AtCoder Grand Contest 006 解説

writer : sugim48

English editorial starts on page 8.

A : Prefix and Suffix

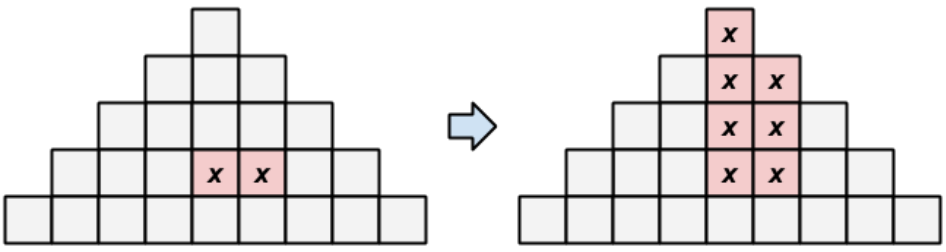
長さの上限は $2N$ です．なぜならば，s と t をそのまま連結した文字列は条件を満たすからです．よって，長さを N から $2N$ まで小さい順に試していって，条件を満たす文字列が見つかった時点でその長さを答えとすればよいです．

長さが l $(N \leq l \leq 2N)$ であるとき，条件を満たす文字列が存在するか，どう判定すればよいでしょうか？ 条件を満たす文字列は，s と t を $2N - l$ 分だけオーバーラップさせたような文字列になります．そのため，s の末尾 $2N - l$ 文字と，t の先頭 $2N - l$ 文字が，完全に一致しているか判定すればよいです．

B : Median Pyramid Easy

まず，$x = 1,\ 2N - 1$ の場合は `Impossible` です．これらは順列の最小値 / 最大値なので，下から 2 段目の時点で完全に消えてしまうからです．逆に，$2 \leq x \leq 2N - 2$ ならば必ず `Possible` です．以降，解を具体的に構成する方法を説明します．

$N = 2$ ならば，解は適当に $(1,\ 2,\ 3)$ などとすればよいです．以降は $N \geq 3$ とします．重要な観察として，左図の赤い 2 マスが x になれば，右図のように自然と頂上のマスも x になります．（このとき，他のマスの値は関係ありません．）



よって，左図の赤い 2 マスを x にするような順列を構成すればよいです．これは例えば

$$(\ldots,\ x-1,\ x,\ x+1,\ x-2,\ \ldots)$$

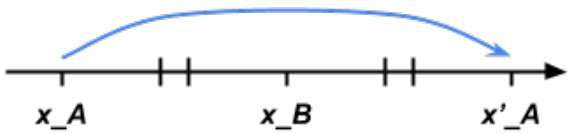
などとすればよいです．ただし，この構成は $x=2$ の場合のみ不適です．$x=2$ の場合は代わりに

$$(\ldots,\ x+1,\ x,\ x-1,\ x+2,\ \ldots)$$

などとすればよいです．

C : Rabbit Exercise

まず，「うさぎ A がうさぎ B に関して対称な座標へ移動する」という操作はどういうことかを考えます．うさぎ A, B の今の座標をそれぞれ x_A, x_B とおき，うさぎ A の新しい座標を x'_A とおきます．すると，x_A と x'_A の中間の値がちょうど x_B なので，$(x_A + x'_A)/2 = x_B$ です．この式を変形することで，$x'_A = 2x_B - x_A$ が得られます．「うさぎ A がうさぎ B に関して対称な座標へ移動する」という操作を数式で書くことができました．



確率論的な現象のままでは考えづらいので，決定論的な現象に言い換えます．今，うさぎ i が，うさぎ $i-1$ またはうさぎ $i+1$ を等確率で選び，選ばれたうさぎに関して対称な座標へ移動するとします．うさぎ $i-1, i, i+1$ の今の座標をそれぞれ x_{i-1}, x_i, x_{i+1} とおき，うさぎ i の新しい座標を x'_i とおきます．このとき，確率 $1/2$ で $x'_i = 2x_{i-1} - x_i$ となり，確率 $1/2$ で $x'_i = 2x_{i+1} - x_i$ となります．よって，x'_i の期待値 $\mathrm{E}[x'_i]$ は

