else ++C3;
```

```
        }
        System.out.println("AC x " + C0);
        System.out.println("WA x " + C1);
        System.out.println("TLE x " + C2);
        System.out.println("RE x " + C3);
    }
}
```

もちろん、上記の疑似コードと見た目が異なるコードを書くことも可能です。その一例として、以下に Python での実装例を挙げます。

```
N = int(input())
s = [input() for i in range(N)]
for v in ['AC', 'WA', 'TLE', 'RE']:
    print('{0} x {1}'.format(v, s.count(v)))
```

C: H and V

(解説: evima)

要するに、「1 行目を赤く塗るか？ 2 行目を塗るか？ $\cdots$ H 行目を塗るか？ 1 列目を塗るか？ 2 列目を塗るか？ $\cdots$ W 列目を塗るか？」という $H+W$ 個の二者択一の問いに「Yes, No, Yes, Yes, $\cdots$」などと答えることが求められています。このような Yes/No 列は 2^{H+W} 通り考えられ、今回の制約 $H, W \leq 6$ のもとでは最大でも $2^{12} = 4096$ 通りです。この程度であれば、全通りの Yes/No 列を試して実際に黒いマスが K 個残るようなものの数を数えても時間切れの心配はありません。

あとは Yes/No 列を列挙するというプログラミングの課題です。再帰関数などを用いる方針も考えられますが、ここでは Yes/No 列を二進数と対応づけて列挙することにします。以下、`0b` で始まる数は二進数です (例: $\texttt{0b0101} = 4 + 1 = 5$)。

しばらく行についてのみ考えます。例えば $H = 3$ のとき、3 個の列に対して $2^3 = 8$ 通りの Yes/No 列が考えられます。そして、「No, No, No」を $\texttt{0b000} = 0$、「Yes, No, No」を $\texttt{0b001} = 1$、「No, Yes, No」を $\texttt{0b010} = 2$、「Yes, Yes, No」を $\texttt{0b011} = 3$、$\cdots$、「Yes, Yes, Yes」を $\texttt{0b111} = 7$ に対応させれば、2^3 通りの Yes/No 列と 0 から $2^3 - 1 = 7$ までの整数が一対一に対応します (左右が反転していますが、整数の表記では大きい桁ほど左に置かれるためです)。

同様に、列の数がいくつであっても、Yes/No 列の前から i 番目の要素を二進数の 2^i の位の桁に紐付けることで、2^H 通りの Yes/No 列と 0 から $2^H - 1$ までの整数が一対一に対応します。列に関しても 2^W 通りの Yes/No 列を 0 から $2^W - 1$ までの整数と一対一に対応させれば、以下の疑似コードのように二重ループで 2^{H+W} 通りすべてのシナリオを列挙することができます。

