

ABC 148 解説

beet, DEGwer, gazelle, kyopro_friends, sheyasutaka, tozangezan

2019 年 12 月 22 日

For International Readers: English editorial starts on page 8.

A: Round One

$6 - A - B$ を計算し、その値を出力すればよいです。

入力および出力の方法は言語によって異なりますが、たとえば C++ であれば cin, cout を使うことで実現できます。

Listing 1 C での実装例

```
1 #include <stdio.h>
2
3 int main(void){
4     int A, B;
5     scanf("%d%d", &A, &B); // 入力からA, B を受け取る
6
7     int result = 6 - A - B; // 6-A-B を計算し、変数result に格納する
8     printf("%d\n", result); // 出力
9
10    return 0;
11 }
```

Listing 2 ferNANDo での実装例

```
1 r a7 a6 a5 a4 a3 a2 a1 a0
2 r f e r n a n n d o
3 r b7 b6 b5 b4 b3 b2 b1 b0
4 d a7 b7
5 e a7 d
6 f b7 d
7 x7 e f
8 d a6 b6
9 e a6 d
```

10 f b6 d
11 x6 e f
12 d a5 b5
13 e a5 d
14 f b5 d
15 x5 e f
16 d a4 b4
17 e a4 d
18 f b4 d
19 x4 e f
20 d a3 b3
21 e a3 d
22 f b3 d
23 x3 e f
24 d a2 b2
25 e a2 d
26 f b2 d
27 x2 e f
28 d a1 b1
29 e a1 d
30 f b1 d
31 x1 e f
32 d a0 b0
33 e a0 d
34 f b0 d
35 x0 e f
36 x4 x4 x4
37 x5 x5 x5
38 x7 x6 x5 x4 x3 x2 x1 x0

B: Strings with the Same Length

1 から N までループを回し、 i 回目のループでは S の i 文字目を出力する、その後 T の i 文字目を空白を開けずに出力する、ということを繰り返すと、最終的に目的の新しい文字列が出力されます。

以下は C++ による実装例です (include 等は省略しています)。

```
1 char S[110];
2 char T[110];
3 int main(){
4     int a;scanf("%d",&a);
5     scanf("%s%s",S,T);
6     for(int i=0;i<a;i++)printf("%c%c",S[i],T[i]);printf("\n");
7 }
```

C: Snack

A の正の倍数を小さい順に見ていき、それが B の倍数であるかを確認すればよいです。このとき、 $A \times B$ は明らかに両方の倍数となるので、高々 B 回確認すれば十分なことがわかります。

以下は C++ による解答例です。オーバーフローに注意してください。

<https://atcoder.jp/contests/abc148/submissions/9083476>

余談ですが、このような数は A と B の最小公倍数と呼ばれており、 $O(\log(\min(A, B)))$ の計算量で求めることも可能です。

D: Brick Break

以下、「すぬけさんが満足するレンガの砕き方」を「いい砕き方」と言います。

まず 1 が書かれたレンガが存在しない場合、答えは明らかに -1 です。

そうでない場合、1 が書かれたレンガの中で最も左にあるものの番号を i とします。このとき、 i 番目のレンガを砕く「いい砕き方」に対して、それと同じ個数のレンガが残るような、 i 番目のレンガを砕かない「いい砕き方」が存在します。というのも、元の砕き方で残る 1 が書かれたレンガを砕き、代わりに i 番目のレンガを砕かないとしても変わらず「いい砕き方」になるからです。よって、残すレンガの個数が最大の「いい砕き方」を探す上で、 i 番目のレンガは砕かないと仮定しても問題ありません。

同様の考え方をすると、 i 番目のレンガより右にあるレンガの中に 2 が書かれたものが存在していれば、そのうち最も左にあるものは砕かずに残すと仮定することが可能です。

この貪欲的なアプローチを繰り返していくことで、残すレンガの個数が最大の「いい砕き方」を見つけることができます。

組合せ最適化問題を解く際に、ある要素を選んで損をすることがないので、その要素を選ぶと仮定して探索空間を小さくする、というアイデアは、競技プログラミングにおける貪欲法で典型的です。

