ABC #069 / ARC #080 Editorial

writer : sugim48

2017 年 8 月 6 日

For International Readers: English editorial starts on page 5.

A : K-City

南北方向の通りの本数は n なので，それらの間の個数は $n-1$ です．同様に，東西方向の通りの本数は m なので，それらの間の個数は $m-1$ です．区画の個数はこれらの積なので，$(n-1)(m-1)$ です．

- C++ のコード例 : https://abc069.contest.atcoder.jp/submissions/1480689
- Java のコード例 : https://abc069.contest.atcoder.jp/submissions/1480691
- Python 3 のコード例 : https://abc069.contest.atcoder.jp/submissions/1480692

B : i18n

s の先頭文字，$|s|-2$，s の末尾文字の順に出力すればよいです．ただし，$|s|$ は s の長さを表します．

- C++ のコード例 : https://abc069.contest.atcoder.jp/submissions/1480693
- Java のコード例 : https://abc069.contest.atcoder.jp/submissions/1480695
- Python 3 のコード例 : https://abc069.contest.atcoder.jp/submissions/1480696

C : 4-adjacent

2 で割り切れる回数によって a の要素を分類します．0 回ならば ① と書き，1 回ならば ② と書き，2 回以上ならば ④ と書くことにします．また，①, ②, ④ の個数をそれぞれ b_1, b_2, b_4 とします．すると，「隣り合う要素の積が 4 の倍数である」という条件は，「① と隣り合う要素は ④ でなければならない」という条件へ言い換えられます．

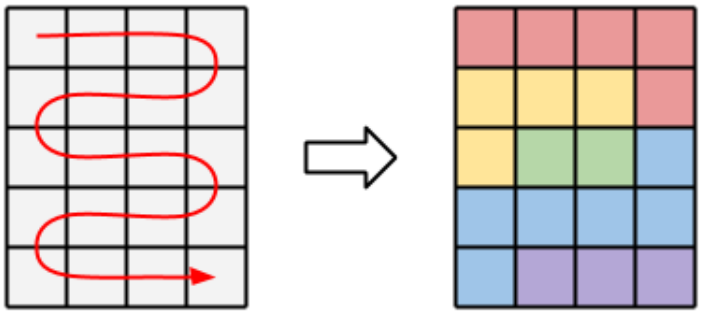
まず，② が存在しない場合を考えます．この場合，①④①④① のように並べるのが最適です．よって，$b_1 \leq b_4 + 1$ ならば `Yes`，そうでないならば `No` となります．

次に，② が存在する場合を考えます．例として ②④①④②②④②② と並べてみます．この例を見ると，ひと繋がりの ② は単体の ① と等価であることが分かります．① はできるだけ少ない方が嬉しいので，② はすべてまとめてひと繋がりにするのが最適です．すると，① を 1 個だけ増やした上で，② が存在し

ない場合へ帰着できます．よって，$b_1 \leq b_4$ ならば `Yes`，そうでないならば `No` となります．具体的には，①④①④①④②②② のように並べるのが最適です．

D : Grid Coloring

次図のようにマスを順序付けした上で，最初の a_1 マスを色 1 で塗り，次の a_2 マスを色 2 で塗り，…，最後の a_N マスを色 N で塗ればよいです．



E : Old Maid

p, q 上の位置を左から順に 0, 1, ..., $N-1$ とします．辞書順で最小の列を求める問題では，「列の先頭から順に，その時点であり得る最小の値を確定していく」という方法が有効です．

まず，q_0 としてあり得る最小の値を確定しましょう．q_0 になり得るのは，p 上で偶数番目の要素です．これらのうち最小の要素を q_0 として確定します．また，この要素の p 上の位置を i とします．次に，q_1 としてあり得る最小の値を確定しましょう．q_1 になり得るのは，p 上で奇数番目の要素であって，位置 i より右にあるものです．これらのうち最小の要素を q_1 として確定します．また，この要素の p 上の位置を j とします．すると，p の残った要素は，3 つの区間 $[0, i), [i+1, j), [j+1, N)$ へ分割されます．続いて，q_2 としてあり得る最小の値を確定しましょう．q_2 になり得るのは，各区間上で偶数番目の要素です．これらのうち最小の要素を q_2 として確定します．また，この要素が区間 S 上にあるとし，この要素の S 上の位置を i' とします．次に，q_3 としてあり得る最小の値を確定しましょう．q_3 になり得るのは，S 上で奇数番目の要素であって，位置 i' より右にあるものです．これらのうち最小の要素を q_3 として確定します．また，この要素の S 上の位置を j' とします．すると，S の残った要素は，位置 i', j' を境にして 3 つの区間へ分割されます．以上のような手続きを繰り返せば，辞書順で最小の q が求まります．

