CADDi 2018 / CADDi 2018 Beginners 解説

semiexp, DEGwer, evima

2018 年 12 月 22 日

For International Readers: English editorial starts on page 5.

A: 12/22

各桁の数字を一つずつ確かめればよいです．N を整数として読み込むより，文字列として読み込んだほうが扱いやすいです．

- C++ による解答例: `https://atcoder.jp/contests/caddi2018b/submissions/3836444`
- Ruby による解答例: `https://atcoder.jp/contests/caddi2018b/submissions/3836446`
- Python3 による解答例: `https://atcoder.jp/contests/caddi2018b/submissions/3836447`

B: AtCoder Alloy

$H \leq A_i$ かつ $W \leq B_i$ ならば i 番目の素材から $H \times W$ の板を切り出すことができます．

- C++ による解答例: `https://atcoder.jp/contests/caddi2018b/submissions/3836462`
- Ruby による解答例: `https://atcoder.jp/contests/caddi2018b/submissions/3836464`
- Python3 による解答例: `https://atcoder.jp/contests/caddi2018b/submissions/3836465`

C: Product and GCD

$a_1, \ldots, a_N$ の最大公約数を g とすると，a_i のそれぞれは g の倍数であることから，$P = a_1 \times \cdots \times a_N$ は g^N の倍数です．よって，P の素因数分解を ${p_1}^{k_1} \ldots {p_\ell}^{k_\ell}$ とすると，$g = {p_1}^{k'_1} \ldots {p_\ell}^{k'_\ell}$ と表すことができ，$Nk'_i \leq k_i$ すなわち $k'_i \leq \frac{k_i}{N}$ が成り立ちます．一方，$k'_i = \lfloor \frac{k_i}{N} \rfloor$ としたとき，$a_1 = \cdots = a_{N-1} = g, a_N = \frac{P}{g^{N-1}}$ とすることで，$a_1 \times \cdots \times a_N = P$ かつ $a_1, \ldots, a_N$ の最大公約数が g となります．よって，${p_1}^{\lfloor \frac{k_1}{N} \rfloor} \ldots {p_\ell}^{\lfloor \frac{k_\ell}{N} \rfloor}$ を求めればよいです．これは，P を素因数分解することで求められます．

D: Harlequin

結論は次の通りです。

```
print('second' if all(a[i] % 2 == 0 for i in range(N)) else 'first')
```

理屈を説明します。「どの色のりんごも偶数個」という状態を E、「いずれかの色のりんごが奇数個」という状態を O と呼ぶことにします。現在の状態が E の場合、すでに「全色 0 個」(相手が勝った) でなければ、どのようにりんごを食べても食べた色のりんごが奇数個になり、必ず O に移ります。一方で現在の状態が O の場合、奇数個の色のりんごだけを 1 個ずつ食べることで E に必ず移れます。

よって、初期状態が E の場合、後手が先手の「真似」をすると先手は E に閉じ込められ、いずれ「全色 0 個」に至り負けます。初期状態が O の場合は、先手は最初に奇数個の色のりんごだけ食べて後手に E を押し付け、それ以降は後手の真似をすることで勝てます。

なお、コンテスト中にこの結論にたどり着くには、(メモ化) 再帰により小さいケースを解くプログラムを書いたり、$N = 2$ のケースを紙と鉛筆で解くことで実験して仮説を立てる方法もあります。

E: Negative Doubling

次の 2 種類の操作を考えると，問題中の操作と同等であることがわかります：

(a) A_i の値を -2 倍にする．ただし，同じ i に対してこの操作は高々 1 回しか行えない．
(b) A_i の値を 4 倍にする．この操作は 2 回の操作として数える．

操作 (b) では値の符号を変えることはできないことから，操作 (a) は，ある $0 \leq \ell \leq N-1$ に対して，$1 \leq i \leq \ell$ なるすべての A_i に対して行う必要があります．一方，操作 (a) をすべて行うと，符号が正の部分と負の部分について，独立に問題を解くことができます．負の部分については正の部分とまったく同様に解けるため，以降正の部分について考えます．以降，単に操作と行ったら操作 (b) を指すことにします．