E: Double Factorial

$f(N)$ を 10 進法で表記したときに末尾に続く 0 の個数は、 $f(N)$ が 2 で割り切れる回数と $f(N)$ が 5 で割り切れる回数の \min です。

以下、 N の偶奇に分けて考えます

- N が奇数の場合 … $f(N)$ はいくつかの奇数の積なので、2 で割れることはありません。よって答えは常に 0 です。
- N が偶数の場合 … 明らかに、 $f(N)$ が 2 で割り切れる回数は 5 で割り切れる回数よりも多いので、5 で割り切れる回数のみを考えます。

$f(N) = N(N-2)(N-4)\cdots 2$ が 5 で割り切れる回数は、 $(2, 4, \dots, N$ のうち 5 で割り切れるものの個数) + $(2, 4, \dots, N$ のうち 5^2 で割り切れるものの個数) + $(2, 4, \dots, N$ のうち 5^3 で割り切れるものの個数) … となります。それぞれ、 $2, 4, \dots, N$ のうち 5 で割り切れるものの個数は $\text{floor}(N/10)$ 、 $2, 4, \dots, N$ のうち 5^2 で割り切れるものの個数は $\text{floor}(N/50)$ 、 $2, 4, \dots, N$ のうち 5^3 で割り切れるものの個数は $\text{floor}(N/250)$ 、… なので、 $N/10$ から順に、分母が N 以下の間、分母を 5 倍して商を求めるというのを繰り返し、総和を取ればよいです。

F: Playing tag on tree

高橋君が最初の行動で青木君のいる頂点にしか移動できないとき答えは0です。そうでないときを考えます。

青木君が最初にいる頂点を根とします。高橋君と青木君はすれ違うことができないので、ゲームが終了するのは高橋君が”行き止まり”に追い詰められたときです(そうでない場合、高橋君は葉の方へ逃げることでゲームを引き延ばせます)。よって、ゲームが終了する直前は「高橋君が葉 x 、青木君が x の親の親にいる」という状況になり、その後2人が1度ずつ行動し、ともに x の親にいる状況でゲームが終了します。

青木君は常に最短距離で高橋君を追い詰めることができるので、青木君の移動回数は高橋君がどの葉で追い詰められたかのみ依存します。したがって高橋君は追い詰められる場所として、「高橋君が青木君より先に到達することができるような葉のうち、青木君から最も遠いもの」を選択するのが最善となります。これは頂点 u, v からの距離をそれぞれ DFS や BFS などによって計算することで求めることができ、計算量は $O(N)$ です。

A: Round One

It would be enough to calculate $6 - A - B$ and output the value.

The way of input and output depends on languages. For example, in C++, you can implement by using cin and cout.

Listing 3 Implementation example in C

```
1 #include <stdio.h>
2
3 int main(void){
4     int A, B;
5     scanf("%d%d", &A, &B); // receive A and B from input
6
7     int result = 6 - A - B; // calculate 6-A-B and store into the variable
      result
8     printf("%d\n", result); // output
9
10    return 0;
11 }
```

Listing 4 Implementation example in ferNANDo

```
1 r a7 a6 a5 a4 a3 a2 a1 a0
2 r f e r n a n d o
3 r b7 b6 b5 b4 b3 b2 b1 b0
4 d a7 b7
5 e a7 d
6 f b7 d
7 x7 e f
8 d a6 b6
9 e a6 d
10 f b6 d
11 x6 e f
12 d a5 b5
13 e a5 d
14 f b5 d
15 x5 e f
16 d a4 b4
17 e a4 d
18 f b4 d
19 x4 e f
20 d a3 b3
```


21 e a3 d
22 f b3 d
23 x3 e f
24 d a2 b2
25 e a2 d
26 f b2 d
27 x2 e f
28 d a1 b1
29 e a1 d
30 f b1 d
31 x1 e f
32 d a0 b0
33 e a0 d
34 f b0 d
35 x0 e f
36 x4 x4 x4
37 x5 x5 x5
38 x7 x6 x5 x4 x3 x2 x1 x0

B: Strings with the Same Length

Perform loop from 1 to N : in the i -th loop, output i -th letter of S , and then output i -th letter of T without printing whitespace. Then the desired new string is printed in the end.

