Tenka1 Programmer Contest/Tenka1 Programmer Beginner Contest 2018 解説

DEGwer

2018/10/27

For International Readers: English editorial starts on page 5.

A: Measure

文字列を読み込み、その長さに応じて別々に処理を行えばよいです。

```
#include<stdio.h>
#include<string>
#include<iostream>
using namespace std;
int main()
{
        string s;
        cin >> s;
        if (s.size() == 2)cout << s << endl;
        else cout << s[2] << s[1] << s[0] << endl;
}
```

B: Exchange

問題文通りにシミュレーションを行えばよいです。

```
#include<stdio.h>
using namespace std;
int main()
{
        int a, b, k;
        scanf("%d%d%d", &a, &b, &k);
        for (int i = 0; i < k; i++)
```

```
        {
                if (i % 2 == 0)b += a / 2, a /= 2;
                else a += b / 2, b /= 2;
        }
        printf("%d %d\n", a, b);
}
```

C: Align

作る列を $p_1, ..., p_N$ としたとき、$p_{i-1} < p_i < p_{i+1}$ や $p_{i-1} > p_i > p_{i+1}$ となるような i は存在しないとしてよいです。これは、そのほかの要素の相対的な順序を保ったまま p_i を列の末尾に移動しても、隣り合う項の差の絶対値の和は減らないことによります。末尾への移動操作で新たにこのような箇所ができる可能性もありますが、その時は末尾から二番目の要素を再び末尾に移動してやることで、最大化したい値を減らさずにこのような箇所をなくすことができます。

さて、ありうるパターンは $p_1 \geq p_2 \leq p_3 \geq ...$ か $p_1 \leq p_2 \geq p_3 \leq ...$ のいずれかになります。これら両方のパターンを試して大きい方を出力することにしましょう。対称性より、前者の場合のみ考えます。

このとき、隣り合う項の差の絶対値の和は、$(p_1 - p_2) + (p_3 - p_2) + (p_3 - p_4) + (p_5 - p_4) + ...$ です。すなわち、各 i に対し、p_i が最大化したい値に何回足される (もしくは、引かれる) かが定まります。

たくさん足される値を大きく、たくさん引かれる値を小さくすることでこの値は最大化できるので、各 p_i につく係数を見て、大きい順に大きい要素を割り当てていけばよいです。よってこの問題を解くことができました。

D: Crossing

選ぶ部分集合の個数を k とします。どの 2 つの部分集合の共通部分サイズも 1 で、また $1, 2, ..., N$ のどの要素もちょうど 2 回しか使われないため、各集合のサイズは (他の集合との交わりの個数に等しく) $k-1$ になります。よって、要素数の合計を見れば、$N = k(k-1)/2$ が分かります。

逆に、$1, 2, ..., k(k-1)/2$ を以下の図 1 のように並べ、以下のように部分集合の組を取れば、条件を満たすことが分かります。ただし、図は各色の線の上にある要素をひとつの部分集合に含めることを意味します。

E: Equilateral

どの 2 つのマンハッタン距離も等しい 3 点の位置関係として、図 2 の左のようなものは不適切であることが簡単な不等式評価で分かります。よって、右のような位置関係を考えます。

このとき、図 2 の長さ a, b, c, d は $b + d = a + c + d = a + b + c$ を満たします。すなわち、$a + c = b = d$ を満たします。特に $b = d$ なので、3 点のうち 2 点は同じ斜め 45 度の直線上に乗ることが分かります。

この斜め 45 度の直線上の 2 点を固定すれば、残りの 1 点としてありうる点は ($a + c = b = d$ より) ある斜め 45 度の直線上の連続する区間です。よって、累積和を用いることで、$O(max(N, M)^3)$ 時間でこの問題を解くことができました。

