

# 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 starts on page 7.*

## 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。

---

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     int n,m;
6     cin>>n>>m;
7     if(n==m)cout<<"Yes"<<endl;
8     else cout<<"No"<<endl;
9     return 0;
10 }
```

---

## 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

---

```
1 #include <stdio.h>
2 #include <string>
3 #include <algorithm>
4 using namespace std;
5 char in[110];
6 int main(){
7     int a,b;scanf("%d%d",&a,&b);
8     for(int i=0;i<b;i++)in[i]='0'+a;
9     string L=in;
10    for(int i=0;i<a;i++)in[i]='0'+b;
11    in[a]=0;
12    string R=in;
13    printf("%s\n",min(L,R).c_str());
14 }
```

---

Listing 2 C++ での実装例 2

---

```
1 #include <stdio.h>
2
3 int main(){
4     int a,b;scanf("%d%d",&a,&b);
5     if(a<b){
6         for(int i=0;i<b;i++)printf("%d",a);
7     }else{
8         for(int i=0;i<a;i++)printf("%d",b);
9     }
10    printf("\n");
11 }
```

---

## C: Low Elements

$i = 1, \dots, N$  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、

$c_{i,j} := \#\{1 \leq k \leq N \mid k \text{ の先頭の桁の数は } i \text{ に等しく、末尾の桁の数は } j \text{ に等しい}\}$

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  $K = A_1 B_1$  とおくと、条件より全ての  $j > 1$  について  $K = A_1 B_1 = A_j B_j$  となります。よって  $K$  は  $A_1, \dots, A_N$  の公倍数です。逆に  $L$  を  $A_1, \dots, A_N$  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、 $A_1, \dots, A_N$  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。

## A: AC or WA

It is enough to check if  $N = M$ . The following is a sample code in C++.

---

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     int n,m;
6     cin>>n>>m;
7     if(n==m)cout<<"Yes"<<endl;
8     else cout<<"No"<<endl;
9     return 0;
10 }
```

---

## B: Comparing Strings

One possible way is constructing to strings actually and compare lexicographically. This can be achieved by, for example in C++, comparing between string type.

However, the concatenation of  $b$  copies of  $a$  starts with  $a$ , and the concatenation of  $a$  copies of  $b$  starts with  $b$ , so if  $a < b$  then the concatenation of  $b$  copies of  $a$  is the smaller lexicographically, and if  $a > b$  then the concatenation of  $a$  copies of  $b$  is the smaller lexicographically. Also, if  $a = b$ , the two strings are the same, so you will obtain the same result by printing either of them.

Listing 3 Sample Code in C++ 1

---

```
1 #include <stdio.h>
2 #include <string>
3 #include <algorithm>
4 using namespace std;
5 char in[110];
6 int main(){
7     int a,b;scanf("%d%d",&a,&b);
8     for(int i=0;i<b;i++)in[i]='0'+a;
9     string L=in;
10    for(int i=0;i<a;i++)in[i]='0'+b;
11    in[a]=0;
12    string R=in;
13    printf("%s\n",min(L,R).c_str());
14 }
```

---

Listing 4 Sample Code in C++ 2

---

```
1 #include <stdio.h>
2
3 int main(){
4     int a,b;scanf("%d%d",&a,&b);
5     if(a<b){
6         for(int i=0;i<b;i++)printf("%d",a);
7     }else{
8         for(int i=0;i<a;i++)printf("%d",b);
9     }
10    printf("\n");
11 }
```

---



## C: Low Elements

For  $i = 1, \dots, N$  we define  $M_i := \min\{P_j | 1 \leq j \leq i\}$ . Then, when  $i(1 \leq i \leq N)$  is fixed,  $M_i = P_i$  holds if and only if for all  $j(1 \leq j \leq i)$ ,  $P_i \leq P_j$  holds. Therefore, it is enough to calculate all the  $M_i$ . These can be calculated in the increasing order of  $i$ , so the problem could be solved in a total of  $O(N)$  time.

## D: Handstand 2

For each pair of integers  $i, j (0 \leq i, j \leq 9)$ , let us define

$c_{i,j} := \#\{1 \leq k \leq N \mid \text{The first digit of } k \text{ is equal to } i, \text{ and the last digit is equal to } j \}$ .

Those can be calculated in a total of  $O(N)$  time. by using  $c_{i,j}$ , the answer for the problem can be calculated by

$$\sum_{i=0}^9 \sum_{j=0}^9 c_{i,j} \times c_{j,i}.$$

## E: Flatten

Assume that  $B_i$  satisfying the given conditions are determined. Then let  $K = A_1 B_1$ , then by the constraints  $K = A_1 B_1 = A_j B_j$  holds for all  $j > 1$ . Therefore  $K$  is common multiple of  $A_1, \dots, A_N$ . Conversely, let  $L$  be the least common multiple of  $A_1, \dots, A_N$ , and let  $B_i = L/A_i$ , then such  $B_i$ s satisfies the conditions. Therefore, the desired answer is  $\sum L/A_i$ . However,  $L$  can be a very big number, so it can't be calculated directly. Then let's consider calculating, holding  $L$  in a factorized form.

Let  $p_i$  be primes, and assume that factorizations are given by  $X = \prod p_i^{e_i}$ ,  $Y = \prod p_i^{e'_i}$ <sup>\*1</sup> (either of  $e_i, e'_i$  may be 0). Then the least common multiple of  $X$  and  $Y$  is given by  $\prod p_i^{\max(e_i, e'_i)}$ . By using this, the least common multiple of  $A_1, \dots, A_N$  can be obtained in a factorized form.

Therefore, this problem can be solved in a total of  $O(N\sqrt{A})$  time, where  $A = \max A_i$ . Also, by speeding up the prime factorization with proper precalculations, the answer can also be obtained in a total of  $O(A + N \log A)$  time.

---

<sup>\*1</sup> The symbol  $\prod$  is a "product version" of  $\sum$

## F: Tree and Constraints

Instead of counting the number of "ways of painting that satisfy all the conditions," let us count the number of "ways of painting that do not satisfy one or more conditions."

By the inclusion-exclusion principle, it is rephrased to the problem of choosing one or more constraints out of  $M$  constraints and count the number of such ways of painting that do not satisfy any of the chosen constraints.

Fix a combination of constraints. Consider how we can efficiently count the such ways of painting that do not satisfy the chosen constraints. The edges on the paths corresponding to the chosen constraints must be painted white. The other edges can be painted with either color. Therefore, let  $C$  be the number of edges covered by the paths corresponding to the chosen constraints, then there are  $2^{N-1-C}$  ways of painting such that do not satisfy any of the chosen constraints.

Regard as a rooted tree by choosing an arbitral root, and for each  $u_i, v_i$  find their Lowest Common Ancestor (LCA) beforehand. Then, when calculating  $C$  you can use cumulative sums, so it can be executed in a total of  $O(N + M)$  time.

Therefore, this problem could be solved in a total of  $O((N + M)2^M)$  time.