以上の方法を愚直に実装すると，全体で $\mathrm{O}(N^2)$ 時間となり TLE してしまいます．TLE を回避するためには，1) ある区間上の偶数 (奇数) 番目の要素の最小値を求めるステップ，2) 次にどの区間から要素を選ぶかを決めるステップ，の両方を高速化する必要があります．1) については，前処理として，p の偶数 (奇数) 番目の要素だけを取り出したものをセグメント木に乗せておけばよいです．このとき，最小値と同時に位置も得られるようにしておきます．2) については，今あるすべての区間を優先度付きキューに入れておき，各区間の優先度には，その区間の偶数番目の要素の最小値を設定すればよいです．以上の方法は，全体で $\mathrm{O}(N \log N)$ 時間で，十分に高速です．

F : Prime Flip

各カードの向きによって，数列 $a = (a_1, a_2, \ldots)$ を定義します．具体的には，カード i が表ならば $a_i = 1$ で，カード i が裏ならば $a_i = 0$ とします．さらに，a の差分をとった数列 $b = (b_0, b_1, b_2, \ldots)$ を定義します．具体的には，$b_i = a_{i-1} \oplus a_i$ とします．ただし，$a_0 = 0$ とします．すると，カード $[l, r)$ をそれぞれひっくり返すという操作は，b_l, b_r の 0/1 をそれぞれ反転するという操作と等価です．また，すべてのカードが裏向きであるという状態は，すべての b_i が 0 であるという状態と等価です．以上より，問題は次のように言い換えられます．

> 0/1 の列 $b = (b_0, b_1, b_2, \ldots)$ がある．「0 以上の整数 l と 3 以上の素数 p を選び，b_l, b_{l+p} の 0/1 をそれぞれ反転する」という操作を繰り返し行うことができる．すべての b_i を 0 にするために必要な操作回数の最小値を求めよ．

操作の順序は結果に影響しません．また，ある時点で $b_i = 1$ であるならば，後に必ず b_i を反転する操作を行う必要があります．よって，各操作において b_l または b_{l+p} の少なくとも一方は 1 である，と仮定できることが分かります．以上より，問題は次のように言い換えられます．

> マス 0, 1, 2, ... がある．最初，$b_i = 1$ ならばマス i には駒が 1 つ置かれており，$b_i = 0$ ならばマス i は空である．次の操作を繰り返し行うことができる．
>
> - 3 以上の素数 p を選ぶ．ある駒をちょうど p だけ離れたマスへ移動する．移動先のマスに既に駒があるならば，2 つの駒は共に消える．
>
> すべての駒を消すために必要な操作回数の最小値を求めよ．

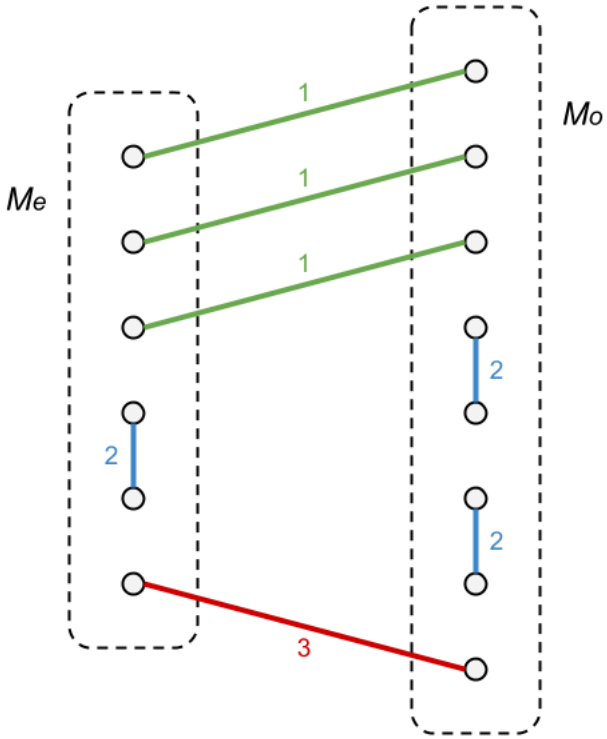
b の定義より，初期状態において駒の個数は偶数です．これを M とします．M 個の駒を $M/2$ 組のペアに分け，各ペアの 2 つの駒を共に消すことにします．このときの操作回数の最小値は，各ペアを消すための最小手数の総和になります．マス i, j $(i \neq j)$ 上の 2 つの駒を共に消すための最小手数を $d(i, j)$ とすると，

$$d(i,j) = \begin{cases} 1 & (|i-j| \text{ が奇数であって，3 以上の素数である}) \\ 2 & (|i-j| \text{ が偶数である}) \\ 3 & (|i-j| \text{ が奇数であって，3 以上の素数でない}) \end{cases}$$

