ARC 102 解説

DEGwer

2018/09/01

For International Readers: English editorial starts on page 6.

A: Pair

K 以下の偶数は $\lfloor K/2 \rfloor$ 個、奇数は $\lceil K/2 \rceil$ 個あるので、これらを足し合わせればよいです。

```
#include<stdio.h>
int main()
{
        int k;
        scanf("%d", &k);
        printf("%d\n", (k / 2)*((k + 1) / 2));
}
```

B: Ruined Square

以下のコードのような式で、残りの点の座標を求めることができます。

```
#include<stdio.h>
using namespace std;
int main()
{
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        int x = c - a, y = d - b;
        printf("%d %d %d %d\n", c - y, d + x, a - y, b + x);
}
```

C: Triangular Relationship

K が奇数の時、a, b, c を K で割ったあまりはすべて 0 である必要があります。K が偶数の時、a, b, c を K で割ったあまりはすべて 0 であるか、あるいはすべて $K/2$ である必要があります。このような組の個数は、N 以下で K で割って 0 あまるものの個数と $K/2$ あまるものの個数から求めることができるので、この問題を解くことができました。

D: All Your Paths are Different Lengths

$2^r \leq L$ なる最大の r を取ります ($r \leq 19$ に注意)。まず、以下のようにすれば、長さ 0 以上 $2^r - 1$ 以下のパスをすべて 1 本ずつ作ることができます。

- $r+1$ 個の頂点 $1, ..., r+1$ を用意し、$N = r+1$ とする
- 頂点 i から 頂点 $i+1$ へ、長さ 0 の辺と長さ 2^i の辺を 1 本ずつ張る

さて、こうして作ったグラフの頂点 t から頂点 N に長さ X の辺を張れば、長さ $X, X+1, ..., X+2^{t-1}-1$ のパスを 1 本ずつ作ることができます。よって、$t = N-1, ..., 1$ の順に見て、$L - 2^{t-1} \geq 2^r$ なら頂点 t から頂点 N に長さ $L - 2^{t-1}$ の辺を張り、L を 2^{t-1} だけ減らし、そうでなければ何もしないことを繰り返せばよいです。

E: Stop. Otherwise...

どの 2 つのサイコロの出目の和も t にならないような出目の組の個数を高速に数えることができれば、この問題を解くことができます。

t が奇数の時、条件は $a + b = t$ となるような各 $1 \leq a, b \leq K$ に対し、a, b のどちらか片方は現れないことです。このような (a, b) の組の個数を p とすれば、p 個の組のうちいくつか (q 個とする) については組の目の両方とも出目として現れず、残りについては片方のみ現れることになります。ここで、q を全通り試せばこの場合の数は簡単な二項係数 (と 2 べき) で表せる形になるので、この場合は $O(K)$ 時間で解くことができます。

t が偶数の時、条件は上記のものに $K/2$ が 1 個以下しか現れないという条件を加えたものです。よって、$K/2$ がいくつ (0 個または 1 個) 現れるかで場合分けをし、上と同様に解けばよいです。

時間計算量は $O(K^2 + N)$ です。

F: Revenge of BBuBBBlesort!

列 $\{a_i\} = (1, 2, ..., N)$ から $a_{i-1} < a_i < a_{i+1}$ なる連続 3 項を反転する (以下「i で操作する」) ことを繰り返して列 p を作れるか判定すればよいです。このようにして作れる列 p の性質を見ていくことにしましょう。

まず、隣接 2 要素の両方で操作することはないことがわかります。これは以下のように証明されます。

証明: i で操作する前に、$i-1$ や $i+1$ で操作したことがあるとする。最後に $i-1$ で操作したなら $p_{i-1} > p_i$ に、最後に $i+1$ で操作したなら $p_i > p_{i+1}$ になっているので、i では操作できない。

特に、i で操作したなら、i の位置にある数はそれ以降変化することはありません。

以下、$l+1, l+3, ..., r-1$ のすべてで操作しており、$l-1, r+1$ では操作していないような区間 $[l, r]$ に列を分解して考えます。これらの区間に対して操作は独立に行われるので、以下ひとつの区間についてのみ考えます。