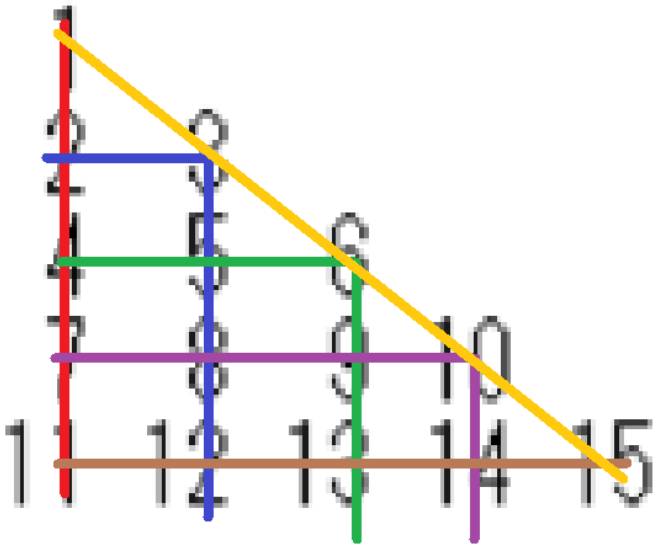

図 1　D 問題

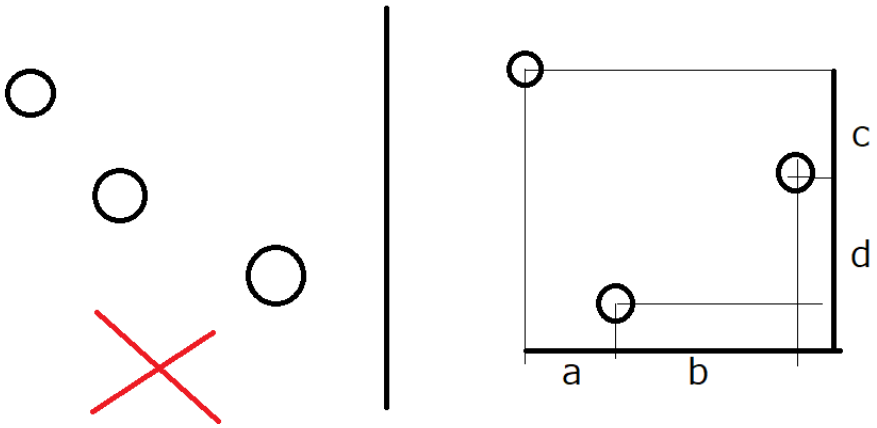

図 2　E 問題

F: Circular

与えられる列 A の全要素が等しい場合は簡単なので、以下そうでないとします。まず、各数の存在する領域は、(円環上で) ある連続した区間を成していないといけないことが分かります。また、行った操作の回数は、1 が連続する個数から 1 を減じたものになることも分かります (これを K とします)。もし存在する領域が連続した区間を成していない数が存在したり、 $K+1$ 個より長く連続している数があるなら、答えは 0 です。

そうでない場合、数を小さい順に見て、その数が元の順列で存在した位置を決めていくことにしましょう。与えられる列 A に i が含まれるとき、数 i を「現れている」と呼ぶことにしましょう。

現れている数 i を見ているとしましょう。i が連続する区間の前に存在する数が i より大きい場合、i のもとの存在位置は (その区間の先頭として) 一意に定まります。また、i が連続する区間の後に存在する数が i より大きい場合、i のもとの存在位置は (その区間の末尾の K 個前として) やはり一意に定まります。なお、もし前後が両方 i より大きく、i の連続する個数が $K+1$ でない場合は、答えは 0 になります。

それ以外の場合、すなわち i の前後に連続する数が両方 i より小さい場合を考えましょう。数のもとの存在位置の定め方より、i の連続する区間の内部や、i の連続する区間の先頭の K 個前まで (特に、末尾の K 個前まで) の場所には、現れている数がもともとあった位置として指定されていません。よって、i の連続する区間の末尾の K 個前から先頭までの区間にあって現れていない数が置かれていない場所のどれも、i がもともとあった位置として指定することができます。

