

# ABC 136 解説

DEGwer, drafeear, evima, satashun

2019 年 8 月 4 日

*For International Readers: English editorial starts on page 7.*

## A: Transfer

容器 1 にはあと  $A - B$  ミリリットルまで入れることができます。よって、容器 2 から移す水の量 (ミリリットル) は  $A - B$  と  $C$  の小さい方になります。したがって、容器 2 に残る水の量は  $C - \min(A - B, C)$  ミリリットルになります。C++ 言語での実装例を以下に示します。

---

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main() {
6     int A, B, C; cin >> A >> B >> C;
7     int cap = A - B;
8     int use = min(C, cap);
9     int ans = C - use;
10    cout << ans;
11 }
```

---

## B: Uneven Numbers

(原案: DEGwer, 準備・解説: evima)

$N$  は 10 万以下とあまり大きくなく、 $1, 2, \dots, N$  という  $N$  個の数の桁数を直接数えるのが楽です。

そして、それぞれの数の桁数を数える最も素直な方法は、数を文字列に変換してその長さを数えるというものでしょう。最も効率的な方法ではありませんが、今回はこれで十分です。

以下に Python による実装例を二つ掲載します。他の言語でもこれらと似たような実装が可能でしょう。

---

```
1 N = int(input())
2 ans = 0
3 for i in range(1, N + 1):
4     if len(str(i)) % 2 == 1:
5         ans += 1
6 print(ans)
```

---

---