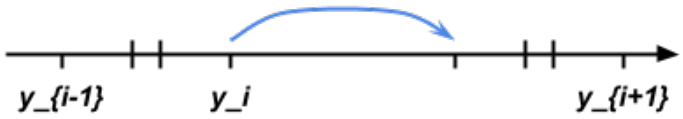
$$\begin{aligned}\mathrm{E}[x'_i] &= \frac{1}{2}\mathrm{E}[2x_{i-1} - x_i] + \frac{1}{2}\mathrm{E}[2x_{i+1} - x_i] \\ &= \frac{1}{2}(2\mathrm{E}[x_{i-1}] - \mathrm{E}[x_i]) + \frac{1}{2}(2\mathrm{E}[x_{i+1}] - \mathrm{E}[x_i]) \\ &= \mathrm{E}[x_{i-1}] + \mathrm{E}[x_{i+1}] - \mathrm{E}[x_i]\end{aligned}$$

と計算できます．よって，x_i の期待値を y_i とおくことで，問題は次のように言い換えられます．

> 数列 $y_1, y_2, \ldots, y_N$ がある．この数列に何回か操作を行う．1 セット分の操作列は，次のような合計 M 回の操作からなる．j 回目の操作では，$i := a_j$ とし，$y_i \leftarrow y_{i-1} + y_{i+1} - y_i$ と置き換える．以上の合計 M 回の操作を 1 セット分の操作列として，K セット分の操作列を続けて繰り返す．最終的な数列を求めよ．

決定論的な現象に言い換えることができました．

以上の考察により，$K = 1$ の場合は愚直にシミュレーションすることで解くことができます．K が大きい場合はどう解けばよいでしょうか？ まず思いつくのは，1 セット分の操作列によって数列がどう変わるかを行列の形で表し，行列累乗をするという方法です．しかし，この方法の計算量は $\mathrm{O}(N^3 \log K)$ と大きすぎます．ここで，$y_i \leftarrow y_{i-1} + y_{i+1} - y_i$ という操作をよく観察すると，$y_i - y_{i-1}$ の値と $y_{i+1} - y_i$ の値が交換されていることに気づきます．



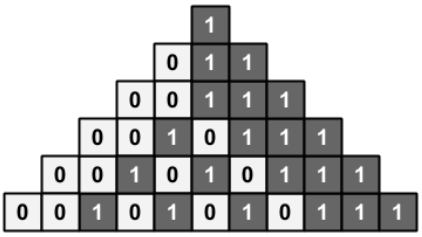
よって，$\mathrm{d}y_i := y_{i+1} - y_i$ と長さ $N-1$ の数列を定義することで，y_i に対する操作を，$\mathrm{d}y_i$ に対する隣接要素のスワップと捉え直すことができます．すると，1 セット分の操作列の結果は $\mathrm{d}y_i$ に対する置換として表すことができます．長さ N の置換どうしの積は $\mathrm{O}(N)$ で計算できるので，行列累乗の要領で置換の累乗をすると，計算量は $\mathrm{O}(N \log K)$ となります．これは十分高速です．最終的な数列 $\mathrm{d}y_i$ が求まれば，その累積和をとることで数列 y_i が復元できます．

D : Median Pyramid Hard

愚直にすべてのマスを埋めていく方法だと，$\mathrm{O}(N^2)$ 時間掛かって間に合いません．そこで，頂上のマスの値 x を二分探索することを考えます．具体的には，「頂上のマスの値が x 以上か」という条件を用い，二分探索することを考えます．

「頂上のマスの値が x 以上か」という判定問題を高速に解くにはどうすればよいでしょうか？「頂上のマスの値が x 以上か」ということだけが気になっているので，最下段の順列の各値を，x 以上ならば 1，x より小さいならば 0，と表してしまいましょう．さらに，これから埋めていくマスの値についても，同様に 0/1 で表してしまいましょう．このように各値を 0/1 へ表してしまっても，マスを埋めていくルールはまったく同じものが適用できることが分かります．具体的には，「あるマスの値は，そのマスの左下，真下，右下の 3 マスの値の多数決で決まる」というルールになります．この単純化された問題設定ならば，頂上のマスの値を高速に計算できそうです．以降，具体的な計算方法を説明します．

まず，すぐに分かるのは，0 または 1 が複数マス連続している区間については，その真上の区間は変化しないということです．よって，頂上のマスがこのような区間に含まれていれば，頂上のマスの値は簡単に求まります．では逆に，0/1 が交互に並んでいる区間については，その真上の区間はどう変化していくでしょうか？ たとえば，図のように変化していきます．