また、現れていない i がもともとあった位置としては、i より小さい数が連続する区間の内部であって、その末尾 K 個でなく、また既にほかの数のもともとあった位置として指定されていない場所のどこでも、指定することができます。

よって、i の前後に連続する数が両方 i より小さい場合や、i が現れていない場合、そのともとあった位置の指定の方法の個数はそれ以前の指定の仕方に依存しません。よってこのような場合は答えに適切な定数をかけるだけで良く、この問題を解くことができました。

Tenka1 Programmer Contest/Tenka1 Programmer Beginner Contest 2018 Editorial

DEGwer

2018/10/27

A: Measure

```
#include<stdio.h>
#include<string>
#include<iostream>
using namespace std;
int main()
{
        string s;
        cin >> s;
        if (s.size() == 2)cout << s << endl;
        else cout << s[2] << s[1] << s[0] << endl;
}
```

B: Exchange

```
#include<stdio.h>
using namespace std;
int main()
{
        int a, b, k;
        scanf("%d%d%d", &a, &b, &k);
        for (int i = 0; i < k; i++)
        {
                if (i % 2 == 0)b += a / 2, a /= 2;
                else a += b / 2, b /= 2;
        }
        printf("%d %d\n", a, b);
}
```

C: Align

Suppose that after rearranging the sequence, we get $p_1, ..., p_N$.

We can assume that there is no i that satisfies $p_{i-1} < p_i < p_{i+1}$ or $p_{i-1} > p_i > p_{i+1}$. If such i exists, we can move p_i to the end of the sequence, and the sum of absolute differences never decreases. (If this operation results in new increasing (or decreasing) three consecutive elements at the end of the sequence, swap the last two elements again.)

Thus, there are two cases:

- $p_1 \geq p_2 \leq p_3 \geq ...$
- $p_1 \leq p_2 \geq p_3 \leq ...$

(Strictly speaking, there are some other cases like $p_1 < p_2 = p_3 > p_4$, but we can eliminate them with the same observation as above).

Since these two cases are symmetric, let's describe the first case. We want to maximize the value $(p_1 - p_2) + (p_3 - p_2) + (p_3 - p_4) + (p_5 - p_4) + ...$ Here, we can forget about the constraint $p_1 \geq p_2 \leq p_3 \geq ...$ because when this condition doens't hold the value never becomes the maximum.

For example, when $n = 5$, $(p_1 - p_2) + (p_3 - p_2) + (p_3 - p_4) + (p_5 - p_4) = (+1) * p_1 + (-2) * p_2 + (+2) * p_3 + (-2) * p_4 + (+1) * p_5$. To achieve the maximum, we should assign the greatest elements to the position with the greatest coefficients. So, in the decreasing order of elements, we should assign them to positions $3, 1, 5, 2, 4$ in this order.

For general n, we can compute coefficients in a similar way, sort them, and assign the greatest elements to the position with the greatest coefficients.

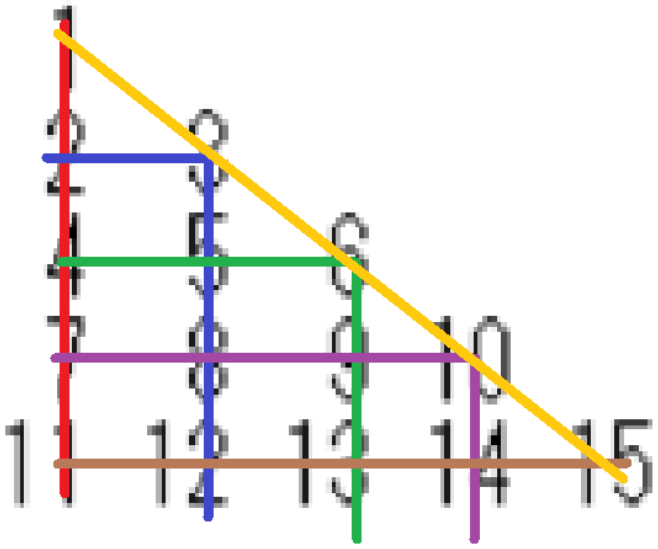

図 1　Problem D

D: Crossing

Let k be the number of subsets we choose.