```
H, W, K, c := input
ans := 0
for maskR = 0, ..., (1 << H) - 1:
    for maskC = 0, ..., (1 << W) - 1:
        black := 0
        for i = 0, ..., H-1:
            for j = 0, ..., W-1:
                if ((maskR >> i) & 1) == 0 and ((maskC >> j) & 1) == 0
                and c[i][j] == '#': black = black + 1
        if black == K: ans = ans + 1
print ans
```

D: Chat in a Circle

A_i をソートして昇順にします．このとき，$\sum_{k=1}^{N-1} A_{N-\lfloor k/2 \rfloor}$ が答えになることを以下に示します (ちなみに，$\sum$ は総和を表す記号です．たとえば上の式であれば，$k = 1, 2, \ldots, N-1$ のときの $A_{N-\lfloor k/2 \rfloor}$ を全て足し合わせた値を表します．便利な記法ですので，ぜひ知っておいてください)．

最大値を証明するには，「1. この値にできる」「2. これより良い値にはできない」をそれぞれ示すのが定石です．ここでもこの方針を採用します．

どうすれば「この値にできる」？

フレンドリーさが高い順に移動させます．$i+1$ 番目に輪に加わった人の両隣に $2i+1, 2i+2$ 番目の人を入れるようにすると，「まだ両隣挿入の対象になっていない人 A の両隣には，A より早く来た人がいる」状態を常に保てることから，上の値を達成できることが言えます．

どうして「これより良い値にはできない」？

まず，フレンドリーさが高い順に移動させる状況のみを考えればよいことを示します．i 番目に移動する人のフレンドリーさを a_i として $a_1 \geq a_2 \geq \cdots \geq a_k < a_{k+1}$ なる k があったとき，

- $k+1$ 人目が割り込む位置が k 人目の隣でないとき，入れ替えてもお互い知ったこっちゃありません．
- $k+1$ 人目が割り込む位置が k 人目の隣であるとき，その時点で x, a_k, a_{k+1}, y と並んでいたとすると，$\min(x, y) + \min(a_k, y) \leq \min(x, y) + \min(x, a_{k+1})$ なので，割り込む順番を入れ替えても損しません．

したがってバブルソートの要領で，移動の順番をフレンドリーさが高い順に並べ替えても損しません．ここからはそのような状況だけを考えることにします．

A_N が心地よさに寄与するのは高々 1 回です．それ以外の人はどうでしょう？ A_k の左隣に割り込みが起きたら，それ以降は A_k の左隣に A_k よりも後に来た人がいる状態が保たれます．右隣も同様のことが言えるので，結局 A_k の寄与は 2 回までです．

A_N (最大値) が 1 つ，$A_1 \sim A_{N-1}$ が 2 つずつあり，そのうち自由に $N-1$ 個を選んで足し合わせるとき，最大値は $\sum_{k=1}^{N-1} A_{N-\lfloor k/2 \rfloor}$ (冒頭に挙げた答え) です．したがって，本問の答えはこれより良くなりません．

以上より，この解法の正当性が示せました．ソートがボトルネックとなり，$\Theta(N \log N)$ 時間で解けます．Median of Medians というテクニックで $\Theta(N)$ 時間を達成できますが，発展的内容なのでここでは軽く触れるにとどめます．

E: Multiplication4

(解説：kyopro_friends)

まず $K = N$ の場合は明らかです。また、A_i の全ての数が負かつ K が奇数のとき、答えは負になります。このときは絶対値が小さい方から K 個の積を取れば良いです。それ以外の場合、積は必ず非負にすることができます。

(証明：値の大きい方から K 個の積を考える。これが非負なら OK。負なら、選んだ要素のうち最大の数 (仮定より正) を取り除き、かわりに選ばなかった要素のうち最小の数 (仮定より負) を加えることで正になる)

したがって、符号に気を付けながら絶対値の最大化をすればよいです。

解法 1: 最後に符号を合わせる

$\{A_i\}$ を絶対値の降順にソートし、大きい方から K 個選んだ積を P とします。P が非負なら答えは P です。P が負の場合、非負にするためには次の 2 種類の操作のどちらかをする必要があります。

(1) 負の数を 1 つ取り除き、非負の数を 1 つ加える
(2) 正の数を 1 つ取り除き、負の数を 1 つ加える

操作後の積の絶対値を最大化するには、取り除く要素の絶対値はなるべく小さく、加える要素の絶対値はなるべく大きくするのがよいので、それぞれの操作で選ぶべき数は一意に決まります。比較の方法に気を付けながら[*1]、より積が大きくなる方の操作を選べば良いです。そもそも一方の操作しか出来ない場合に注意してください。

解法 2: 非負の状態を保ちながら積を最大化する

S を空集合とします。「S の数の積が非負である」という条件を保ちながら、K 個の数を S に加えて積を最大化することを考えます。S の積が非負である状態を保ちながら要素を加える操作は全て、「非負の数を 1 つに加える」「負の数を 2 つ加える」という 2 種類の操作の繰り返しに分解できます。このことに注意すると、次のようなアルゴリズムで答えを得ることが出来ます。

次の操作を S の要素が K 個になるまで繰り返す。
操作：「まだ選んでいない非負の要素のうち絶対値が大きい順の 2 つ x_1, x_2 の積」と「まだ選んでいない負の要素のうち絶対値が大きい順の 2 つ y_1, y_2 の積」を比較する。前者の方が大きいか $|S| = K-1$ ならば x_1 を S に加える。そうでなければ y_1, y_2 を S に加える。

[*1] 「x_i を取り除いて y_i を加える」$(i = 1, 2)$ という 2 つの操作のどちらで積が大きくなるかは、$|P/x_1 * y_1| \leq |P/x_2 * y_2|$ かどうかがわかればよく、整理すると $|x_2 * y_1| \leq |x_1 * y_2|$ となり、今回の制約の下では 64bit 整数の範囲で計算できます。

これは、あらかじめ $\{A_i\}$ を非負の数と負の数に分け、絶対値の降順にソートしておくことで高速に行えます。「まだ選んでいない非負の要素・非負の要素」が 1 つ以下となる場合に注意してください。

F: Intervals on Tree

実際にシミュレーションをする方法としては、L を固定して R を順に $+1$ していきながら Union Find 木などで連結成分を管理することで、連結成分の個数を実際に全ての L, R について計算することができますが、この時間計算量は $O(N^2\alpha(N))$ となり遅いです。

元のグラフが木であることを利用するために、まず固定された L, R に対して $f(L, R)$ の値がどうなるか考えてみましょう。$S = \{x | L \leq x \leq R\}$ が誘導する部分グラフは、各連結成分が木であるようなグラフ (森) になります。木の場合、頂点数は辺の本数 $+1$ であるので、森の場合には 頂点数 $=$ 辺の本数 $+$ 連結成分 (木) の個数 となります。

よって、連結成分数 ($=$ 頂点数 $-$ 辺数) に各頂点と辺が寄与していると見なすことで、連結成分数の和を求める代わりに、各頂点・辺が何回足し引きされるかを考えればいいことがわかります。これは、頂点についてはその頂点を含む S の個数、辺についてはその両端点を含む S の個数をそれぞれ計算すれば良いので、実は答えを入力を読みながら直接計算することができ、時間計算量は $O(N)$ です。