```
1 N = int(input())
2 print(len(list(filter(lambda x: len(str(x)) % 2 == 1, range(1, N + 1)))))
```

---

## C: Build Stairs

右のマスから順に操作を選ぶことにすると、できるだけ高くしておいた方が後で選択肢が増えるので、

- 何もしなくて良いなら何もしない
- そうでなくて、右のマスよりもちょうど1高いなら1低くする
- そうでないならどう頑張ってもダメなので答えは No

を右から順に行き、うまく構成できれば Yes を出力することで  $O(N)$  の計算時間で答えを求めることができます。また、同様の考え方で、左から順に行うこともできます。

他の方法としては、 $i$  番目までの高さの最大値を  $M_i$  としたときに、全ての  $i = 1, 2, \dots, N$  について、 $H_i \geq M_i - 1$  が成り立てば Yes、そうでなければ No と判定することもできます。これは帰納法によって示すことができます。

実際に「 $\forall i. H_i \geq M_i - 1 \Leftrightarrow$  答えは Yes」を示してみます。

$\Leftarrow$  は簡単です。対偶を取ると、示すべきは「 $\exists i. H_i < M_i - 1 \Rightarrow$  答えは No」なので、これは自明です\*1。

$\Rightarrow$  は帰納法で示します。

- $N = 1$  のときは必ず答えは Yes です。
- $N = N' + 1$  のとき、仮定より  $i = 1, 2, \dots, N'$  について条件を満たすので、帰納法の仮定より  $H_1, H_2, \dots, H_{N'}$  に対する解 (単調非減少にする操作の仕方) が存在します。この解への操作に上書きして、 $H_i = M_N$  となる  $i \in \{1, 2, \dots, N'\}$  について1低くする選択肢を選ぶとしても  $H_1, H_2, \dots, H_{N'}$  に対する解となっていて、これに加えて  $H_N$  については何もしない選択をすると  $H_1, H_2, \dots, H_N$  に対する解が構成でき、Yes です。

---

\*1  $H_i < M_i - 1$  となる  $i$  を  $x$  とし、 $H_j = M_x$  となる  $j$  を  $y$  とします。すると、 $y < x$  かつ  $H_x \leq H_y - 2$  が成り立ちます。つまりあるマスより右側のマスで、高さが2以上低いマスが存在するのでどう頑張ってもこの2マスに対する条件を満たすことができません。

## D: Gathering Childrens

“RR...RLL...L”の部分に分解すると、他の部分と行き来することはないので独立に考えることができます。例えば、“RRLRL”は“RRL”と“RL”について独立に考えることができます。以降、“RR...RLL...L”の形についてのみ考えます。

十分な回数移動を行うので、境界部分、すなわち“RL”の部分に子供が集まり、他の部分には誰も居ない状態になります。さらに、最初左から偶数番目のマスにいた子供と奇数番目のマスにいた子供が同じマスに集まることはなく、逆に左からの位置の偶奇が等しいマスにいた子供たちは最終的には全員同じマスに集まります。移動回数は  $10^{100}$  回と偶数回なので、“RL”の“R”には“R”から偶数マス離れた子どもたちが集まり、“RL”の“L”には“L”から偶数マス離れた子どもたちが集まります。

このように計算することで時間計算量  $O(N)$  で答えを求めることができます。

実装では、文字列を連続した文字ごとに分解する (例えば “AABCCC” を (‘A’, 2), (‘B’, 1), (‘C’, 3) のように分解する) ランレングス圧縮のライブラリがあると便利です。

別解として、次のような解法もあります。マス数は高々  $10^5$  個なので、 $10^5$  回移動した後の状態を求められれば十分です。この状態と  $10^{100}$  回移動した後の状態は一致します。また、左から  $i$  番目のマスをマス  $i$  とし、 $dp(i, j)$  を「最初マス  $j$  に居た子供が  $i$  回の移動の後に居るマス」と定義します。すると、 $dp(1, j)$  は  $S_j$  が L なら  $j-1$ 、R なら  $j+1$  です。さらに、任意の  $i_1, i_2, j$  について  $dp(i_1 + i_2, j) = dp(i_2, dp(i_1, j))$  なので、ダブリングにより  $dp(X, j)$  を  $O(N \log X)$  で計算できます。 $X = 10^5$  について求めたいので、十分間に合います。ダブリングとは、 $x^n$  を  $x^1, x^2, x^4, x^8, \dots$  を組み合わせて計算する (例えば  $x^{10} = x^8 \times x^2$  と計算します) 繰り返し二乗法を応用した手法で、今回の場合だと  $dp(10, j) = dp(8, dp(2, j))$  と計算します。ここで、 $dp(2^{n+1}, j)$  は  $dp(2^n, dp(2^n, j))$  と計算します。この手法を用いれば、例えば問題が  $10^{100}$  回の移動後の状態ではなく  $K$  回の移動後の状態を求めるものであったり、L,R といった単純な移動でなく右に  $R_i$  マス ( $R_i$  が負なら左に  $-R_i$  マス) 移動といった場合にも答えを求めることができます。

## E: Max GCD

まず、 $K$  回の操作で  $A$  に対してどのような操作ができるか考えてみましょう。 $A_1+d_1, A_2+d_2, \dots, A_N+d_N$  になるとします。

このとき、各操作で和が変化しないことに注目すると、 $d_1+d_2+\dots+d_N=0$  である必要があります。また、 $d$  のうち正のものの和が  $K$  より大きい場合はありえません。負についても同様であり (和が 0 の時は自動的に満たされますが)、これら全てが満たされる場合はそのように操作可能であることが帰納的に示せます。

答え ( $d$  とします) の候補としては、 $S = A_1 + A_2 + \dots + A_N$  としたとき、 $d$  が  $S$  の約数である必要があります。 $d$  を  $S$  の約数全て試すことにして固定しましょう。このような  $d$  の候補は  $O(\text{sqrt}(N * \max(A)))$  個あります。

まず、 $A_1, \dots, A_N$  を  $d$  で割った余りを  $r_1, r_2, \dots, r_N$  としましょう。0 であるものではなく、これは小さい順にソートされているとします。このとき、 $d$  を正にするものについては最低  $(d - r_i)$  の和の回数の操作が必要です。 $d$  を負にするものについては最低  $r_i$  の和の回数の操作が必要です。それぞれ、 $d$  の倍数回の操作は余分に行うことができます。よって、 $r$  の小さい方から途中までを  $d$  を負に設定し、そこからは正になるようにする形のみを考えればよく ( $r$  の和が  $d$  の倍数であることに注意しましょう)、累積和などを用いて全ての区切り方の  $d$  が正のものの回数の和と負のものの回数を求めることができます。

よって、 $d$  を固定した場合には  $O(N \log N)$  で解けるので、この問題は  $O(\text{sqrt}(N * \max(A)) * N \log N)$  で解けました。

## F: Enclosed Points

まず、空でない部分集合は  $2^N - 1$  通りあります。これらについて直接  $f$  の値を考えるのではなく、各点が長方形に含まれるような集合が何個あるかを各点について求めることを考えます。

ある点  $P$  に注目しているとしましょう。 $P$  を中心に、 $P$  との  $x, y$  座標の大小関係は 4 方向ありますが、このそれぞれに点が存在するかどうかという状態を固定すればこの  $P$  が長方形に含まれるかが決まることになります。ある領域に  $K$  個の点があるとした時、この領域に点が存在する場合は  $2^K - 1$  通りで、点が存在しない場合は 1 通りです。例えばですが、点がそれぞれに存在するかどうかの  $2^4$  通りについて、長方形に  $P$  が含まれる場合にこれらをかけ合わせて足していけばよいです。(  $P$  が集合に含まれるかどうかについても場合分けが必要です )

よって、各点  $(x_i, y_i)$  について  $x < x_i, y < y_i$ 、 $x < x_i, y > y_i$ 、 $x > x_i, y < y_i$ 、 $x > x_i, y > y_i$  それぞれの領域についてそれぞれ何点存在するかが求まれば、答えを求めることができます。これはまず点の座標を  $1 - N$  に圧縮した後、点を  $x$  座標についてソートし、 $x$  の小さい方からと大きい方からで 2 回 segment tree / BIT (fenwick tree) を用いて各  $y$  に存在する点の個数を管理していけば  $O(N \log N)$  で計算できます。

以上でこの問題は  $O(N \log N)$  で解けました。

解答例

# ABC 136 Editorial

DEGwer, drafeard, evima, satashun

August 4, 2019

## A: Transfer

We can put at most  $A - B$  milliliters of water into Bottle 1. Therefore, the amount of water that we will transfer from Bottle 2 would be the smaller of  $A - B$  and  $C$  (milliliters). Therefore, the amount of water that remains in Bottle 2 would be  $C - \min(A - B, C)$ . The following is an implementation example in C++.

---

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main() {
6     int A, B, C; cin >> A >> B >> C;
7     int cap = A - B;
8     int use = min(C, cap);
9     int ans = C - use;
10    cout << ans;
11 }
```

