ABC 104 Editorial

問題・解説: @evima0

2018 年 8 月 5 日

A: Rated for Me

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）整数変数 R を宣言する。
1. 標準入力から R を受け取る。
2. R から何らかの方法で答えとなる文字列を得る。
3. 求めた文字列を標準出力に出力する。

手順 0, 1, 3 については、practice contest の問題 A の言語別サンプルコードが参考になる（ほぼそのまま使える）でしょう。手順 2 については、if, else などの構文を用いることになります。C++ による実装例を示します。

```cpp
#include <iostream>
using namespace std;
int main(){
    int R;
    cin >> R;
    string ans;
    if(R < 1200){
        ans = "ABC";
    }else if(R < 2800){
        ans = "ARC";
    }else{
        ans = "AGC";
    }
    cout << ans << endl;
}
```

B: AcCepted

「先頭から 3 文字目から最後から 2 文字目までに C がちょうど 1 個現れ、それ以外の文字はすべて小文字である」という内容を正確に実装することにほぼ終始します（「それ以外の文字」に先頭から 2 文字目と最後の文字も含まれることに注意）。
素直に実装するには、for 文を用いてすべての文字の case（大文字か小文字か）を確かめます。一つ一つの文字が大文字か小文字かを判定するには、言語ごとに用意されたメソッドを使うか、ASCII コードでの文字の並び順を利用します。C++ による実装例を示します。

```cpp
#include <iostream>
using namespace std;
int main(){
    string S;
    cin >> S;
    int L = S.size();
    string ans = "AC";
    if(S[0] != 'A'){
        ans = "WA";
    }
    int cnt = 0;
    for(int i = 1; i < L; ++i){
        if(isupper(S[i])){
            if(i == 1 || i == L - 1 || S[i] != 'C'){
                ans = "WA";
            }
            ++cnt;
        }
    }
    if(cnt != 1){
        ans = "WA";
    }
    cout << ans << endl;
}
```

C: All Green

実行時間制限が許すならば、100 点の問題を何問解き、200 点の問題を何問解き、…という配点ごとの解く問題の数の組み合わせをすべて試して最良のものを採用すればよいです。しかし、最大で 10 種類の配点それぞれに対して問題が最大で 100 問存在するため、このような組み合わせは最大で 101^{10} 通り存在します。これらをすべて試すには 1 万年以上の時間が必要となるでしょう。

しかし、最適解を求める上で 101^{10} 通りすべての組み合わせを試す必要はありません。例えば、100 点の問題と 200 点の問題が 10 問ずつあったとして、最終的にこれらを 3 問ずつ解く、という選択を考慮する必要はありません。なぜなら、その代わりに 200 点問題を 6 問解くことで、解く問題数を増やさずに総合スコアを高くすることができるからです。

以下、「$100i$ 点の問題を p_i 問解く」ことを「$100i$ 点を完全に解く」、「$100i$ 点の問題を 1 問以上 $p_i - 1$ 問以下解く」ことを「$100i$ 点を中途半端に解く」、「$100i$ 点の問題を 0 問解く」ことを「$100i$ 点をまったく解かない」と表現します。上の議論を一般化すると、以下のような選択のみを考慮しても最適解はそれらの選択に含まれます。

- 中途半端に解く配点は 1 種類以下であり、それ以外の配点は完全に解くかまったく解かない。
- 中途半端に解く配点があるなら、それは完全に解く配点以外の配点の中で最も高い配点である。

このような選択をすべて列挙するには、D 種類の配点それぞれに対してその配点を完全に解くか否かを決定し、必要であれば残りの配点のうち最も高いものを中途半端に解く配点とすればよいです（中途半端に解く配点の問題を 1 問除きすべて解いても目標点に達しない場合は、完全に解く配点の選択が不適切であったとして無視します）。

この方針を実装するには、再帰関数を定義するか、0 以上 $2^D - 1$ 以下の整数それぞれの 2 進数表記を D 回の選択を表すリストのようなものとして用いる（下から i 桁目 $(0 \leq i < D)$ が 1 なら $100(i+1)$ 点を完全に解き、0 なら完全には解かない）のが素直でしょう。素直な実装で $O(2^D D)$ 時間の解法が得られます。

参考実装 (提出 #2954675, C++)

D: We Love ABC

表面的なことですが、文字列 S の ABC 数を「S の文字のうち 3 文字に丸をつけ、丸をつけた文字を左から読むと `ABC` となるようにする方法」と捉えると考えやすいかもしれません。求めたいのは、「S に含まれる `?` をそれぞれ `A`, `B`, `C` のいずれかに書き換えつつ、S の文字のうち 3 文字に丸をつけて、丸付きの文字が左から `ABC` となるようにする方法」です。

このように考えると、動的計画法（リンク先は Wikipedia の同名の記事）の適用が見えてきます。値 $dp_{i,j} (0 \leq i \leq |S|, 0 \leq j \leq 3)$ を「S の先頭から $i-1$ 文字目までに対する処理（`?` の置換と丸つけ）をすでに行っていて、これまでに丸を j 個つけている場合の、S の残りの部分に対する処理を行う方法の個数」とすると、以下が成り立ちます（文字列の先頭の文字を 0 文字目とします）。

- $dp_{|S|,3} = 1$
  （処理が正常に終了。これから行うべきことは「何もしない」の 1 通り）
- $j < 3$ のとき、$dp_{|S|,j} = 0$
  （丸を 3 個つける前に最後の文字まで見終わってしまい、手遅れ。これからできることは 0 通り）
- $i < |S|$ のとき、$dp_{i,3} = m \times dp_{i+1,3}$, ここで $S_i =$ `?` なら $m = 3$, そうでなければ $m = 1$
  （丸は 3 個つけ終わったので、あとは `?` を置換するのみ）
- $i < |S|, j < 3$ のとき、$dp_{i,j} = m_1 \times dp_{i+1,j} + m_2 \times dp_{i+1,j+1}$,
  ここで $S_i =$ `?` なら $m_1 = 3$, そうでなければ $m_1 = 1$ であり、
  $S_i =$ `?` または S_i が `ABC` のうち j 番目（0 から数えて）の文字なら $m_2 = 1$, そうでなければ $m_2 = 0$
  （式の前半部分は i 文字目に丸をつけない場合、後半部分は丸をつける場合に対応）

これを元に $dp_{i,j} (0 \leq i \leq |S|, 0 \leq j \leq 3)$ の値を i, j の降順にすべて求めることができ、$O(N)$ 時間の解法が得られます。

参考実装 (提出 #2955456, C++)