From the condition in the statement, for each pair (i, j) such that $1 \leq i < j \leq k$, there must be exactly one element that is contained in both S_i and S_j. Also, all elements must appear this way, because all elements are contained in exactly two sets. Thus, once we fix the value k, we can essentially uniquely determine subsets. N must be the number of pairs (i, j), thus $N = k(k-1)/2$.

We first find k that satisfies $N = k(k-1)/2$ (If such k doesn't exist, there's no answer). Then, we can uniquely determine the subsets except for labelling.

For example, when $k = 4$,

- There is an element contained in S_1 and S_2. Let's call it A.
- There is an element contained in S_1 and S_3. Let's call it B.
- There is an element contained in S_1 and S_4. Let's call it C.
- There is an element contained in S_2 and S_3. Let's call it D.
- There is an element contained in S_2 and S_4. Let's call it E.
- There is an element contained in S_3 and S_4. Let's call it F.

Then we get $S_1 = \{A, B, C\}$, $S_2 = \{A, D, E\}$, and so on (and somehow assign the letters to integers between 1 and N).

The following picture illustrates the case $k = 6$.

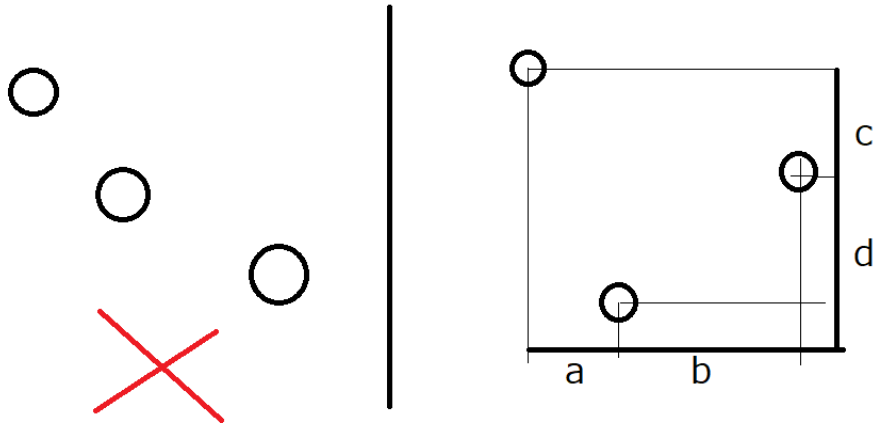

図 2　Problem E

E: Equilateral

There are two types of arrangements of three points on a plane: two points form a diagonal of a rectangle and the other point is inside the rectangle (like the picture on the left), or not (like the picture on the right).

It's easy to see that left-type arrangement never satisfies the condition in the statement, so let's consider the right-type arrangement, and define the lengths a, b, c, d as in the picture.

There lengths must satisfy $b + d = a + c + d = a + b + c$. Thus, we get $b = d$. This means that two points must be on the same diagonal (whose slope is 45 degrees).

Let's fix these two points. For example, suppose that these points are (x, y) and $(x + d, y + d)$.

Then, the candidates of the coordinates of the other point are:

- $(x - d + i, y + d + i)$ for $0 \leq i \leq d$.
- $(x + d + i, y - d + i)$ for $0 \leq i \leq d$.

With proper pre-computation (similar to prefix sums), we can count the number of existing points among these candidate coordinates in $O(1)$. Since there are $O(max(N, M)^3)$ ways to fix two points on the same diagonal, we can solve the entire problem in $O(max(N, M)^3)$.

Note that it's important to avoid double-countings. One possible way to do this is, in each phase, for the third point, only consider these coordinates:

- $(x - d + i, y + d + i)$ for $0 \leq i < d$.

Then after each phase rotate the entire board by 90 degrees. After four phases, each valid triplet will be counted once.

F: Circular

Please wait for a while.