AGC 008 Editorial

writer : sugim48

2016 年 12 月 25 日

For International Readers: English editorial starts on page 8.

A : Simple Calculator

最短の操作列は

(B を 0 or 1 回押す) → (A を 0 回以上押す) → (B を 0 or 1 回押す)

の形に限ることが示せます (証明は後述).

そこで,「最初に B を押す / 押さない」と「最後に B を押す / 押さない」の組合せを計 4 通り試すことにします．最初に B を押す場合，x の符号を反転しておきます．また，最後に B を押す場合，y の符号を反転しておきます．すると，残りの操作は「A を 0 回以上押す」だけです．よって，$x \leq y$ が成り立っていれば，A を $y - x$ 回押すことで目標を達成できます．このようにして求めた操作回数のうち最小値が答えです．

(証明) 操作列中の “BB” を “” へ，“ABA” を “B” へ，それぞれ置き換えても結果は変わらず，また操作列はより短くなります．よって，最短の操作列に “BB” や “ABA” は含まれません．よって，最短の操作列に “B” が含まれるならば，それは先頭か末尾に限ります．

B : Contiguous Repainting

この問題のように何かを上書きしていく設定に対しては，操作列を逆順に見るという考え方が有効です．元々の設定は次のようでした．

> 最初，すべてのマスは白色である．「長さ K の区間と，白色または黒色を選び，区間に含まれるすべてのマスを選んだ色で上書きする」という操作を繰り返す．

操作列を逆順に見ると，設定は次のように変わります．

> 最初，すべてのマスは未確定である．「長さ K の区間と，白色または黒色を選び，区間に含まれるすべての未確定のマスを選んだ色に確定させる」という操作を繰り返す．色が確定したマスはその後上書きされない．

一般に，後者の設定の方が扱いやすいことが多いです．以降は後者の設定のもとで考えます．

とりあえず，(操作列を逆順に見たとき) 最初の操作によって，どこかの長さ K の区間がすべて白色かすべて黒色に確定します．実は，この区間以外のすべてのマスは，1 個ずつ自由に色を確定させることができます．たとえば，最初の操作で選んだ区間のすぐ右隣のマスの色を確定させたければ，次の操作で選ぶ区間を 1 マスだけ右にずらせばよいです．同様にして，残りのマスも 1 個ずつ色を確定させていけばよいです．

以上より，次のような方法で解くことができます．まず，長さ K の区間を全探索し，その区間をすべて白色にするかすべて黒色にするかを両方試します．そして，その区間以外の各マス i について，$a_i \geq 0$ ならば黒色にし，$a_i < 0$ ならば白色にします．このようにして求まるスコアのうち最大値が答えです．

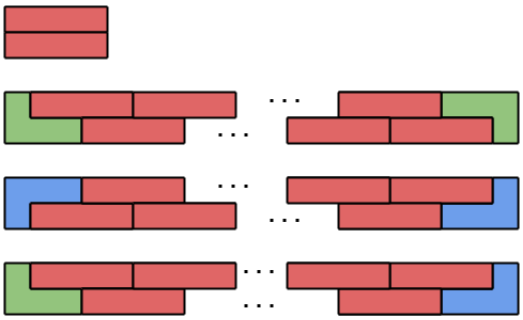
この方法を愚直に行うと，時間計算量が $\mathrm{O}(N^2)$ で間に合いません．そこで，前計算として，a_i の累積和と $b_i := \max\{0, a_i\}$ の累積和をそれぞれ計算しておきます．すると，時間計算量が $\mathrm{O}(N)$ となって間に合います．

C : Tetromino Tiling

まず，T, S, Z 型を使うことはできません．これらのテトロミノが長方形に含まれるとすると，そのテトロミノによって長方形は左右に分断されます．このとき，分断された左右の領域はともに奇数個のマスからなります．テトロミノはどれも 4 マスからなるので，どうしても奇数個のマスからなる領域を埋めることはできません．よって，T, S, Z 型を使うことはできません．

次に，O 型はすべて使うことができます．余っている O 型があれば，それを長方形の右端に追加することは常に可能です．また，O 型はすべて長方形の右端に寄せることができます．長方形の中間に O 型が挟まれていれば，それを抜いて長方形の右端に追加することは常に可能です．よって，I, J, L 型だけで長方形を作るときの最大個数と，a_O の和が答えとなります．

以上より，I, J, L 型だけで長方形を作るときの最大個数を求めればよいことになりました．I, J, L 型だけで作る長方形は，次の 4 種類のパーツを左右に連結したものに限られます．



