ABC 152 解説

writers: latte0119, beet, DEGwer, drafear, IH19980412, kyopro_friends, tozangezan, yokozuna57

2020 年 1 月 19 日

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

A: AC or WA

$N = M$ であるかどうか判定すればよいです。以下は C++ における実装例です。

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

int main(){
        int n,m;
        cin>>n>>m;
        if(n==m)cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
        return 0;
}
```

B: Comparing Strings

考えられる方法の一つは、2 つの文字列を実際に作り、辞書順比較することです。辞書順比較は、例えば C++ ならば string 型同士で比較をすることで可能です。

しかし、a を b 回繰り返した文字列は a で始まり、b を a 回繰り返した文字列は b で始まるため、$a < b$ ならば常に a を b 回繰り返した文字列の方が辞書順で小さいですし、$a > b$ ならば常に b を a 回繰り返した文字列の方が辞書順で小さいです。また、$a = b$ なら二つの文字列が等しくなるのでどちらを出力しても同じになります。

Listing 1　C++ での実装例 1

```
#include <stdio.h>
#include <string>
#include <algorithm>
using namespace std;
char in[110];
int main(){
        int a,b;scanf("%d%d",&a,&b);
        for(int i=0;i<b;i++)in[i]='0'+a;
        string L=in;
        for(int i=0;i<a;i++)in[i]='0'+b;
        in[a]=0;
        string R=in;
        printf("%s\n",min(L,R).c_str());
}
```

Listing 2　C++ での実装例 2

```
#include <stdio.h>

int main(){
        int a,b;scanf("%d%d",&a,&b);
        if(a<b){
                for(int i=0;i<b;i++)printf("%d",a);
        }else{
                for(int i=0;i<a;i++)printf("%d",b);
        }
        printf("\n");
}
```

C: Low Elements

$i = 1, ..., N$ に対して、$M_i := min\{P_j | 1 \leq j \leq i\}$ と定義します。すると、$i(1 \leq i \leq N)$ を固定して考えたときに、 $M_i = P_i$ であることと、 任意の $j(1 \leq j \leq i)$ に対して $P_i \leq P_j$ であることが同値となります。したがって、 M_i をすべて計算することができれば十分です。これは、 i が小さいほうから順に計算していけばよく、 $O(N)$ でこの問題を解くことができました。

D: Handstand 2

整数 $i, j (0 \leq i, j \leq 9)$ に対して、
$c_{i,j} := \#\{1 \leq k \leq N | k$ の先頭の桁の数は i に等しく、末尾の桁の数は j に等しい $\}$
と定義します。これは、$O(N)$ で計算することができます。$c_{i,j}$ を用いると、この問題の答えは
$\sum_{i=0}^{9} \sum_{j=0}^{9} c_{i,j} \times c_{j,i}$
と計算することができます。

E: Flatten

条件を満たすような B_i を決めたとします。このとき $K = A_1B_1$ とおくと、条件より全ての $j > 1$ について $K = A_1B_1 = A_jB_j$ となります。よって K は $A_1, ..., A_N$ の公倍数です。逆に L を $A_1, ..., A_N$ の最小公倍数とおいたとき、$B_i = L/A_i$ と定めると、このような B_i たちは条件を満たします。よって求める答えは、$\sum L/A_i$ です。しかし、L は非常に大きな数となることがあるため、そのまま計算することはできません。そこで、L を素因数分解した形で保持し、計算することを考えます。

p_i を素数とし、$X = \prod p_i^{e_i}$, $Y = \prod p_i^{e'_i}$ と[*1]素因数分解されているとします (e_i, e'_i の一方は 0 でもよい)。このとき、X と Y の最小公倍数は $\prod p_i^{\max(e_i, e'_i)}$ になります。このことを用いると、$A_1, ..., A_N$ の最小公倍数を素因数分解された形で求めることができます。

以上より、この問題は $A = \max A_i$ として、$O(N\sqrt{A})$ で解くことができました。また、適当な前計算により素因数分解を高速化するなどの工夫により $O(A + N \log A)$ で求めることもできます。

[*1] 記号 $\prod$ は、$\sum$ の"掛け算バージョン"です

F: Tree and Constraints

”すべての条件を満たす塗り方”を数える代わりに、” 1 つ以上の条件を満たさない塗り方”を数えることにします。

包除原理を使うと、 M 個の制約から 1 つ以上の制約を選んで、選んだ制約を 1 つも満たさないような塗り方を数える問題になります。

制約の選び方を 1 つ固定します。選ばれた制約を 1 つも満たさないような塗り方の数を効率的に計算することを考えます。選ばれた制約に対応するパス上に存在する辺は、必ず白く塗らなければなりません。そうでない辺は、どちらの色で塗ってもよいです。したがって、選ばれた制約に対応するパスたちに覆われている辺の数を C とおくと、選ばれた制約を 1 つも満たさないような塗り方の数は、 2^{N-1-C} 通りです。

適当に根を選んで根付き木として考え、各 u_i, v_i に対して最近共通祖先 (LCA) を事前に求めておきます。すると、 C の計算の際には、木上で累積和を用いればよく、 $O(N+M)$ で実行できます。

以上より、 $O((N+M)2^M)$ でこの問題を解くことができました。