正の部分については，次のような問題を考えることになります：

> 操作によって，$A_\ell \leq A_{\ell+1} \leq \cdots \leq A_N$ が成り立つようにするとき，操作は何回必要かを，各 ℓ について求めよ．

ℓ が固定されていれば，これは $i = \ell$ から順に貪欲法により答えを求めることができます．すなわち，$i = \ell, \ell+1, \ldots, N-1$ の順に，$A_i > A_{i+1}$ である限り A_{i+1} に操作を行うときの操作回数を求めればよいです．（実際に 4 倍を繰り返していると値が大きくなりすぎるので，各 A_i に何回の操作を行ったかを持っておくとよいです．）

$dp[\ell][x]$ を，「A_ℓ にちょうど x 回操作を行うことにしたとき，$A_\ell \leq A_{\ell+1} \leq \cdots \leq A_N$ が成り立つようにするのに必要な全操作回数」とします．各 $x \geq 0$ に対して $dp[\ell+1][x]$ がわかっていれば，$dp[\ell][x]$ も求めることができますが，x は大きくなりうるため，単純に DP テーブルの値をすべて持つことはできません．ところで，$10^9 < 4^{15}$ であることから，A_ℓ に 15 回以上操作を行うと，A_i $(i \geq \ell)$ に対しても 1 回以上の操作が必要になります．よって，A_ℓ に行う操作の回数を 1 回増やすと，$A_{\ell+1}, \ldots, A_N$ に対して行う操作も 1 回ずつ増えることがわかります．ゆえに，ℓ を固定したとき，$dp[\ell][x]$ は $x \geq 15$ で公差 $N-\ell+1$ の等差数列になることから，$0 \leq x \leq 15$ で DP テーブルを計算すれば十分です．よって DP テーブルの値が計算できるので，問題を解くことができます．

あとは，操作 (a) を行う範囲をすべて試し，正負それぞれの部分に対して必要な操作回数を求め，全体で必要な操作回数を計算し，その最小値を求めればよいです．

F: Square

マス目の i 行 j 列に書かれる値を $a_{i,j}$ とします。$a_{i,i}$ たちがすべて決まっている状況を最初に考えましょう。

マス $a_{i,i}, a_{i,i+1}, a_{i+1,i}, a_{i+1,i+1}$ からなる 2×2 の領域に書かれた値の和が mod 2 で 0 であることから、$a_{i,i+1} + a_{i+1,i} \equiv a_{i,i} + a_{i+1,i+1}$ が分かります。

次に、マス $a_{i,i}$ を左上、$a_{i+2,i+2}$ を右下とする 3×3 以上の正方形領域について考えれば、上の結果と併せて、$a_{i,i} + a_{i-1,i+1} + a_{i+1,i-1} \equiv 0$ も分かります。

4×4 以上の正方形領域に対する条件はどのように言い換えられるでしょうか？　実はこの条件は、$|i-j| \geq 3$ については $a_{i,j} = a_{j,i}$ が成り立っていることと等価であることが簡単な帰納法により証明できます。

以上より、以下のような基準で各マスに入る値を定めてやることができます。

- $|i-j| \geq 3$ について $a_{i,j}, a_{j,i}$ がともに定まっていてかつ異なるなら矛盾。片方だけ定まっているならもう片方も定まる。どちらも定まっていないなら適当に定め、求める場合の数を 2 倍する
- それ以外に対しては、上述の条件を満たすように決める

さて、上記の「それ以外」の領域は、正方形の対角線に沿った幅 5 の領域に他なりません。この領域を斜めに見ていき、前いくつかのマスをどの値で埋めたかをキーにした bit DP を行うことで、条件を満たす値の埋め方を数え上げることができます。

また、上述の条件たちがすべて $\mathbb{F}_2$ 上での線形制約式 (mod 2 での等式条件) であることを利用し、自由度を数え上げてやる方針でも解くことができます。どちらの解法も、時間計算量は $O(N+M)$ となるように実装できます。

CADDi 2018 / CADDi 2018 Beginners Editorial

semiexp, DEGwer, evima

December 22nd, 2018

For International Readers: English editorial starts on page 200.