後半 3 種類のパーツにおいて，I 型の連続部分をできるだけ I + I 型のパーツへ分解すると，ありうるパーツは次の 4 種類に限定されます．

ここで，I + J + L 型のパーツは高々 1 個しか使う必要はありません．なぜなら，I + J + L 型のパーツ 2 個を組み直すと，I + I 型のパーツと J + J 型のパーツと L + L 型のパーツへ分解できるからです．また，I + J + L 型のパーツを 0 個使うか 1 個使うかを決めると，後は I + I 型のパーツと J + J 型のパーツと L + L 型のパーツを貪欲に作るだけになります．よって，I + J + L 型のパーツを 0 個使うか 1 個使うかを両方試して最適な方を選ぶことで，I, J, L 型だけで長方形を作るときの最大個数が求まります．

D : K-th K

まず，数列 a の x_1, x_2, ..., x_N 番目には，それぞれ 1, 2, ..., N を置かなければなりません．残りの場所に整数を置くとき，どのような条件が成り立つようにすればよいでしょうか？ これは，各 $1 \leq i \leq N$ について，数列 a の x_i 番目より左にちょうど $i-1$ 個の整数 i が置かれており，数列 a の x_i 番目より右にちょうど $N-i$ 個の整数 i が置かれている，という条件になります．この条件が成り立つように，残りの場所に整数を置いていきましょう．

まずは，「各 $1 \leq i \leq N$ について，数列 a の x_i 番目より左にちょうど $i-1$ 個の整数 i が置かれている」という条件を成り立たせるように整数を置いていきましょう．このために置く整数はできるだけ左側に詰めて置くのがよいことが分かります．ここでは，左端から順に 1 個ずつ整数を置いていくことにします．このとき，最初に置いていく整数は，x_i が最も小さいような整数 i_1 にするのが最適であることが分かります．整数 i_1 をちょうど i_1-1 個置き終えたら，次に置いていく整数は，x_i が 2 番目に小さいような整数 i_2 にするのが最適であることが分かります．このような順番で，各 $1 \leq i \leq N$ についてちょうど $i-1$ 個ずつ整数 i を置いていきます．

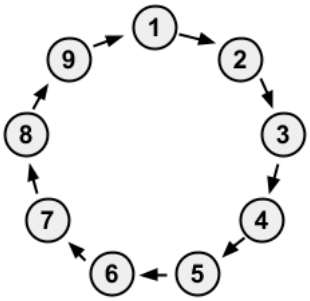
次に，「各 $1 \leq i \leq N$ について，数列 a の x_i 番目より右にちょうど $N-i$ 個の整数 i が置かれている」という条件を成り立たせるように整数を置いていきましょう．このために置く整数はできるだけ右側に詰めて置くのがよいことが分かります．ここでは，右端から順に 1 個ずつ整数を置いていくことにします．このとき，最初に置いていく整数は，x_i が最も大きいような整数 i_1 にするのが最適であることが分かります．整数 i_1 をちょうど $N-i_1$ 個置き終えたら，次に置いていく整数は，x_i が 2 番目に大きいような整数 $N-i_2$ にするのが最適であることが分かります．このような順番で，各 $1 \leq i \leq N$ についてちょうど $N-i$ 個ずつ整数 i を置いていきます．

以上のようにして，最適な方法ですべての場所に整数を置くことができました．最後に，成り立つべき条件が実際に成り立っているかチェックします．成り立っていれば，答えは `Yes` であり，構成した数列 a を出力すればよいです．成り立っていなければ，他にどのようにしても条件が成り立つようにできないので，答えは `No` です．