となります (ゴールドバッハ予想を主に用いて示せます)．初期状態において，偶数 / 奇数マス上の駒の個数をそれぞれ M_e, M_o とします．ここで，最小手数が 1 のペアを k $(k \leq M_e, M_o)$ 組作ることができたとします．すると，残った駒たちの最適なペア分けは，偶数マス上の駒たちから $\lfloor (M_e - k)/2 \rfloor$ 組ペアを作り，奇数マス上の駒たちから $\lfloor (M_o - k)/2 \rfloor$ 組ペアを作り，もし $M_e - k, M_o - k$ が奇数ならば，余った偶数マス上の駒と奇数マス上の駒で 1 組ペアを作る，というようになります (次図)．このときの最小手数の総和は

$$k \times 1 + (\lfloor (M_e - k)/2 \rfloor + \lfloor (M_o - k)/2 \rfloor) \times 2 + ((M_e - k)\%2) \times 3$$

となりますが，これは k の増加に対して広義単調減少です．よって，k の最大値を求められればよいことが分かります．これは二部グラフの最大マッチングなので，フローを用いて簡単に解くことができます．


Me
Mo
1
1
1
2
2
2
3

ABC #069 / ARC #080 Editorial

writer : sugim48

2017 年 8 月 6 日

A : K-City

- C++ Code Example : https://abc069.contest.atcoder.jp/submissions/1480689
- Java Code Example : https://abc069.contest.atcoder.jp/submissions/1480691
- Python 3 Code Example : https://abc069.contest.atcoder.jp/submissions/1480692

B : i18n

- C++ Code Example : https://abc069.contest.atcoder.jp/submissions/1480693
- Java Code Example : https://abc069.contest.atcoder.jp/submissions/1480695
- Python 3 Code Example : https://abc069.contest.atcoder.jp/submissions/1480696

C : 4-adjacent

- When an element in a is an odd number, we represent it as ①.
- When an element in a is an even number but not divisible by 4, we represent it as ②.
- When an element in a is divisible by 4, we represent it as ④.

Let b_1, b_2, b_4, be the number of ①, ②, ④. We want to arrange them such that all elements that are adjacent to ① are ④.

In case $b_2 = 0$

The optimal arrangement will look like ①④①④①. If $b_1 \leq b_4 + 1$, the answer is `Yes`, otherwise the answer is `No`.

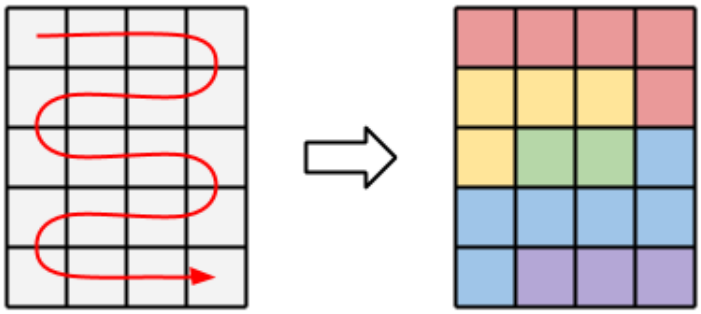
In case $b_2 > 0$

For example an arrangement will look like ②④①④②②④②②. Here, a maximal consecutive sequence of ② is equivalent to ①. Thus, we should concatenate all ② and handle them like a single ①. If

$b_1 \leq b_4$, the answer is `Yes`, otherwise the answer is `No`. In particular, we should arrange them like ①④①④①④②②②.

D : Grid Coloring

We number the cells as follows. Then, we color the first a_1 cells with color 1, the next a_2 cells with color 2, and so on.



E : Old Maid

Here we use 0-based indices, i.e., $p = (p_0, \ldots, p_{N-1}$ and $q = (q_0, \ldots, q_{N-1})$.