つまり，0/1 が交互に並んでいる区間については，0/1 が反転しながら，幅が左右 1 マスずつ狭くなっていきます．一方，左右の連続区間については，幅が 1 マスずつ広くなっていきます．そして，左右の連続区間がぶつかったところで，変化が止まります．この挙動をよく観察することで，頂上のマスの値を計算できます．ただし，最下段全体で 0/1 が交互に並んでいる場合に注意してください．

以上の方法を用いると，頂上のマスの値を $\mathrm{O}(N)$ 時間で計算できます．この値が 0 か 1 かで二分探索の分岐をすることで，頂上のマスの真の値が求まります．全体の計算時間は $\mathrm{O}(N \log N)$ で，十分に間に合います．

E : Rotate 3x3

まずは，Rotate 2x2 という問題を考えてみましょう．つまり，マス目の縦幅が 2 であり，正方形の大きさが 2×2 であるような，簡単な問題設定です．2×2 の正方形内の配置を 180° 回転するとはどういうことか考えてみましょう．これは，隣り合う 2 列の位置をスワップし，同時に各列を上下反転する操作と見ることができます．操作を行うたびに列が上下反転されると考えづらいので，次のように見方を工夫することにします．マス目のうち偶数列目の位置にある列は，常に上下が反転して表示されるとします．すると，先述の操作は，隣り合う 2 列の位置をスワップするだけの操作に見えることになります．よって，初期配置と目標配置のそれぞれについて偶数列目を上下反転しておけば，あとはこれらの配置がスワップ操作のみによって移り合えるか判定するだけの問題になります．これは結局，初期配置と目標配置で列の集合が一致しているか判定すればよいです．以上より，Rotate 2x2 が解けました．

さて，元の問題である Rotate 3x3 に戻りましょう．まずは，3×3 の正方形内の配置を 180° 回転するとはどういうことか考えてみましょう．これは，左の列と右の列の位置をスワップし，同時に各列を上下反転した後，さらに真ん中の列も上下反転する操作と見ることができます．ここでも，Rotate 2x2 と同様に見方を工夫することで，問題設定を簡単にします．つまり，3, 4, 7, 8, 11, 12, . . . 列目の位置にある列は，常に上下が反転して表示されるとします．すると，先述の操作は，左の列と右の列の位置をスワップした後，真ん中の列を上下反転する操作に見えることになります．また，初期配置と目標配置のそれぞれについて，3, 4, 7, 8, 11, 12, . . . 列目を上下反転しておきます．このように簡単化された問題設定において，解法を考えていきます．

「左の列と右の列の位置をスワップした後，真ん中の列を上下反転する」操作では，奇数列目ど

うし，または，偶数列目どうしがスワップされます．よって，奇数列目の並べ替えと，偶数列目の並べ替えは，ある程度独立に考えることができます．ここでは，奇数列目の並べ替えに注目してみましょう．とりあえず，「隣り合う列のスワップ」と「ある列の上下反転」の 2 種類の操作が好きな順番で行えるとしたとき，初期配置（の奇数列目）から目標配置（の奇数列目）へ変形できないならば，答えは `Impossible` とすぐ分かります．変形できるならば，「隣り合う列のスワップ」の最小回数と，「ある列の上下反転」の最小回数を計算し，それぞれ inv_o, $flip_o$ としておきます．同様に，偶数列目についても，変形できるかどうかを判定し，inv_e, $flip_e$ を計算しておきます．実は，inv_o, $flip_o$, inv_e, $flip_e$ の偶奇をチェックすることで，`Possible` か `Impossible` かを判定できます．

まず，inv_o, $flip_o$, inv_e, $flip_e$ の偶奇に関する必要条件を考えてみましょう．「左の列と右の列の位置をスワップした後，真ん中の列を上下反転する」操作を行うと，inv_o と $flip_e$ の偶奇がともに変化するか，または，inv_e と $flip_o$ の偶奇がともに変化します．よって，inv_o と $flip_e$ の偶奇は一致している必要があり，かつ，inv_e と $flip_o$ の偶奇は一致している必要があります．実は，$N \geq 5$ ならば，これらは十分条件になっていることが示せます（証明は後述）．以上より，Rotate 3x3 が解けました．転倒数を計算するパートが最も重く，全体の計算量は $\mathrm{O}(N \log N)$ です．