The following is an implementation example in C++ (include etc. are abbreviated).

```
1 char S[110];
2 char T[110];
3 int main(){
4     int a;scanf("%d",&a);
5     scanf("%s%s",S,T);
6     for(int i=0;i<a;i++)printf("%c%c",S[i],T[i]);printf("\n");
7 }
```

C: Snack

It will be enough to look up each of the multiples of A in the increasing order, and check if it is multiple of B . Since $A \times B$ is obviously multiple of both A and B , it turn out that it is enough to check at most B times.

The following is a sample code by C++. Be careful of overflows.

<https://atcoder.jp/contests/abc148/submissions/9083476>

Just an aside, such integer is called the least common multiple of A and B , and it can be found in a $O(\log(\min(A, B)))$ time.

D: Brick Break

Hereinafter we will call a combination of bricks you will break "good" if Snuke will be satisfied after they are broken.

First, if there are no bricks that 1 is written on it, then the answer is obviously -1 .

Otherwise, let i be the index of the leftmost brick with 1 written on it. Then, for any "good combination" including i -th brick, there exists a "good combination" of same size such that i is not included. This is because if you break the brick with 1 written on it which originally remains, and instead leaving i -th brick unbroken, then the resulting combination of bricks is still good. Therefore, when looking for the "good combination" of breaking such that the number of remaining blocks is maximum, you can always assume that you will not break i -th block.

Similarly, if there exists a brick with 2 written on it on the right of i -th brick, then you can assume that you will leave the leftmost brick among them without breaking.

By repeating this greedy approach, you can find a "good combination" of breaking such that the number of remaining blocks is maximum.

When solving combinational optimization problem, the idea of "choosing this element does not lead to disadvantage, so we can narrow down the search space by assuming that we choose it" is very typical in greedy algorithms in competitive programming.

E: Double Factorial

The number of trailing zeros in the decimal notation of $f(N)$ is min of how many times $f(N)$ can be divided by 2 and how many $f(N)$ can be divided by 5.

We will divide into cases depending on parity of N .

- If N is odd \dots $f(N)$ is product of some odd numbers, so it can never be divided by 2. Therefore, the answer is always 0.
- If N is even \dots Obviously, the number of times $f(N)$ can be divided by 2 is more than the number of times it can be divided by 5, so we only have to consider how many times it can be divided by 5.

The number of times $f(N) = N(N-2)(N-4)\dots 2$ can be divided by 5 is equal to (how many integers out of $2, 4, \dots, N$ can be divided by 5?) + (how many integers out of $2, 4, \dots, N$ can be divided by 5^2 ?) + (how many integers out of $2, 4, \dots, N$ can be divided by 5^3 ?) \dots . Since $\text{floor}(N/10)$ integers out of $2, 4, \dots, N$ can be divided by 5, $\text{floor}(N/50)$ integers out of $2, 4, \dots, N$ can be divided by 5^2 , $\text{floor}(N/250)$ integers out of $2, 4, \dots, N$ can be divided by 5^3 , \dots , so it would be enough to, starting from $N/10$, repeat multiplying the denominator by 5 and calculating the quotient while the denominator is no more than N , and find the sum of the quotients.

F: Playing tag on tree

If Takahashi can only move to the vertex where Aoki is standing for the first turn, then the answer is 0. Let's consider the other cases.

Let the vertex where Aoki is initially standing be the root. Since Takahashi and Aoki cannot pass each other, the game ends when Takahashi is cornered into a "dead end" (or otherwise, Takahashi can extend the game by escaping in the direction of leaves). Therefore, right before the game ends, "Takahashi is on a leaf x and Aoki is on the parent of the parent of x ," and soon after that they move once each to the parent of x , when the game ends.

Since Aoki can always take the shortest path to corner Takahashi, so the number of moves Aoki performs always depend on which leaf Takahashi will be cornered into. Therefore, the optimal choice of vertex that Takahashi will be cornered into is "the furthest vertex from Aoki which Takahashi can reach before Aoki can." This can be obtained by calculating the distance from u and v by DFS or BFS, and the time complexity is $O(N)$.