A: 12/22

- C++ Solution: `https://atcoder.jp/contests/caddi2018b/submissions/3836444`
- Ruby Solution: `https://atcoder.jp/contests/caddi2018b/submissions/3836446`
- Python3 Solution: `https://atcoder.jp/contests/caddi2018b/submissions/3836447`

B: AtCoder Alloy

- C++ Solution: `https://atcoder.jp/contests/caddi2018b/submissions/3836462`
- Ruby Solution: `https://atcoder.jp/contests/caddi2018b/submissions/3836464`
- Python3 Solution: `https://atcoder.jp/contests/caddi2018b/submissions/3836465`

C: Product and GCD

Let g be the GCD of $a_1, \ldots, a_N$. Since a_i must be a multiple of g for each i, their product $P = a_1 \times \cdots \times a_N$ must be a multiple of g^N. On the other hand, if P is a multiple of g^N, we can actually make the GCD g by assigining $a_1 = \cdots = a_{N-1} = g, a_N = g \times \frac{P}{g^N}$. Thus, the answer is the maximum g such that g^N divides P.

Let $P = {p_1}^{k_1} \ldots {p_\ell}^{k_\ell}$ be the prime factorization of P. Then, g must be of the form $g = {p_1}^{k'_1} \ldots {p_\ell}^{k'_\ell}$, and g^N divides P when $Nk'_i \leq k_i$ for each i. In order to maximize g, we should choose $k'_i = \frac{k_i}{N}$. Thus, the answer is ${p_1}^{\lfloor \frac{k_1}{N} \rfloor} \ldots {p_\ell}^{\lfloor \frac{k_\ell}{N} \rfloor}$.

We can get this value by actually factorizing P.

D: Harlequin

The conclusion is as follows:

```
print('second' if all(a[i] % 2 == 0 for i in range(N)) else 'first')
```

Let us explain the reason. We will call the state E where "there are even apples of every color", and the state O where "there are odd apples of some colors". If the current state is E, unless "there are zero apples of every color" (the opponent won), eating apples in any way will result in odd apples of the colors eaten, and we will always transition to O. On the other hand, if the current state is O, by eating only the colors with odd apples, we can always transition to E.

Thus, when the initial state is E, if the second player "imitate" the first, the first will be locked in E and eventually lose by arriving at "zero apples of every color". When the initial state is O, the first player can win by first eating only the colors with odd apples to force E against the second, then imitating her.

By the way, to reach this conclusion, there are such approaches as writing a program that solves small cases by (memoized) recursion, or solving the case where $N = 2$ by hand, to experiment and set up a hypothesis.

E: Negative Doubling

The operation in the statement is equivalent to the following two types of operations:

(a) Multiply A_i by -2. This operation can be performed at most once for each i.
(b) Multiply A_i by 4. (This operation is counted as "two operations").

Since the sequence must be non-decreasing after the operations, the sequence must start with zero or more negative numbers, followed by zero or more positive numbers. Thus, there exists some i ($0 \leq \ell \leq N$), and we perform the operation (a) for all of the first i terms (and for no other terms).

Let's fix the value of this i. After that, we need to make the sequence non-decreasing by performing operations (b). Now, we can handle the positive part and the negative part independently.

Let's consider the positive part only (negative part is similar). We want to solve the following problem:

> How many operations (b) is required to satisfy $A_\ell \leq A_{\ell+1} \leq \cdots \leq A_N$? Compute this value for each ℓ.

