ARC084/ABC077 解説

DEGwer

2017/11/04

For International Readers: English editorial starts on page 5.

A: Accepted...?

改行を無視して 6 文字を読み込み、1 文字目と 6 文字目、2 文字目と 5 文字目、3 文字目と 4 文字目がそれぞれ等しいかどうかを調べればよいです。

以下の実装では、1 文字目から 3 文字目までを配列 a に、4 文字目から 6 文字目までを配列 b に、それぞれ読み込んでいます。

```
#include<stdio.h>
int main()
{
        char a[10], b[10];
        scanf("%s%s", a, b);
        if (a[0] == b[2] && a[1] == b[1] && a[2] == b[0]) printf("YES\n");
        else printf("NO\n");
}
```

B: Different Distribution

答えの候補を小さい順に試していきます。i^2 が n を初めて超えたときの、$(i-1)^2$ が答えです。

```
#include<stdio.h>
int main()
{
        int num;
        scanf("%d", &num);
        for (int i = 1;; i++)
        {
                if (i*i > num)
                {
                        printf("%d\n", (i - 1)*(i - 1));
                        break;
                }
        }
}
```

C: Snuke Festival

$A_i < B_j < C_k$ なる組 (i, j, k) の個数を数える問題です。j を固定すれば、求める個数は $A_i < B_j$ なる i の個数と、$B_j < C_k$ なる k の個数の積となります。

よって、全ての j に対し、上述の i, k の個数を数えて掛け合わせ、足し合わせればよいです。

では、i の個数や k の個数はどのように求めればいいでしょうか？　愚直に配列をすべて見ると、$O(N^2)$ となり間に合いません。

配列 A, C をあらかじめソートしておきます。すると、i や k の個数は、二分探索を用いて求めることができます。C++ の場合、標準ライブラリに用意されている lower_bound 関数や upper_bound 関数を用いるのが良いでしょう。

時間計算量は $O(NlogN)$ となり、間に合います。

D: Small Multiple

全ての正整数は、1 から始めて、以下の 2 つの操作を繰り返すことで作ることができます。

- 今の数に 1 を足す。このとき、各桁の和は 1 増える。(1 の位が 9 のときはこの限りではありませんが、この問題を解くにあたっては考慮しなくていいことをあとで示します)
- 今の数を 10 倍する。このとき、各桁の和は変わらない。

さて、全ての正整数を頂点とし、以上の操作の「今の数」から「新しい数」に辺をはったグラフを考えます。求めるべきは、1 から K の倍数のうちのいずれかへの、このグラフ上での最短経路長に 1 を足したものです。

このままでは頂点数は無限ですが、辺のはられ方を考えれば、各正整数に対応する頂点たちは $modK$ で同一視できることが分かります。よって、各頂点を $modK$ で同一視した K 頂点のグラフ上で、1 から 0 への最短路を求めればよく、これは 01BFS を用いることで $O(K)$ 時間で求めることができます。1 の位が 9 の整数に 1 を足す遷移は、最短路のアルゴリズムを思い出せば行き先の頂点がすでに訪れられていることがわかるので、特別に扱う必要はありません。

なお、01BFS とは、deque (両端キュー) を用意し、コスト 0 の辺の遷移は deque の先頭に、1 の辺の遷移は deque の末尾に要素を追加することによる、幅優先探索のアルゴリズムです。

E: Finite Encyclopedia of Integer Sequences

数列に使える整数の種類数 K が偶数の場合、最初の整数が $K/2$ 以下である数列と $K/2+1$ 以上である数列の個数は等しいので、答えは $K/2$ の後に K が $N-1$ 個続いた数列になります。以下 K が奇数の場合を考えます。

$(K+1)/2$ が N 個続いた数列 (これを B とおく) の前にある数列の個数と、後ろにある数列の個数の差はいくつでしょうか？　ある数列 X が B の前にあるとき、X の各要素 X_i を $K+1-X_i$ で置き換えれば、X が B の接頭辞でない限りは、B の後ろにきます。逆に、ある数列 X が B の後ろにあるとき、同様に置き換えれば、X が B の接頭辞でない限りは、B の前にきます。