Which element can be q_0? It must be at an even position in p. Since we want to minimize q lexicographically, we want q_0 to be minimized. We can determine q_0 as the minimum number among elements in p at even positions. Let $q_0 = p_i$.

Which element can be q_1? If $q_1 = p_j$, j must be odd, and $j > i$. We should choose j that minimizes p_j under the constraints above.

Now, p is divided into (at most) three intervals: $[0, i)$, $[i + 1, j)$, $[j + 1, N)$.

How should we choose q_2? In p, it must be in one of the three intervals above, and the parity of its position must be the same as the parity of the first element of its intervals. We should choose q_3 from the same intervals (in the same way as q_1). By repeating this, we get an $O(N^2)$ solution.

To make it $\mathrm{O}(N \log N)$,

- We should use Range Minimum Query to get the minimum in an interval.
- We should keep all intervals in a priority_queue. The priority of an interval $[l, r)$ is the minimum of p_i such that $l \leq i < r, i \equiv l \pmod 2$.

F : Prime Flip

Define b_i as follows:

- If the card i and the card $i - 1$ are faced in the same way, $b_i = 0$.

- Otherwise, $b_i = 1$.

(Assume that card 0 is face down.)

When you perform an operation on cards $[l, r)$, you flip b_l and b_r. Thus, we can restate the problem as follows:

> You are given a sequence of 0/1, $b = (b_0, b_1, b_2, \ldots)$. In each operation you can choose a (non-negative) integer l and an odd prime p, and flip both b_l and b_{l+p}. Compute the number of operations required to transform the sequence to $0, 0, 0, \ldots$.

Consider each 1 in the sequence as a token. In each operation, you can move a token by a distance of odd prime. When two tokens meets, they disappear altogether.

When there are two tokens at i and j, $d(i, j)$, the cost required to erase them, can be computed as follows:

- If $|i - j|$ is an odd prime, $d(i, j) = 1$.
- If $|i - j|$ is an odd composite, $d(i, j) = 3$.
- If $|i - j|$ is even, $d(i, j) = 2$.

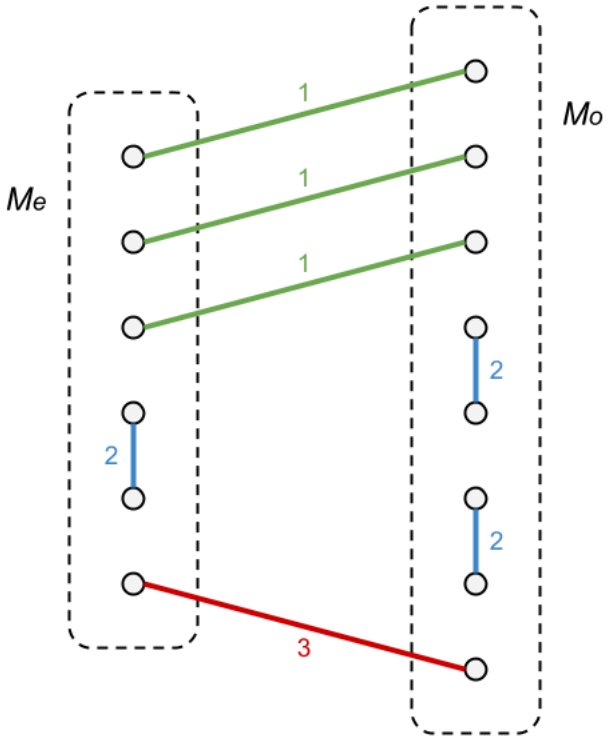
(Strictly speaking, this is related to distribution of primes, for example we have to prove that arbitrary even number can be a difference of two primes. We confirmed this up to the constraints of the problem experimentally.)

Let $y_1, \ldots, y_M$ be the initial positions of the tokens. We want to divide them into $M/2$ pairs, $(a_1, b_1), \ldots, (a_{M/2}, b_{M/2})$, and minimize the sum of $d(a_i, b_i)$.

Let M_e, M_o be the number of tokens initially at even, odd positions, respectively. If we make k ($k \leq M_e, M_o$) pairs of cost 1, the total cost will be:

$$k \times 1 + (\lfloor (M_e - k)/2 \rfloor + \lfloor (M_o - k)/2 \rfloor) \times 2 + ((M_e - k)\%2) \times 3$$

Since this is a monotonous function of k, we want to maximize k. This maximum k can be computed by a bipartite matching.


Me
Mo
1
1
1
2
2
2
3