---

## B: Uneven Numbers

(Draft: DEGwer, Preparation and Editorial: evima)

$N$  is not so large, no more than 100 thousands, so it is easy to count the digits of each of  $N$  numbers,  $1, 2, \dots, N$ .

And the most natural way to count the digits of each value would be converting the number into string and count its length. It is not the fastest way, but it is enough for now.

The following two example is implementation examples in Python. You would be able to implement in other languages in similar way.

---

```
1 N = int(input())
2 ans = 0
3 for i in range(1, N + 1):
4     if len(str(i)) % 2 == 1:
5         ans += 1
6 print(ans)
```

---

---

```
1 N = int(input())
2 print(len(list(filter(lambda x: len(str(x)) % 2 == 1, range(1, N + 1)))))
```

---



## C: Build Stairs

Suppose that you choose the operation from right to left; then you will have more choices if you make them as high as possible, so by performing such operations that

- if you don't have to do anything, do nothing
- otherwise, if it's higher than the square by exactly one, decrease it by one
- otherwise, you cannot achieve it no matter what you do, so the answer is No

from left to right, and printing Yes if construction succeeded, the answer can be found in time complexity of  $O(N)$ . Similarly, you can perform it from left to right.

As another solution, let  $M_i$  be the maximum of heights until  $i$ -th square, then it can be judged Yes if  $H_i \geq M_i - 1$  for all  $i = 1, 2, \dots, N$ , and No otherwise. This can be proved recursively.

Let's prove that " $\forall i. H_i \geq M_i - 1 \Leftrightarrow$  the answer is Yes".

$\Leftarrow$  is easy. Its contraposition is that " $\exists i. H_i < M_i - 1 \Rightarrow$  the answer is No", so it's obvious<sup>\*1</sup>.

Let's prove  $\Rightarrow$  inductively.

- If  $N = 1$ , the answer is always Yes.
- If  $N = N' + 1$ , since the proposition is true for  $i = 1, 2, \dots, N'$ , there exist an solution (way of operations to make it non-decreasing) for  $H_1, H_2, \dots, H_{N'}$ . If you overwrite the solution so that you chose to decrease the height by 1 of the square  $i \in \{1, 2, \dots, N'\}$  such that  $H_i = M_N$ , it is still solution for  $H_1, H_2, \dots, H_{N'}$ , and in addition, if you choose to do nothing for  $H_N$ , you can construct a solution for  $H_1, H_2, \dots, H_N$ , and it's Yes.

---

<sup>\*1</sup> Take one  $i$  such that  $H_i < M_i - 1$ , and let  $y$  be such  $j$  that  $H_j = M_x$  holds. Then  $y < x$  and  $H_x \leq H_y - 2$  holds. Therefore, there exist a square that the height of some square left to it is at least two units lower than that of the square, so there is no way to achieve the condition between those two squares.

## D: Gathering Childrens

If you split it into parts of “RR...RLL...L”, the children do not cross the border, so each of them can be considered independently. For example, “RRLRL” can be considered independently for each “RRL” and “RL”. Hereafter, only the sequence like “RR...RLL...L” is considered.

Since there are enough times of movement, all the children gather into the border, that is, “RL”, and nobody is in elsewhere. Moreover, a child who was initially in a square with even number of index and a child initially in odd index do never gather into same square, and conversely, children in same parity of squares all gather into the same square finally. There are even number,  $10^{100}$  times, of movements, so children whose square's initial index is distant from “R” of “RL” by even number of distance gather into “R”, and children whose square's initial index is distant from “L” of “RL” by even number of distance gather into “L”.

This way, the answer can be found in time complexity  $O(N)$ .

