C: Vacant Seat `= A. Vacant Seat`

Consider the following two types of intervals (we call it "good interval"):

(a) A closed interval $[l, r]$ such that $r - l$ is odd and l, r are filled by people with the same sex.
(b) A closed interval $[l, r]$ such that $r - l$ is even and l, r are filled by people with different sex.

In both cases, we can see that the interval contains at least one empty seat. (Otherwise, males and females must sit alternately in the interval $[l, r]$, and we get a contradiction in both cases.)

We use binary search, and find an empty seat in $O(\log N)$ queries.

- First, perform a query on seat 0. If this is empty, we finish. Otherwise we find a good interval $[0, N]$. (Since the seats are cyclic, we can call seat 0 "seat N").
- Suppose that we have a good interval $[l, r]$. Let $m := \lfloor l + r \rfloor / 2$ and perform a query on seat m. If this is empty, we finish. Otherwise, we can prove that at least one of intervals $[l, m]$, $[m, r]$ will be a good interval by a simple argument about parities.

G: Colorful Doors = `B. Colorful Doors`

Make the segment cyclic, that is, assume that door 1 is to the right of door $2N$. Now we have a circle, there are $2N$ doors on the circle, and there are also $2N$ section between two doors. The new section between $2N$ and 1 should be walked through.

Under this setting, it's easier to prove the claim from the statement: "It can be shown that he will eventually get to the right bank". Choose an arbitrary section (call it section s), and start walking from section s (keep walking forever). Since the number of sections is finite, he will eventually pass through the same section twice. So, the sequence of sections we pass through will be periodic from some point: "$s, \cdots, t, \cdots, t, \cdots, t, \cdots$,". Furthermore, since the section we pass through immediately before a certain section can be uniquely determined, the entire sequence will be periodic: "$s, \cdots, s, \cdots, s, \cdots, s, \cdots$,". This proves the claim from the statement because if we start from section 0, we will return to this section in the future.

By the same observation, we can divide the $2N$ sections into one or more cycles. We want to find a coloring of doors such that the set of sections represented by '1' in the input corresponds to one of the cycles.

- In case all sections should be walked through (i.e., all sections are in the same cycle)
  - (i) In case $2N = 4k$ for some k, the sequence of doors $1, 2, 1, 2, 3, 4, 3, 4, 5, 6, 5, 6, \ldots, 2N - 1, 2N, 2N - 1, 2N$ is a valid answer.
  - (ii) In case $2N = 4k + 2$, we can prove that the answer doesn't exist. Let's start from the case $N = 1$. In this case the number of cycles is two. After that, increase N one by one. In each step, we can see that the parity of the number of cycles always change, regardless of the positions of two new doors we add. Thus, when $2N = 4k + 2$, we have even number of cycles, and all sections can't be in one cycle.
- Other cases

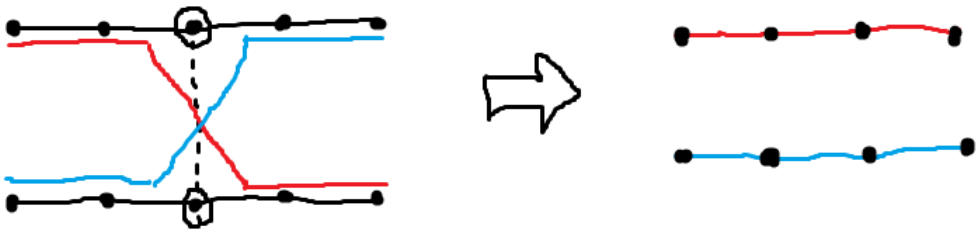
  Now, the set of '1's will be a set of two or more paths. Let's define the length of a path as the number of doors in the path, excluding two doors at the ends. For example, if $s =$ `010110011`, after we make it cyclic we get $s =$ `1010110011`, and the set of lengths of paths is $0, 1, 2$. It's easy to see that only this set matters: the order of these paths or the number of '0's between them don't matter. Now, we consider several cases depending on the sum of these lengths.

  Let's call a door "internal" if it's between '1' and '1', "external" if it's between '0' and '0', and "boundary" otherwise. An internal door must be passed through in both directions, a boundary door must be passed through in one direction, and an external door must not be passed through.
  - (iii) The sum of these lengths is odd

    From the observation above, an internal door must be matched with another internal door. However, in this case, the total number of internal doors is odd, and this is impossible.
  - (iv) The sum of these lengths is $4k$

Suppose that the set of lengths contains $\{x\}$ and $\{y\}$. By connecting the first section of $\{y\}$ immediately after the last section of $\{x\}$, we can regard them as $\{x+y\}$. By repeating this, we will eventually have a single element divisible by 4, and it reduces to the case (i).