よって、B の接頭辞以外の数列については B の前にある数列と後ろにある数列の間で一対一対応が成り立つため、B の前後の数列の個数の差は、B の接頭辞の個数、すなわち $N-1$ になります。すなわち、求める数列は、B の $\lfloor \frac{N-1}{2} \rfloor$ 個前にある数列になります。

さて、これはどのように求めればいいでしょうか？　1 個前の数列を得る操作は、

- 末尾が 1 の場合、取り除く
- 末尾が 1 でない場合、その整数を 1 減らし、さらに N 文字になるまで末尾に K を付け加える

という操作です。これはならし $O(1)$ 時間で実装できるので、この問題を $O(N)$ 時間で解くことができました。

F: XorShift

与えられた 2 進数たちを、$\mathbb{F}_2$ 上の多項式として見ます。すなわち、2 進数 $a_k a_{k-1} \ldots a_1 a_0$ を、多項式 $a_k x^k + a_{k-1} x^{k-1} + \ldots + a_1 x + a_0$ と同一視し、さらにその係数は $mod2$ で見ることにします。

操作を多項式の言葉に翻訳します。数を 2 倍することは多項式を x 倍することに、数同士の xor を取ることは多項式同士の和を取ることに対応します。

多項式を 2 つとってきます。次数の大きいほうを P 、小さいほうを Q として、その次数の差を d とします。Q を d 回 x 倍し、P と足し合わせることで、P より次数の低い多項式を作ることができます。こうしてできた多項式を新しい P として (適宜 P, Q を入れ替えて) 同様の操作を続けるアルゴリズムは、ユークリッドの互除法にほかなりません。すなわち、2 つの多項式の GCD を黒板に書くことができます。

これを繰り返せば、最初に黒板に書かれた多項式たちすべての GCD である多項式 P を黒板に書くことができます。また、全ての多項式は P を因子に持ち、操作によってその性質が崩される

ことはないため、黒板に書ける多項式は P の倍数のみです。さらに、x 倍をする操作と和を取る操作を適切に繰り返せば、P の任意の多項式倍を黒板に書くことができます。よって、黒板に書ける多項式は P の倍多項式全てです。

この多項式たちの中で、2 進数として見て X 以下のものの個数を数えましょう。P の次数を d とします。P の倍多項式の d 次以上の係数がすべて定まれば、残りの係数は一意に定まります。d 次以上の部分が X の d 次以上の部分より (2 進数としてみて) 小さいようなものに対しては、ちょうど 1 個の P の倍多項式が定まります。等しいものに対しては、その $d-1$ 次以下の係数を実際に求め (これは高次の係数から順に決めて行くことで求められます)、X より小さいかどうかを調べればよいです。

時間計算量は、GCD は 1 回あたり $O(d^2)$ 時間で求められ (ただし、d は与えられる整数の桁数)、$O(Nd^2)$ となります。また、bitset などを用いて xor を並列的に計算することで、($m = 32$ または 64 として) $O(Nd^2/m)$ 時間で計算することも可能です。

ARC084/ABC077 Editorial

DEGwer

2017/11/04

A: Accepted...?

```
#include<stdio.h>
int main()
{
        char a[10], b[10];
        scanf("%s%s", a, b);
        if (a[0] == b[2] && a[1] == b[1] && a[2] == b[0]) printf("YES\n");
        else printf("NO\n");
}
```

B: Different Distribution

```
#include<stdio.h>
int main()
{
        int num;
        scanf("%d", &num);
        for (int i = 1;; i++)
        {
                if (i*i > num)
                {
                        printf("%d\n", (i - 1)*(i - 1));
                        break;
                }
        }
}
```

C: Snuke Festival

We want to compute the number of triplets (i, j, k) such that $A_i < B_j < C_k$. If we fix j, the number of such triplets is the product of the number of i such that $A_i < B_j$ and the number of k such that $B_j < C_k$. Therefore, for each j, we compute the number of i, k mentioned above, and we can get the answer.

How can we compute the number of such i (or k) for all j? Straightforward solution works in $O(N^2)$ and too slow.

Let's sort A, C in advance. Then, the number of such i (or k) can be computed using a binary search.

This solution works in $O(NlogN)$ time.