(十分性の証明) inv_o と $flip_e$ の偶奇が一致していて，かつ，inv_e と $flip_o$ の偶奇が一致していれば，`Possible` であることを示します．「左の列と右の列の位置をスワップした後，真ん中の列を上下反転する」操作をうまく組み合わせることで，奇数列目から任意の 2 列，または，偶数列目から任意の 2 列を選び，それらをともに上下反転できる，ということが示せれば十分です．これを示すためには，距離がちょうど 2 だけ離れた任意の 2 列を選び，それらをともに上下反転できる，ということが示せれば十分だと分かります．以降，これを示します．

マス目のうち，連続する 5 列を好きな位置から取り出してきます．この 5 列を `a b c d e` と表すことにします．さらに，上下反転した列を `A` のように大文字で表すことにします．次のような操作列を構成してみます．

- `a b c d e`
- `c B a d e`
- `c B e D a`
- `e b c D a`
- `e b a d c`
- `a B e d c`
- `a B c D e` (1)
- `a D C B e`
- `C d a B e`
- `C B A d e`
- `A b C d e` (2)

(1) および (2) を見ると，距離がちょうど 2 だけ離れた 2 列をともに上下反転できていることが分かります．連続する 5 列は好きな位置から取り出せるので，距離がちょうど 2 だけ離れた任意の 2 列を選べることになります．以上より，十分性が証明できました．

F : Blackout

とりあえず，次のように問題を言い換えます．

> N 頂点 M 辺の有向グラフがある．辺 $(x, y), (y, z)$ が存在し，辺 (z, x) が存在しないならば，辺 (z, x) を張ることができる．可能な限り辺を張ったとき，最終的な辺の本数を求めよ．

明らかに，グラフの弱連結成分（有向辺を無向辺と見たときの連結成分）どうしは独立に計算することができ，答えはそれらの総和になります．よって，以降はグラフ全体が弱連結であると仮定します．

突然ですが，N 個の頂点に A, B, C のどれかのラベルを付けて，次の条件が満たされるようにしましょう．

- M 本の辺のそれぞれについて，その始点と終点のラベルは (A → B) または (B → C) または (C → A) のようになっている．

このようなラベル付けは一意ではないですし，そもそもどうラベル付けしても矛盾してしまう場合もあります．しかし，DFS などをすることで，「ラベル付けをひとつ求める」か「矛盾を検出する」ことができます．

ラベル付けの結果で場合分けをすることで，最終的な辺の張られ方が次のように求まります．証明は後述します．

- A, B, C すべてのラベルが使われていない場合
  新たに辺を張ることができないので，最初の辺集合のままである．
- A, B, C すべてのラベルが使われている場合
  (A → B) または (B → C) または (C → A) のような辺はすべて張ることができる．それ以外の辺は張ることができない．
- どうラベル付けしても矛盾してしまう場合
  任意の頂点から任意の頂点へ辺を張ることができる．特に，任意の頂点から自己ループを張ることができる．

どの場合においても，最終的な辺の本数は簡単に計算できます．よって，答えを求めることができました．計算量は $O(N + M)$ です．

（A, B, C すべてのラベルが使われていない場合の略証）この場合は，辺 $(x, y), (y, z)$ の組が存

在しないので，新しく辺を張ることができません．□

（A, B, C すべてのラベルが使われている場合の略証）この場合は，辺 (x,y), (y,z) の組がどこかに存在するはずなので，辺 (z,x) を新しく張って，頂点 x, y, z からなる三角形を作っておきます．今，x, y, z のラベルがそれぞれ A, B, C であるとしても，一般性を失いません．最初の目標は，「任意の頂点 v について，v が x, y, z のうち 2 個と辺で結ばれている」ようにすることです．たとえば，v のラベルが A ならば，辺 (v,y), (z,v) が存在するようにします．これは，三角形 x, y, z に近い頂点から順に，所望の辺を張っていくことで達成できます．「任意の頂点 v について，v が x, y, z のうち 2 個と辺で結ばれている」ようにできれば，その繋がりを利用して，(A $\to$ B) または (B $\to$ C) または (C $\to$ A) のような辺をすべて張ることができます．逆に，それ以外の辺は張れないこともすぐに分かります．□