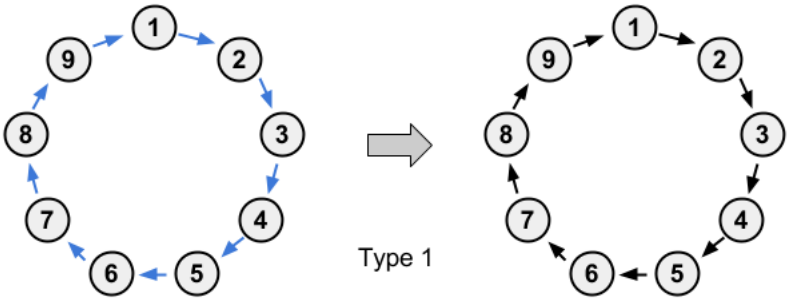
E : Next or Nextnext

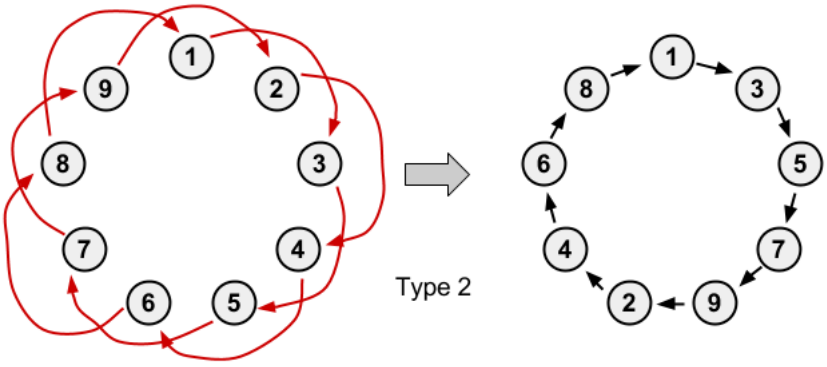
$(1, 2, ..., N)$ の順列 $(p_1, p_2, ..., p_N)$ が与えられたとき，頂点 i から頂点 p_i へそれぞれ有向辺を張ることで，有向グラフを作ることができます．同様にして，数列 $(a_1, a_2, ..., a_N)$ からも有向グラフを作ることができます．a_i の値として p_i または p_{p_i} が選べるとき，数列 $(a_1, a_2, ..., a_N)$ のグラフはどのような形がありうるでしょうか？

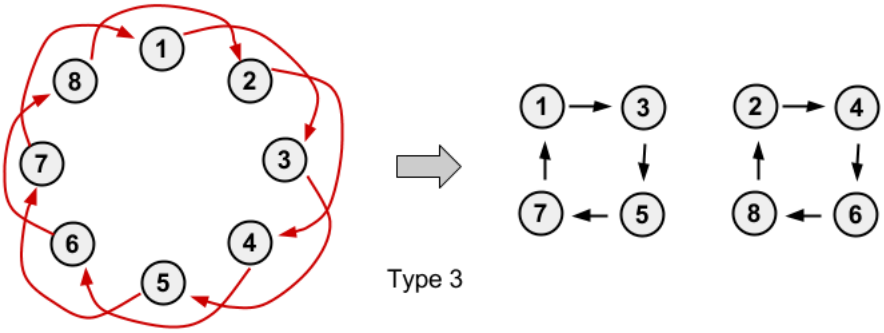
まず，順列 $(p_1, p_2, ..., p_N)$ のグラフはいくつかの閉路からなることが分かります．それら閉路のうちひとつに注目して考えることにします (次図)．

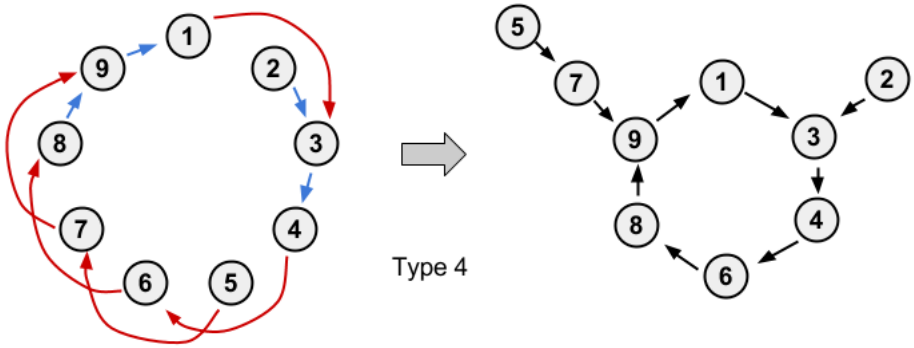


この閉路において，a_i の値として p_i または p_{p_i} が選べるとき，数列 $(a_1, a_2, ..., a_N)$ のグラフの形は次図の 4 種類に大別できます．なお，$a_i = p_i$ を青い矢印で，$a_i = p_{p_i}$ を赤い矢印で表しています．