D: Small Multiple

All positive integers can be obtained by performing the following two types of operations to 1 zero or more times:

- Add 1. By this operation, the sum of digits increases by one (unless the last digit is 9, but this is not important).
- Multiply the current number by 10. The sum of digits remains the same.

Construct a graph with infinite number of vertices. Each vertex corresponds to a positive integer. In this graph,

- For each x, we add an edge from x to $x+1$ with cost 1.
- For each x, we add an edge from x to $10x$ with cost 0.

The answer is (the shortest distance from 1 to one of multiples of K in this graph) plus one.

We can compree this graph to K vertices. When $x \equiv y \pmod{K}$, assume that x and y are the same vertex. Then, in this graph, the answer is the shortest distance from 1 to 0, plus one.

This can be done in $O(K)$ by 01-bfs (a variant of Dijkstra's algorithm, when all costs are 0 or 1, you can use a deque instead of priority_queue).

Note: Isn't the trouble with '9' important? Suppose that the algorithm above finds an invalid solution that contains a digit "10" at position i. However, due to the periodicity of powers of tens, (when K is comprime to 10) we can find another j such that 10^i and 10^j are the same in modulo K, so we can move one digit from i to j. Even when K is not coprime to 10, we can make it periodic by attaching sufficient number of zeroes at the end.

E: Finite Encyclopedia of Integer Sequences

If K is even, exactly half of sequences starts with an integer $K/2$ or less. Thus, the answer is $K/2$ followed by $N-1$ occurrences of K.

Suppose that K is odd. Let B be a sequence that contains N occurrences of $(K+1)/2$ (this is very close to the middle). What is the difference between the number of sequences before B and after B?

For a sequence X, let $f(X)$ be a sequence such that X_i in X is replaced with $K+1-X_i$. This is almost an bijection from a sequence after B to a sequence before B. The only exception happens for prefixes of B. Therefore, the difference is $N-1$ (the number of non-trivial prefixes of B), and B is the $ceil(X/2) + floor(N/2)$-th sequence.

We start with B, and repeat the following operation $\lfloor \frac{N-1}{2} \rfloor$ times:

- If the current sequence ends with 1, remove it.
- Otherwise, decrement the last element by 1, and while the sequence contains less than N elements, attach K at the end.

This operation takes $O(1)$ on average, so the amortized complexity of this solution is $O(N)$.

F: XorShift

Let's interpret the given binary numbers as polynomials on $\mathbb{F}_2$. That is, a binary number $a_k a_{k-1} \ldots a_1 a_0$ corresponds to a polynomial $a_k x^k + a_{k-1} x^{k-1} + \ldots + a_1 x + a_0$, here the coefficients are computed modulo 2.

You can perform the following operations for the polynomials:

- Multiply a polynomial by x. (corresponds to "double")
- Compute the sum of two polynomials. (corresponds to "xor")

Let P, Q be two given polynomials, and suppose that $degP > degQ$. By computing $P' = P + (Q * x^{degP - degQ})$, you can get a polynomial whose degree is smaller than P. Then, you can repeat the same operation for two polynomials P' and Q (when necessary, you swap P and Q). This is similar to Euclid's algorithm, and you will eventually get $gcd(P, Q)$.

Let G be the GCD of all given polynomials. As we see above, we can write G on the blackboard. Then, by using the two types of operations properly, you can write all multiples of G (H is a multiple of G if H can be written as $H = H'G$ using some polynomial H'). On the other hand, all polynomials on the blackboard are always divisible by G.

Therefore, we want to compute the number of polynomials P such that:

- P is a multiple of G.
- P is less than or equal to X (as a binary number).

Let $d = degG$ and $D = degX$. If we know all coefficients of a polynomial except for the last (least significant) d coefficients, we can uniquely determine the remaining d coefficients to make it a multiple of G. Therefore, if the first $D - d + 1$ coefficients of P are smaller than that of G (as a binary number), we can uniquely determine P. If the first $D - d + 1$ coefficients are the same, we compute the remaining d coefficients and compare it with X.

It takes $O(d^2)$ time per one gcd computation (where d is the number of digits), thus the complexity of this solution is $O(Nd^2)$. This is fast enough. You can also use bitsets to achieve $O(Nd^2/m)$ ($m = 32 or 64$).