（どうラベル付けしても矛盾してしまう場合の略証）目標は，ある頂点について自己ループを張ることです．これができれば，ひとつ前の略証と同様にして，任意の頂点から任意の頂点へ辺を張ることができます．まず，どうラベル付けしても矛盾するようなサイクルをひとつ見つけてきます．（このサイクルは必ずしも有向閉路である必要はなく，無向辺として見たときにサイクルであればよいです．）このサイクルには必ず辺 (x,y), (y,z) の組がどこかに含まれるので，辺 (z,x) を新しく張り，サイクルのサイズをひとつだけ小さくします．このとき，ひとつだけ小さくなったサイクルも，どうラベル付けしても矛盾することが分かります．この操作を繰り返していくことにより，サイクルのサイズを 1 まで小さくすることができます．これは欲しかった自己ループそのものです．□

AtCoder Grand Contest 006 Editorial

writer : sugim48

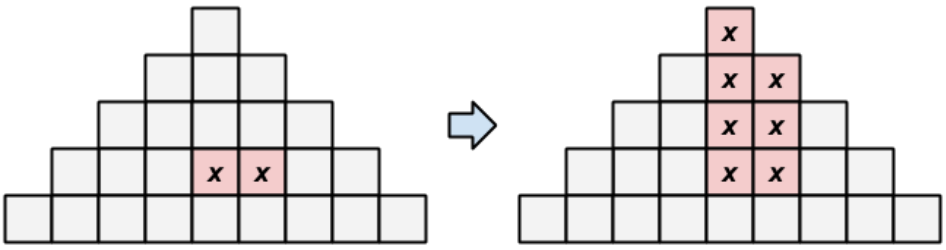
A : Prefix and Suffix

The answer is always at most $2N$: the concatenation of s and t satisfies the conditions. Thus, for each l ($N \leq l \leq 2N$), we want to check whether there exists a string of length l that satisfies the conditions. Such strings exist if and only if the last $2N - l$ characters of s and the first $2N - l$ characters of t are the same.

B : Median Pyramid Easy

In case $x = 1,\ 2N-1$, the answer is `Impossible`. This is because since 1 and $2N-1$ are the max/min of the permutation, these numbers can't appear in the second row from the bottom. Otherwise, the answer is always `Possible`, as explicitly constructed as follows:

Suppose that $N \geq 3$ (if $N = 2$, the answer is $(1,\ 2,\ 3)$ for example). An important observation is that, if the two red cells in the left picture are x, the topmost cell will also become x as the right picture shows.



Thus, it is sufficient to come up with a permutation that makes the two red cells x. For example, when $x \neq 2$, the permutation can be

$$(\ldots,\ x-1,\ x,\ x+1,\ x-2,\ \ldots)$$

and when $x = 2$,

$$(\ldots,\ x+1,\ x,\ x-1,\ x+2,\ \ldots)$$

C : Rabbit Exercise

Everything about this task is to notice a very simple, but yet hard-to-notice trick. If you notice the trick, the task looks like an *A + B problem* and you can solve the task immediately, while if you don't the task looks like *the hardest task you've ever seen* and you will never solve the task forever in your life!

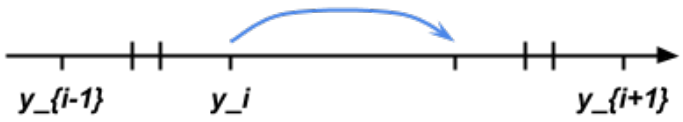
When an operation is performed on the rabbit i, the expected position of this rabbit after the operation is:

$$\frac{1}{2}(2x_{i-1} - x_i) + \frac{1}{2}(2x_{i+1} - x_i) = x_{i-1} + x_{i+1} - x_i$$

Thus, an operation on the rabbit i is equivalent to:

$$x_i := x_{i-1} + x_{i+1} - x_i$$

What does this operation mean? Let's draw a diagram that shows the positions of $y_i, y_{i-1}, y_{i+1}, y_{i-1} + y_{i+1} - y_i$:



Here the trick comes. The operation just swaps (the distance between the rabbits i and $i+1$) and (the distance between the rabbits i and $i-1$)!

Now everything becomes easy. Let $\mathrm{d}x_i := x_{i+1} - x_i$, and an operation on the rabbit i means

$$swap(\mathrm{d}x_{i-1}, \mathrm{d}x_i);$$

Thus, the problem can be solved in $\mathrm{O}(N \log K)$ time by computing an exponentiation of a permutation.