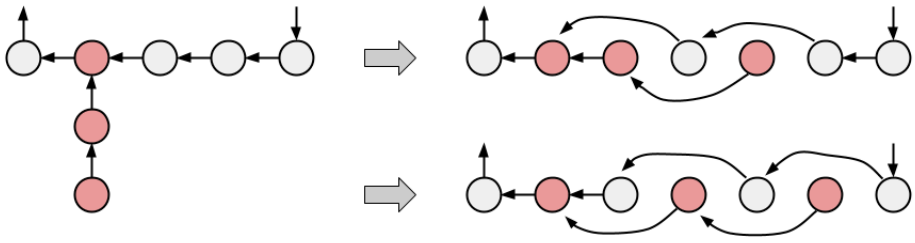


Type 1 は，前と後でグラフの形がまったく同じです．Type 2 は，閉路のサイズが 3 以上の奇数の場合のみ発生し，前と後で頂点の順番のみが変わっています．Type 3 は，閉路のサイズが偶数の場合のみ発生し，サイズが半分の閉路 2 個に分裂します．Type 4 は，閉路に何本か足が生えた形になっています．このとき，足に分岐はなく，また同じ箇所から複数本の足が生えることはありません．

以上の観察をもとに，元の数え上げ問題の解法を考えていきます．まず，与えられた数列 $(a_1, a_2, ..., a_N)$ から先述のグラフを作ります．このグラフはいくつかの連結成分に分かれていますが，各連結成分はそれぞれ Type 1 - 4 のどれかによって発生したはずです．よって，もし足に分岐があったり，同じ箇所から複数本の足が生えていたりしたら，すぐに答えは 0 と分かります．以降は，各連結成分は「ただの閉路」または「閉路に適切な足が何本か生えたもの」であるとします．これらの元々の姿としてありうるグラフを数え上げます．

まずは，「ただの閉路」たちの元々の姿としてありうるグラフを数え上げます．同じサイズの閉路が 2 個ある場合，それらは元々 1 個の閉路だったものが Type 3 によって分裂した可能性があります．よって，「ただの閉路」たちは独立に元々の姿を数え上げることはできず，閉路のサイズ k ごとにその個数 n_k をカウントしておき，まとめて元々の姿を数え上げる必要があります．これは $O(n_k)$ の DP で可能です．DP の状態としては，未処理の閉路の個数を持ちます．未処理の閉路のうち適当にひとつ選んできたときに，その閉路は Type 1 によって発生したか，k が 3 以上の奇数ならば Type 2 によって発生したか，または他の未処理の閉路とペアになって Type 3 によって発生したか，のどれかです．これらを DP の遷移とすればよいです．なお，Type 3 の場合に閉路 2 個をペアにするとき，閉路 2 個の噛み合わせ方は k 通りあることに注意してください．

次に，「閉路に足が生えたもの」たちの元々の姿としてありうるグラフを数え上げます．「閉路に足が生えたもの」たちは Type 4 のみによって発生したので，他のものとペアになって発生したということはありえません．よって，「閉路に足が生えたもの」たちは独立に元々の姿を数え上げることができます．「閉路に足が生えたもの」は元々ただの閉路だったので，足の部分をうまく閉路の部分に収納してあげる必要があります．このとき，それぞれの足の収納方法は次図の 2 通りです．



実際には，足の収納スペースがどれくらいあるかで場合分けが必要です．今注目している足の辺数を l_1，今注目している足の付け根から次の足の付け根までの辺数を l_2 とします．$l_1 < l_2$ ならば，前図の 2 通りの収納方法が可能です．$l_1 = l_2$ ならば，前図のうち上の 1 通りの収納方法のみが可能です．$l_1 > l_2$ ならば，この足は収納できず，すぐに答えは 0 と分かります．以上の場合分けをすべての足について行えばよいです．

以上の解法をまとめます．まず，与えられた数列 $(a_1, a_2, ..., a_N)$ から有向グラフを作り，連結成分ごとに形をチェックします．このとき，足の付け根や途中に分岐があれば，すぐに答えは 0 と分かります．そうでなければ，「ただの閉路」たちと「閉路に足が生えたもの」たちのみが存在することになります．「ただの閉路」たちの元々の姿の通り数は，DP によって求まります．「閉路に足が生えたもの」たちの元々の姿の通り数は，足の収納方法の通り数の積で求まります．これらの積が答えとなります．計算量は $\mathrm{O}(N)$ です．

F : Black Radius

まず思いつくのは，お気に入りの頂点 v を全探索するという方法です．v を固定して距離 d を自由に動かすとき，何通りの配色がありうるかというのは簡単に求められます．しかし，これらの通り数をすべて合計しても正しい答えは求まりません．なぜなら，異なる v から同一の配色が作られる場合に，それらを重複して数えてしまうからです．重複が発生しないようにするためには，何を全探索すればよいでしょうか？

どのような配色でも，黒い頂点はひと繋がりの部分木をなすことがすぐに分かります．この黒い部分木の center vertex および center edge を全探索することを考えます．center vertex および center edge が異なれば当然黒い部分木も異なるため，重複は発生しません．以降は，center vertex および center edge を固定したときに何通りの黒い部分木がありうるかを考えることにします．