It will be useful for implementation if you can use run-length encoding library, which disassembles a string into consecutive same characters (for example, disassembles “AABCCC” into a sequene like (‘A’, 2), (‘B’, 1), (‘C’, 3)).

There is another solution like the following. Since there are at most  $10^5$  squares, it is sufficient if the state after  $10^5$  movements can be found. This states corresponds to the state after  $10^{100}$  movements. Let square  $i$  be the  $i$ -th square counted from left, and  $dp(i, j)$  be “the index of square where children who was initially in square  $i$  moves into after  $j$  movements.” Then,  $dp(1, j)$  is  $j - 1$  if  $S_j$  is L, and  $j + 1$  if it's R. Moreover, for any  $i_1, i_2, j$  it holds that  $dp(i_1 + i_2, j) = dp(i_2, dp(i_1, j))$ , so by using doubling you can find  $dp(X, j)$  in  $O(N \log X)$ . You want to find the value of  $X = 10^5$ , so it's sufficient. The doubling is development of the fast exponentiation algorithm, which is finding  $x^n$  combining  $x^1, x^2, x^4, x^8, \dots$  (for example, in such way that  $x^{10} = x^8 \times x^2$ ), and this time, it is calculated like  $dp(10, j) = dp(8, dp(2, j))$ . Here,  $dp(2^{n+1}, j)$  is  $dp(2^n, dp(2^n, j))$ . By using this technique, you can also obtain the answer even if the problem asks to find the state after  $K$  movements, or the movements are not as simple as L, R but moving  $R_i$  sqaures to the left (if  $R_i$  is negative, move  $-R_i$  squares to the right).

## E: Max GCD

First, let's think what kind of modification can be applied to  $A$  by performing  $K$  operation. Suppose that it is changed into  $A_1 + d_1, A_2 + d_2, \dots, A_N + d_N$ .

Since each operation does not change the sum of the array,  $d_1 + d_2 + \dots + d_N = 0$  has to be 0. Moreover, the sum of positive elements of  $d$  cannot be more than  $K$ . So does that of negative elements (if the sum is 0, it is automatically fulfilled, though), and it can be proved inductively that if all of these conditions holds, the operations can be performed in such way.

Let  $S = A_1 + A_2 + \dots + A_N$ , then the candidate of answer (let's denote it as  $d$ ) should be divisor of  $S$ . Let's fix some  $d$  among the divisor of  $S$ . There are  $O(\sqrt{N * \max(A)})$  candidates of such  $d$ .

First, let  $r_1, r_2, \dots, r_N$  be the remainder of  $A_1, \dots, A_N$  divided by  $d$ . Suppose that there aren't 0 in it, and they are sorted in increasing order. Then, for the element such that  $d$  is positive, the number of operations that should be performed is at least sum of  $(d - r_i)$ , and for the element of negative, it's at least sum of  $r_i$ . Therefore, it is optimal that  $d$  is set to negative value for the former part of  $r$  until certain element, and set to positive for the rest (note that the sum of  $r$  is multiple of  $d$ ), and you can try all the borders and calculate the sum of positive elements of  $d$  and that of negative elements by using cumulative sum.

Therefore, when  $d$  is fixed it can be solved in  $O(N \log N)$ , so this problem can be solved in  $O(\sqrt{N * \max(A)} * N \log N)$ .

## F: Enclosed Points

First, there are  $2^N - 1$  non-empty subsets. Instead of counting the value of  $f$  for each of them, for each point, let's count how many sets there are such that the point is contained in rectangles.

Let's focus on a point  $P$ . Depending on the comparison of the  $x$ - and  $y$ - coordinates with those of  $P$ , the coordinate system is divided into four regions, and if the presence of points in each region is determined, it can also be determined whether this  $P$  is contained in the rectangle or not. When a region contains  $K$  points, there are  $2^K - 1$  subsets of those points such that there exists at least one point in the region, and 1 set such that none of them are in it. For instance, for each of  $2^4$  combinations of existence of each region, you can calculate the product of them if the rectangle contains  $P$  and sum them up. (You also have to check if  $P$  is included in the set)

Therefore, you can find the answer if you can count the number of points in each region  $x < x_i, y < y_i$ ,  $x < x_i, y > y_i$ ,  $x > x_i, y < y_i$  and  $x > x_i, y > y_i$  for each point  $(x_i, y_i)$ . You can achieve this in  $O(N \log N)$  by compressing the coordinates into  $1 - N$ , sorting the points by  $x$ -coordinate, and managing the number of points by their  $y$ -coordinates using segment tree / BIT (fenwick tree) twice, in its increasing and decreasing order.

So this problem can be solved in  $O(N \log N)$ .

[Link to sample code](#)