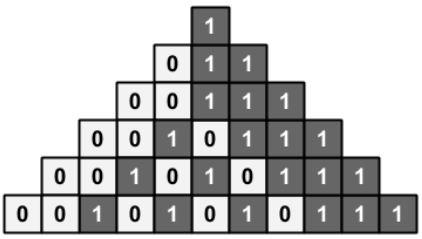
D : Median Pyramid Hard

The straightforward simulation is $\mathrm{O}(N^2)$. It may be surprising, but we can use binary search to achieve a better complexity. For a given value x, let's try to decide whether the value on the top is at least x or not. If we can do this, we can compute the value on the top by repeating $\mathrm{O}(logN)$ questions of this type by using binary search.

In order to answer this question, we don't need to distinguish integers less than x, and we don't need to distinguish integers greater than or equal to x. The only important thing is that the integers in the cells are at least x or not. Thus, let's replace all integers less than x with 0, and the other integers with 1. The rule for computing the median is exactly the same: when we decide a value of a cell, we check three integers (directly or diagonally) under the current cell, and fill it with the mode (the most frequent value) among the three integers. This simplifies the simulation. Indeed, when the values are only 0 and 1, we can compute the value on the top in linear time, as described below.

Suppose that two zeroes are horizontally adjacent in the bottom row. In this case, you can see that all values in these two columns will be zero. The same happens for horizontally adjacent ones.

What happens when 0 and 1 appear alternately? See the following diagram:



When there are two horizontally adjacent same numbers, this number also appears on top of it, and the "width" of the range of this number increases as we go up, and makes a V-shape. When two shapes of this type meet, they stop in the middle. In other areas, the value of a cell is the negation of the value directly below it. You can compute the value on the top by using these properties. (Make sure not to forget the case where the bottom row entirely alternates).

This way, when the values are only 0 or 1, we can compute the value on the top in $\mathrm{O}(N)$ time. With binary search, we can solve the original problem in $\mathrm{O}(N \log N)$ time in total.

E : Rotate 3x3

What happens when you rotate a 3×3 subrectangle by 180 degrees? This is equivalent to the following operation: swap the left column and the right column, and then flip all three columns.

Thus, even if you repeat multiple operations, the set of columns won't change. For each k, the three numbers $3k+1, 3k+2, 3k+3$ will be always in the same column. However, the order of these numbers may be reversed (here "reverse" means "upside down"). In this column, the numbers are $3k+1, 3k+2, 3k+3$ or $3k+3, 3k+2, 3k+1$ from top to bottom.

Also, you can notice that odd-indexed columns are always swapped with odd-indexed

columns. For each integer l, the numbers $6l+1, 6l+2, 6l+3$ will be always in an odd-indexed column. The same holds for even-indexed columns: the numbers $6l+4, 6l+5, 6l+6$ will be always in an even-indexed column.

It is clear that these conditions are necessary - but are these sufficient? To check this, the easiest way is to count the number of reachable configurations. For example, when $N = 10$, you can check that there are 3686400 reachable configurations by writing a brute force program. The conditions above will give $120 \times 120 \times 2^{10} = 14745600$ configurations. This is 4 times bigger than the actual number and the conditions are not sufficient - and it suggests that there are two types of "parities".

Let's think about odd-indexed columns. Consider two operations: "swap two consecutive (that is, two columns with distance 2) odd-indexed columns" and "reverse an odd-indexed column". Clearly, if you can't reach the final configuration from the initial configuration by using these two types of operations, the answer is `Impossible`. If you can, let inv_o be the number of operations of the former type, and $flip_o$ be the number of operations of the latter type. The values of inv_o and $flip_o$ can't be uniquely determined, but their parities are unique. Similarly, compute inv_e and $flip_e$ for even-indexed columns.

We want to come up with the necessary conditions for inv_o, $flip_o$, inv_e, and $flip_e$. If you perform the operation "swap the left column and the right column, and then flip all three columns", one of the following two things will happen:

- The parities of inv_o and $flip_e$ change.
- The parities of inv_e and $flip_o$ change.

Thus, the parities of inv_o and $flip_e$ must be the same, and the parities of inv_e and $flip_o$ must be the same. Now we found two "parities". We can prove that, when $N \geq 5$, these conditions are indeed sufficient (proved later). Finally we solved the problem. The time complexity is $O(N)$ in total.

(Proof) Now, we'll give a proof that the conditions above are indeed sufficient. That is, we want to prove that whenever the parities of inv_o and $flip_e$ are the same, and the parities of inv_e and $flip_o$ are the same, the answer is `Possible`. It is sufficient to prove that, by repeating the operation "swap the left column and the right column, and then flip all three columns", you can reverse arbitrary two odd-indexed (or even-indexed) columns. Furthermore, it is sufficient to prove that you can choose two rows with distance 2, and reverse both columns.