まずは，center vertex を頂点 v に固定したときに何通りの黒い部分木があるかを考えます (このとき，必ずしも v はお気に入りの頂点とは限らないことに注意してください)．center vertex が v である黒い部分木の半径 r をひとつ決めたとします．すると，当然 v からの距離が r より大きい頂点は，すべて白色でなければなりません．さらに，実は v からの距離が r 以下の頂点は，すべて黒色でなければならないということも分かります．よって，半径 r をひとつ決めると，対応する黒い部分木は (存在するならば) 一意に定まります．以上より，黒い部分木が存在するような r の個数を求めればよいことになります．

実は，黒い部分木が存在するような r というのは，ある下限と上限の間の連続区間になっています．まずは上限から考えてみましょう．黒い部分木の半径がちょうど r となるためには，v にぶら下がっている各部分木のうち，黒い頂点を深さちょうど r に含むようなものが，少なくとも 2 本なければなりません．よって，v にぶら下がっている各部分木のうち，深さが 2 番目に大きいものの深さを d_2 とすると，$r \leq d_2$ でなければなりません．よって，r の上限は d_2 です．

次に，r の下限を考えてみましょう．もし，v がお気に入りの頂点ならば，v を選んで距離 $d = r$ とすることで，$0 \leq r \leq d_2$ の範囲の r をすべて実現できます．よって，この場合 r の下限は 0 です．では，v がお気に入りの頂点でない場合はどうすればよいでしょうか？ この場合，v 以外のお気に入りの頂点 w を代わりに選び，うまく距離 d を設定することで，黒い部分木の center vertex が v かつ半径が r になるようにしなければなりません．v にぶら下がっている各部分木のうち，頂点 w を含むものを T_w と書くことにします．距離 d を設定する際には，T_w の頂点がすべて黒くなる必要があることが分かります．T_w に白い頂点が残っていると，黒い部分木全体の center vertex が v になりえないからです．逆に，T_w の頂点がすべて黒くなるように距離 d を十分大きくすれば，あとは $r \leq d_2$ の範囲で r をいくらでも大きくできます．ただし，T_w で最も深い頂点が黒いという前提があるので，r は T_w の深さより小さくはできません．よって，お気に入りの頂

点 w としてどのようなものを選ぶべきかというと，v にぶら下がっている各部分木のうち，最も深さの小さい部分木に含まれるお気に入りの頂点を選ぶべきだと分かります．以上より，v にぶら下がっている各部分木でお気に入りの頂点を含むもののうち，最も深さの小さいものの深さを d_1 とすると，$d_1 \leq r \leq d_2$ の範囲の r をすべて実現できます．よって，r の下限は d_1 です．

では次に，center edge を辺 (u, v) に固定したときに何通りの黒い部分木があるかを考えます．この場合も，center vertex を固定したときと同様に，半径 r を決めると黒い部分木が一意に定まります．すなわち，頂点 u 側の頂点たちについて，頂点 u からの距離が r 以下ならば黒色，r より大きければ白色となります．頂点 v 側の頂点たちについても同様です．ここで，お気に入りの頂点を u 側の頂点たちから選んできたとします．このとき，先の議論と同様にして，頂点 u 側の頂点はすべて黒くなる必要があることが分かります．頂点 v 側についても同様なので，結局，頂点 u 側か頂点 v 側のどちらかはすべて黒くなっているということがいえます．また，辺 (u, v) が center edge となるためには，頂点 u 側の部分木と頂点 v の部分木のうち，より浅い方 (同じ場合はどちらでもよい) がすべて黒くなる必要があります．よって，実は center edge を辺 (u, v) に固定したときの黒い部分木は (存在すれば) 1 通りです．存在を判定するには，頂点 u 側の部分木と頂点 v の部分木のうち，より浅い方 (同じ場合はどちらでもよい) にお気に入りの頂点が含まれるかをチェックすればよいです．

以上の計算に必要なすべての情報は，全方位木 DP によって計算量 $\mathrm{O}(N)$ で求められます．

AGC 008 Editorial

writer : sugim48

December 25, 2016

A : Simple Calculator

You can prove that the optimal solution is always of the following form:

(Push B 0 or 1 times) → (Push A 0 or more times) → (Push B 0 or 1 times)

Proof: In the optimal solution, the substrings "BB" and "ABA" never appear because they are equivalent to "" and "B". Thus, "B" appears only at the ends of the operation sequence.