For a fixed ℓ, we can compute this value greedily. That is, in the order $i = \ell, \ell+1, \ldots, N-1$, we repeat performing operations on A_{i+1} while $A_i > A_{i+1}$. (If we actually multiply numbers by 4, the values will be too big. Instead we should keep the number of operations we performed for each term.)

Let $dp[\ell][x]$ be the minimum number of operations we need to satisfy $A_\ell \leq A_{\ell+1} \leq \cdots \leq A_N$, with an additional constraint that we must perform exactly x operations in A_ℓ. If we know the values of $dp[\ell+1][x]$ for all $x \geq 0$, we can get $dp[\ell][x]$, but we can't do this because there are infinite possibilities for x.

Notice that since $10^9 < 4^{15}$, if we perform 15 or more operations on A_ℓ, we must perform at least one operation for all A_i ($i \geq \ell$). Thus, after we perform 15 operations on A_ℓ, each time we perform an additional operation on A_ℓ, we also need one additional operation for each of $A_{\ell+1}, \ldots, A_N$. Therefore, for a fixed ℓ, $dp[\ell][x]$ is an arithmetic sequence in the range $x \geq 15$ (with difference $N - \ell + 1$). We only need to keep the values of $dp[\ell][x]$ for $0 \leq x \leq 15$. This way we can compute the DP table, and $dp[\ell][0]$ corresponds to the value we want to compute.

Afret this, we can compute the optimal answer to the original problem by trying all possible prefixes of performing (a).

F: Square

Let $a_{i,j}$ be the value written on the cell (i, j). Assume that all sums in this editorial are done in modulo 2.

For each pair i, j such that $i < j$, only the sum $a_{i,j} + aj, i$ matters because the cell (i, j) is contained in a square if and only if the cell (j, i) is contained in the square. For simplicity, we first consider the case where $a_{i,j} = 0$ for all cells (i, j) such that $i > j$. (Later we'll describe how to handle general cases.)

In this case, it turns out that if we decide the values on diagonal cells, we can always uniquely determine the values on remaining cells as follows:

- Suppose that $a_{i,i} = x_i$ for each i.
- By considering 2×2 squares, we get $a_{i,i+1} = x_i + x_{i+1}$ for each i.
- By considering 3×3 squares (and the result above), we get $ai, i + 2 = x_{i+1}$ for each i.
- By considering larger squares, we get $a_{i,j} = 0$ for all $j - i \geq 3$.

Thus, we want to compute the number of ways to choose $x_1, \ldots, x_N$ such that it doesn't contradict with the given information.

Each cell filled with an integer adds a constraint on $x_1, \ldots, x_N$ in the following way:

- If $a_{i,i}$ is filled for some i, we know the value of x_i.
- If $a_{i,i+1}$ is filled, we know the value of $x_i + x_{i+1}$.
- If $a_{i,i+2}$ is filled, we know the value of x_{i+1}.
- If $a_{i,j}$ is filled with one for some $j - i \geq 3$, we get a contradiction (thus the answer is zero). If it's filled with zero, we can simply ignore it.

In all cases, the value of a single variable or the sum of two variables (in modulo 2) is given. This is a well-known problem, and the number of possible assignments for $x_1, \ldots, x_N$ can be computed by computing the number of connected components in the graph constructed in a standard way. It's also possible to solve it by DP with bitmasks, because when $x_i + x_j$ is fixed $|i - j|$ is always small.

Both solutions works in $O(N + M)$ time.

Now, how to handle general cases? For each pair i, j such that $i < j$,

- If both (i, j) and (j, i) are fixed, assume that (i, j) is fixed with their sum and (j, i) is fixed with zero.
- If exactly one of (i, j) and (j, i) is fixed, assume that (i, j) is not fixed and (j, i) is fixed with zero. (This conversion doesn't change the distribution of $a_{i,j} + aj, i$).
- If none of (i, j) and (j, i) are fixed, assume that (i, j) is not fixed and (j, i) is fixed with zero. Later, multiply the answer we get by two, for each such pair (i, j).