Let `a b c d e` be consecutive 5 columns. We'll denote reversed columns as an uppercase letter, for example `A` is a reversed `a`.

Consider the following sequence of operations:

- `a b c d e`

- c B a d e
- c B e D a
- e b c D a
- e b a d c
- a B e d c
- a B c D e (1)
- a D C B e
- C d a B e
- C B A d e
- A b C d e (2)

- a b c d e
- C B A d e
- C B E D a
- e b c D a
- e b A d C
- a B E d C
- a B c D e (1)
- a d C b e
- c D A b e
- c B a d e
- A b C d e (2)

If we see (1) and (2), we can see that we reversed two rows with distance 2. Therefore, we proved that the conditions we gave are sufficient.

F : Blackout

Let's state the problem in terms of graph theory.

> You are given a graph with N vertices and M edges. When there are two edges $x \to y$ and $y \to z$, you can add an edge $z \to x$. You keep adding edges while you can. How many edges will be there at the end?

Obviously, we can compute the answer for each weakly-connected component (that is, the connected component when we ignore directions), and the answer is the sum of these numbers. We'll assume that the graph is (weakly) connected.

Let's do some experiments. Consider a path graph. That is, the vertices are labeled with

integers, and there is an edge from vertex i to $i+1$ for each integer i. If you repeat operations on this graph, you can see that there will be an edge from s to t if and only if $t \equiv s + 1($ (mod 3)). It suggests that the problem has something to do with the length of path modulo 3.

Let's try to label the N vertices with labels `A`, `B`, `C` such that:

- For each of the M edges, the labels of the endpoints of the edge is (`A` $\to$ `B`) or (`B` $\to$ `C`) or (`C` $\to$ `A`).

Such labelling may not exist, but when it exists the labelling is unique (except for cyclic shift). By DFS, you can construct such an labelling (or conclude that it doesn't exist).

There are three cases depending on the result of labelling. We'll first state the conclusion, and later we'll give the proof.

- When the labelling is possible, and not all of `A`, `B`, `C` are used
  You can't perform any operations and the set of edges remain unchanged.
- When the labelling is possible, and all of `A`, `B`, `C` are used
  You can add an edge between all pairs with labelling (`A` $\to$ `B`), (`B` $\to$ `C`), and (`C` $\to$ `A`).
  You can't add any other edges.
- When the labelling is impossible
  You can add an edge between any pair of vertices, including self loops.

In all cases, we can easily compute the number of edges in the final configuration. The total complexity of this task is $O(M)$.

Now we'll give a proof. In the proof we'll denote the edge $x \to y$ as (x, y).

When the labelling is possible, and not all of `A`, `B`, `C` are used If two edges (x, y) and (y, z) exist, all labels will be used and this is a contradiction. Thus, in this case there is no such pairs of edges, and you can't perform any operations. □

When the labelling is possible, and all of `A`, `B`, `C` are used In this case, there are some x, y, z such that both edges (x, y), (y, z) exist. We can add an edge (z, x), and these edges from a triangle. Let's assume that the labels of x, y, z are `A`, `B`, `C`, respectively. Consider a vertex v that is adjacent to one of x, y, or z. For example, when there is an edge (v, x), a triangle with v, x, y will be formed. Similarly, we can prove that v is always adjacent to two of x, y, z. Then, consider another vertex w that is adjacent to one of v, x, y, z, and prove that w is always adjacent to two of x, y, z. By repeating this process, we can prove that each vertex is adjacent to two of x, y, z, and by using this we can add all edges between (`A` $\to$ `B`), (`B` $\to$ `C`), and(`C` $\to$ `A`). On the other hand, it is clear that we can't add other edges. □

When the labelling is impossible Let's prove that at least one self-loop will be formed. If we prove this, the remaining part of the proof is almost equivalent to the former case.

First, find a cycle of the graph that will cause a contradiction in modulo 3. (This cycle is not necessarily a directed cycle: this is an undirected cycle in general.) This cycle must contain two edges (x, y) and (y, z), and thus we can make the size of the cycle smaller by adding an edge (z, x). This smaller cycle will also cause a contradiction in modulo 3. By repeating this process, we will get a self loop. $\square$