Now you can try four patterns: whether you push B at the beginning (change x to $-x$) or not, and whether push B at the end (change y to $-y$) or not. After changing the signs of x and y, if $x \leq y$ holds, this sequence is possible with $y - x$ more operations.

B : Contiguous Repainting

Let's see the operations in the reverse order. The problem will be as follows:

> Initially all cells are unpainted. You repeat the following operation: choose K consecutive cells and paint it black or white. However, once a cell is painted, the color of the cell never changes.

Obviously, after all cells are painted, there is at least one subsegment of length K that is painted with the same color. (The K cells that are colored in the first operation satisfy this condition).

On the other hand, this condition is sufficient: once you color K consecutive cells with the same color, you can determine the color of the remaining cells arbitrarily.

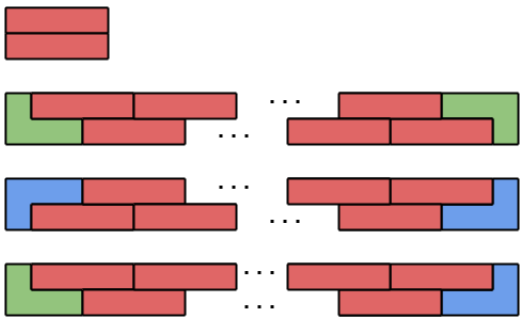
Thus, we get the following $O(N^2)$ solution: brute force all subsegments of length K, and for this segment and for each of the other cells, determine the color greedily. With proper pre-computation, this can be done in $O(N)$ time.

C : Tetromino Tiling

We can't use T, S, Z minoes: if you put one of these minoes on a $2 \times 2K$ board, the board will be splitted into two parts whose areas are odd.

We can always use all O minoes: when we can create a $2 \times 2K$ rectangle with an O mino, we can create a $2 \times 2K - 2$ rectangle without it. Thus, the answer is the sum of a_O and the maximum size of the rectangle that can be formed by only using I, J, L minoes.

When we can use only I, J, L minoes, it is easy to see that there are only four patterns (and their concatenations):



In the last three patterns, when there are two or more I minoes, we can remove it and create an I + I type rectangle. Thus, it is sufficient to consider the following four patterns:

Also we can assume that we use the I + J + L pattern at most once (otherwise we can create I + I, J + J, and L + L). Thus, we can try two possibilities for the number of used I + J + L patterns, and the number of the other patterns can be computed easily.

D : K-th K

We should divide the integers $1, \ldots, N^2$ into the following sets:

- Integers $x_1, \ldots, x_N$
- Sets $L_1, \ldots, L_N$. L_i corresponds to the positions of is to the left of i-th i. $|L_i| = i - 1$ and each element in it must be smaller than x_i.

- Sets $R_1, \ldots, R_N$. R_i corresponds to the positions of is to the right of i-th i. $|R_i| = N - i$ and each element in it must be greater than x_i.

Now, in the order $p = 1, \ldots, N^2$, we classify p into one of the sets described above.

When $p = x_i$ for some i, it is obvious. Now we assume that there is no such i.

We say p can be assigned to L_i if the current size of L_i is still smaller than the desired size of it and p is smaller than x_i. Similarly, define "p can be assinged to R_i".

In case there is no set that p can be assigned to, the answer is "No". In case there are muultiple such sets, how can we choose a set?

First, when p can be assigned to both L_i and R_j for some i, j, we can prove that p should be assigned to an L-type set. This is because if we assign p to R_j and, in the future, if we assign q to L_i, we can swap p and q.

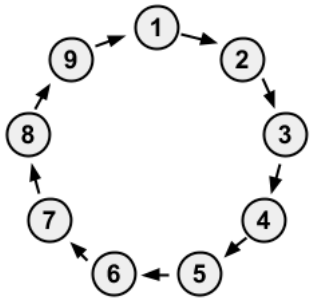
When p can be assigned to both L_i and L_j ($x_i < x_j$), p should be assigned to L_i for a similar reason.

Also, when p can be assigned to multiple R-type sets, it doesn't matter which to choose - again for a similar reason.