$l+1, l+3, ..., r-1$ の位置にある数は ($l, l+2, ..., r$ では操作しないので)($l+1, l+3, ..., r-1$ のまま) 変化することはありません (これらをピボットと呼ぶことにしましょう)。ピボットでない数が操作によって移動したとします。もしこの数が左に移動したなら、この数のすぐ右にあるピボットの値はこの数より小さく、よってそれ以降この数は現在の位置より右に移動することはないことが分かります。右に移動した場合も同様に、それ以降この数が現在の位置より左に移動することはありません。この議論はすべての操作について成り立つので、ピボットでない数は左に移動し続けるか右に移動し続けるかのどちらかになります (区間の取り方より、移動しないことはありません)。

さて、左に移動し続ける数同士の順番が変化するなら、それらが入れ替わるときに片方は右に移動していることになって矛盾します。よって、初期状態より左に移動した数は、常にその相対的な順序を保つことになります。同様に、初期状態より右に移動した数も、常にその相対的な順序を保つことになります。

逆に、これらの条件が満たされるとき、すなわち各区間 (区間への分解は上記の議論より列 p から定まる) においてその偶数番目の数は移動せず、奇数番目の数はすべて移動し、左に移動する数たちと右に移動する数たちはそれぞれその相対的な順序を保つ場合、数が (まだ右に移動する数 ピボット まだ左に移動する数) という順に並んでいる場所を一個取って操作を行うことを繰り返すことで p を作ることができることが分かります。

よって、以上の条件を満たすかどうかを判定すればよく、これは線形時間で可能なので、この問題を解くことができます。

ARC 102 Editorial

DEGwer

2018/09/01

A: Pair

There are $\lfloor K/2 \rfloor$ even numbers and $\lceil K/2 \rceil$ odd numbers among positive integers up to K. The answer is the product of these values.

```
#include<stdio.h>
int main()
{
        int k;
        scanf("%d", &k);
        printf("%d\n", (k / 2)*((k + 1) / 2));
}
```

B: Ruined Square

You can compute the coordinates of the reaming points as in the code below:

```
#include<stdio.h>
using namespace std;
int main()
{
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        int x = c - a, y = d - b;
        printf("%d %d %d %d\n", c - y, d + x, a - y, b + x);
}
```

C: Triangular Relationship

When K is odd, a triplet (a, b, c) satisfies the conditions when all of them are divisible by K. It's easy to count such triplets.

When K is even, a triplet (a, b, c) satisfies the conditions when all of them are divisible by K, or when all of them are equivalent to $K/2$ modulo K. We can count such triplets by counting the number of integers up to N that are equivalent to 0 or $K/2$, modulo K.

D: All Your Paths are Different Lengths

Solution 1.

Let's take the maximum integer r such that $2^r \leq L$.

First, consider the following graph:

- There are $N = r + 1$ vertices numbered $1, \ldots, N$.
- For each i, add two edges from vertex i to vertex $i + 1$. One of them has length 0, and the other has length 2^{i-1}.

In this graph, there are 2^r paths from vertex 1 to vertex N, and their lengths are $0, \ldots, 2^r - 1$.

What happens if we add an edge of length X from vertex t to vertex N? This way we can add $X, X + 1, \ldots, X + 2^{t-1} - 1$ to the set of lengths.

By adding edges of this type properly (by writing L as a sum of powers of two using binary representation), we can get a desired graph.

Solution 2.

Suppose that we have a graph G that satisfies the condition for $L = X$. We'll show how to construct solutions for $L = X + 1$ and $L = 2X$ using this graph. By repeating these operations properly, we can get a desired graph.

- $L = X + 1$ is easy: just add an edge of length X from source to sink to G.
- $L = 2X$ can be constructed as follows. First, double the lengths of all edges in G. Now the set of lengths is $0, 2, \ldots, 2X - 2$. Then, add a new vertex to G (it will be the new sink), and add two edges from the old sink to the new sink, with lengths 0 and 1. Now the set of lengths is $0, 1, 2, \ldots, 2X - 1$, and this is what we want.

E: Stop. Otherwise...

Let's handle each query independently. For a fixed value of t, we want to count the number of combinations of dice such that the sum of no two dice is t.