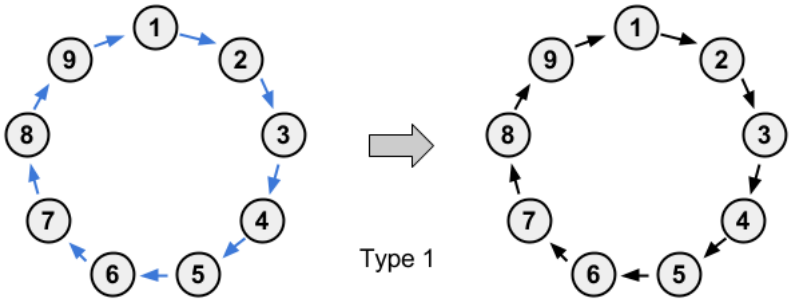
This way we can greedily determine the set p is assigned to, in $O(N^3)$ or $O(N^2 logN)$ time.

E : Next or Nextnext

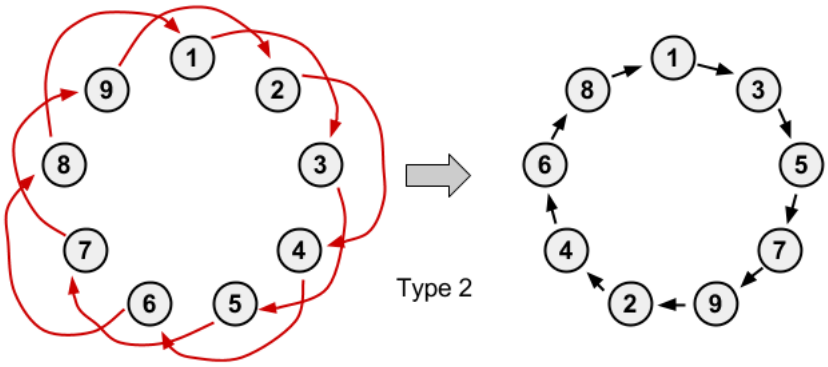
Construct a graph with N vertices and for each i, add an edge from i to p_i. This graph is a set of cycles. Consider one of the cycles in the graph.



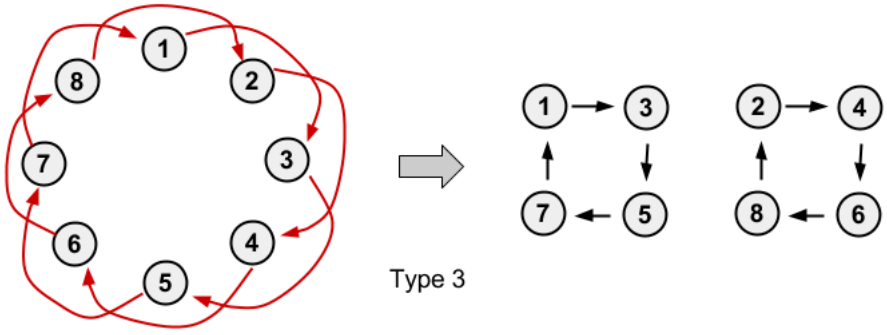
We construct a new graph by adding edges from x to p_x or p_{p_x} for each vertex x in this cycle: There are four possible types for the new graph:



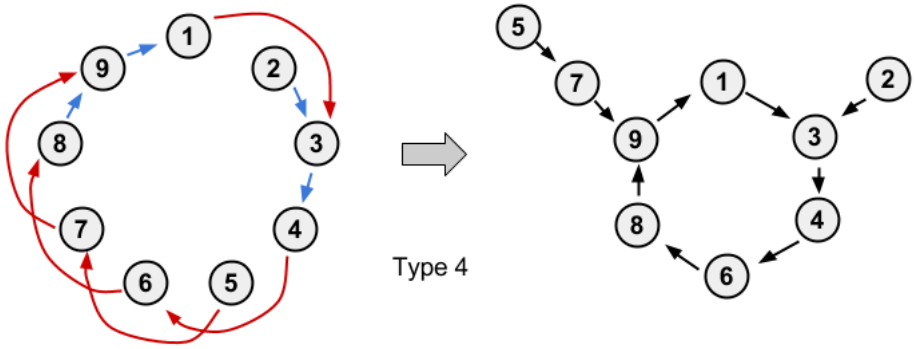
Type 1



Type 2



Type 3



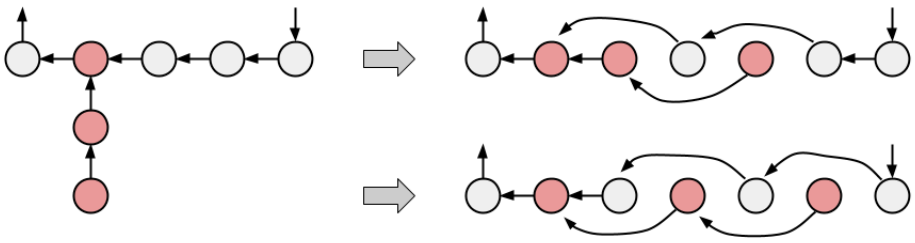
- In Type 1 we get the same graph.
- Type 2 occurs only when the size of the cycle is an odd number (except for 1), and we get an isomorphic graph.
- Type 3 occurs only when the size of the cycle is an even number, and the cycle is splitted into two cycles of the half size.
- Type 4 is a cycle with "feet". No two feet can grow from the same vertex and there are no branches in the feet.

Let's return to the original problem. First construct a similar graph for the sequence a (that is, add edges from i to a_i). Each connected component in this graph must be a cycle or "a cycle with feet". Otherwise the answer is zero.

First, let's think about cycles. Two cycles of the same size may come from the same cycle in the graph of p (because of Type 3).

Suppose that there are n_k cycles of the size k. We can count the number of ps that corresponds to these cycles in $O(n_k)$ DP. Each cycle may come from Type 1 or Type 2, or it may be paired with another cycle and come from Type 3. Note that when we use Type 3, there are k ways to "merge" two cycles.

Next, consider "a cycle with feet". Since this comes from Type 4, we can compute the number of ways for each connected component independently. For each foot, we can merge it with the "cycle part" in at most two ways:



Let l_1 be the length of the foot. Let l_2 be the distance between the root of current foot and the root of previous foot.

- If $l_1 < l_2$, there are 2 ways.
- If $l_1 = l_2$, there is 1 way.
- If $l_1 > l_2$, there are 0 ways.

In summary,

- First constcuct a graph with edges from i to a_i.
- If there is an "invalid component", the answer is zero.
- Then, we first handle simple cycles: count the frequency of cycles by size and do DP.
- Then, we handle cycles with feet: for each root of foot, there are 0 or 1 or 2 ways to merge it with the cycle part.
- The answer is the product of these numbers.

In total, we can solve the problem in O(N).

F : Black Radius

Let $f(x, d)$ be the set of vertices y such that the distance between x and y is at most d. The partial score of this task asks the number of sets that can be written of the form $f(x, d)$ (and we call such sets "good").

Let S be a good set. In general, it can be written of the form $f(x, d)$ in multiple ways, so in order to avoid duplicates it is important to define the "canonical form". For simplicity, we assume that S is not the entire set. Note that the entire set is always a good set.

Suppose that $S = f(x, d_1) = f(y, d_2)$ for some two distinct vertices x, y. Let $x, v_1, v_2, \ldots, v_{k-1}, y$ be the path from x to y. If we analyze this situation carefully, we can prove that there is some vertex z on this path and the following holds:

$$f(x, d_1) = f(v_1, d_1 - 1) = f(v_2, d_2 - 1) = \ldots = f(z, d_3) = \ldots = f(v_{k-1}, d_2 - 1) = f(y, d_2) \quad (1)$$

In other words, for each set there is a canonical form (in the case above $f(z, d_3)$), and other representations of this set is of the form $f(v, d_3 + dist(v, z))$. Note that the set $f(v, d_3 + dist(v, z))$ may not be the same for all vertices v: however, the set of such vs form a subtree (connected subset in the tree).

Now, how can we get the partial score? For each vertex x, we compute the maximum integer d such that

- $f(x, d)$ is not the entire set.
- $f(x, d)$ is not the same as $f(y, d - 1)$ for any vertex y adjacent to x.

In order to compute d, first we see the tree as a rooted tree rooted at x. Then, for each child y of x, compute the depth of the deepest vertex in the subtree of y. This can be done in $O(N)$ tree DP in total for all x: make sure not to write $O(N^2)$ solution when the given tree is a "star". With this pre-computation the value of d can be computed easily.

Now, the sets $f(x,0),\ldots,f(x,d)$ corresponds to the good sets written in canonical forms, so the answer is the sum of $d+1$ for each vertex x plus one (for the entire set).

How can we get the full score?

We see that the good sets can be written in "canonical form" in a unique way, and that corresponds to $\{\ f(x,i) | i \leq high_x\ \}$ for some array $high_x$. Now, for each $f(x,i)$ that is written in canonical form, we want to decide if it can be rewritten of the form $f(y,j)$ using a good vertex y.

- When x is a good vertex, of course this is possible.
- When x is not a good vertex, there is a number low_x such that the set $f(x,i)$ can be rewritten if and only if $i \geq low_x$.

Thus, we can compute the array low and the answer is the sum of $high - low + 1$ plus one.