For simplicity, we say "x exists" when at least one die shows x. Suppose that t is odd. In this case, the condition can be restated as follows: for each pair (a, b) such that $a + b = t$, at least one of a or b should not exist. Let's divide integers $1, \ldots, K$ into pairs, such that the sum of two numbers in each pair is t (and some numbers will be left unpaired).

Let p be the number of pairs formed this way. Then, the combination of dice must be of the following form for some integer q:

- For q pairs, exactly one of the two numbers in the pair exists.
- For the remaining $p - q$ pairs, both numbers don't exist.
- $K - 2p$ numbers are left unpaired: we don't care if these numbers exist.

There are $\binom{p}{q}2^q$ ways to choose the set of existing numbers (among paired numbers). After that, the number of ways to distribute the number of occurrences of those numbers among N dice are equal to the number of solutions to the following equation:

- $x_1 + \cdots + x_q + y_1 + \ldots + y_{K-2p} = N$
- $x_1, \cdots, x_q$ are positive integers
- $y_1, \cdots, y_{K-2p}$ are non-negative integers

Here, $x_1, \cdots, x_q$ corresponds to the number of occurrences of each existing paired numbers, and $y_1, \cdots, y_{K-2p}$ corresponds to the number of occurrences of each unpaired numbers.

This is a well-known task and the answer can be represented by a binomial coefficient. By trying all possible values of q, we can count the desired number in $O(K)$ time (assuming that we do some pre-computation and we can get binomial coefficients and powers in $O(1)$).

When t is even, there is an additional constraint that the number of occrrences of $t/2$ must be either 0 or 1. Try both possibilities, and the remaining part is the same as above.

This solution works in $O(K^2 + N)$ time.

F: Revenge of BBuBBBlesort!

Let's see the operations in the reverse order: we start with a sequence $\{a_i\} = (1, 2, ..., N)$, and we repeat performing operations of the following type: Choose i such that $a_{i-1} < a_i < a_{i+1}$, and reverse their orders (we call it "an operation on i"). We want to check if we can get p this way.

First, we'll prove that we can never make operations in two adjacent positions. Suppose that before performing an operation on i, we have performed operations on $i-1$ or $i+1$. If the last operation on $i-1$ or $i+1$ was on $i-1$, after the opeartion, $a_{i-1} > a_i$ holds. We can never change this inequality: an operation on $i-2$ only makes a_{i-1} greater, by the assumption above we don't make operations on $i+1$, and operations on other positions obviously don't affest this inequality. Similarly, if the last operation was on $i+1$, $a_i > a_{i+1}$ holds, and we can never change this inequality. Thus, we can't make an opeation on i, and we get a contradiction.

Let's split the sequence into intervals as follows: $a_l, \ldots, a_r$ forms an interval if we perform operations on all of $l+1, l+3, ..., r-1$, but not on $l-1, r+1$. All elements move within these intervals, thus we can consider these intervals independently. From now on, we only considers the interval $a_l, \ldots, a_r$.

We start with $a_l, \ldots, a_r = l, \ldots, r$. How can we change these numbers by performing operations only at $l+1, l+3, ..., r-1$?

Elements at positions $l+1, l+3, ..., r-1$ will never move. Let's call these elements "pivots".

Consider a non-pivot element x in this interval. From the definition of intervals, this element must move (unless $l = r$). Suppose that it moves to the left in the first operation that involves this element. Then, after that, this element is always greater than the pivot to the right of it; and it can never move to the right in the future. We can do the same observaion when it moves to the right first. Thus, each non-pivot element keeps moving in the same direction.

Consider two non-pivot elements that moves to the left. The relative order of these elements will not change: to change their relative order, we must swap them, but this is impossible because they must keep moving left. We can say the same thing for elements moving right.

On the other hand, these conditions are sufficient; we can achieve the goal by repeating operations on consecutive three elements such that ((An element that wants to go right), (pivot), (An element that wants to go left)).

To summarize, the conditions are:

- $a_{l+1}, a_{l+3}, \ldots, a_{r-1}$ must not change, and other elements must change.
- If we only consider all elements that moves to the left, they are increasing from left to right.
- If we only consider all elements that moves to the right, they are increasing from left to right.

From the first condition, we can uniquly determine the way to split the sequence into intervals; after that, the remaining two conditions can be checked in linear time.