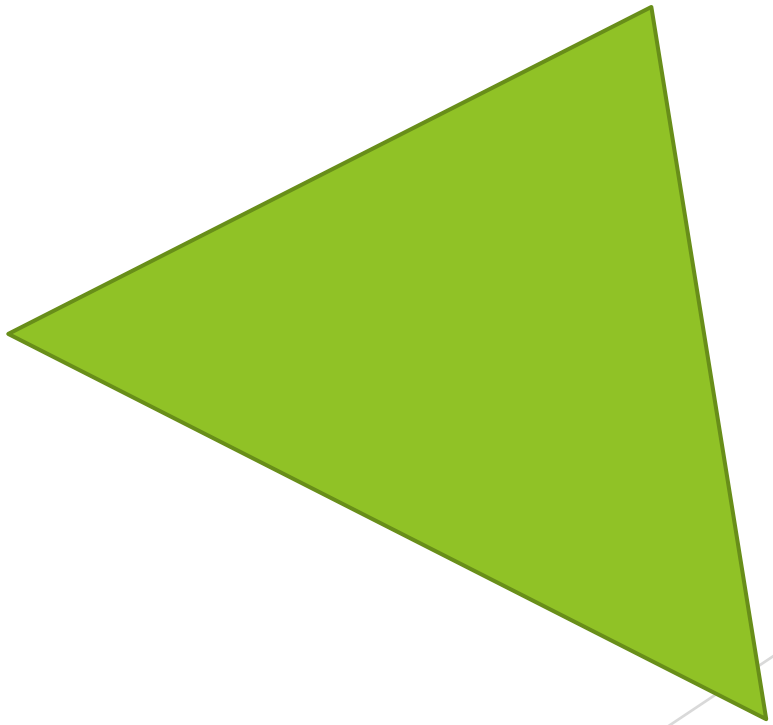
- The sum of these lengths is $4k+2$ - now we have two cases:
  - (v) We have at least two paths with nonzero lengths
    Suppose that the set of lengths contains two nonzero elements $\{x\}$ and $\{y\}$. Choose one internal door from each path and match them. Then, these two paths can be regarded as two paths whose sum is $x+y-2$ (see the picture below). It reduces to the case (iv).



  - (vi) All paths but one have lengths of zero
    All paths of lengths zero must be merged together, and this reduces to the case (ii). This is impossible.

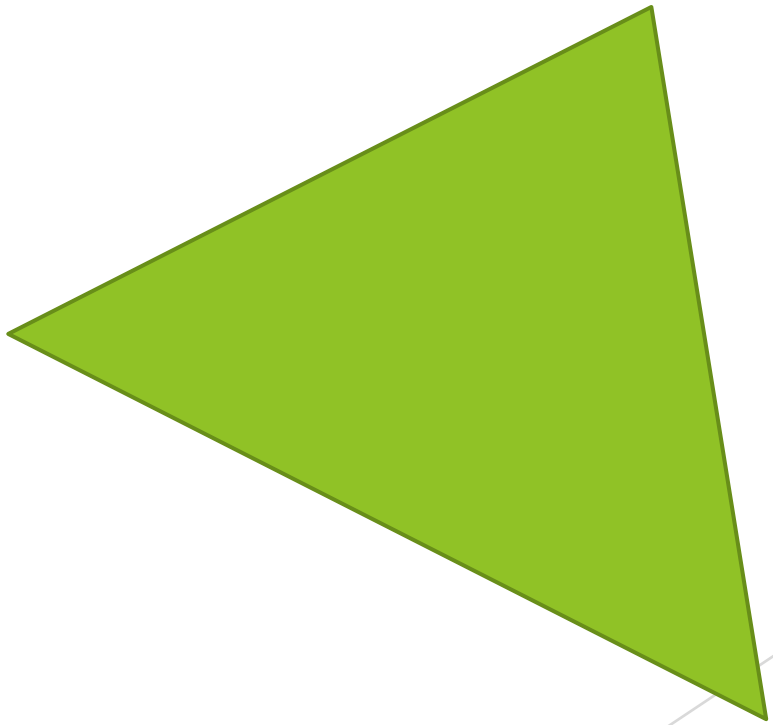
G: Construct One Point `= C. Construct Point`

- 三角形が与えられる
- 中の点を1個見つける



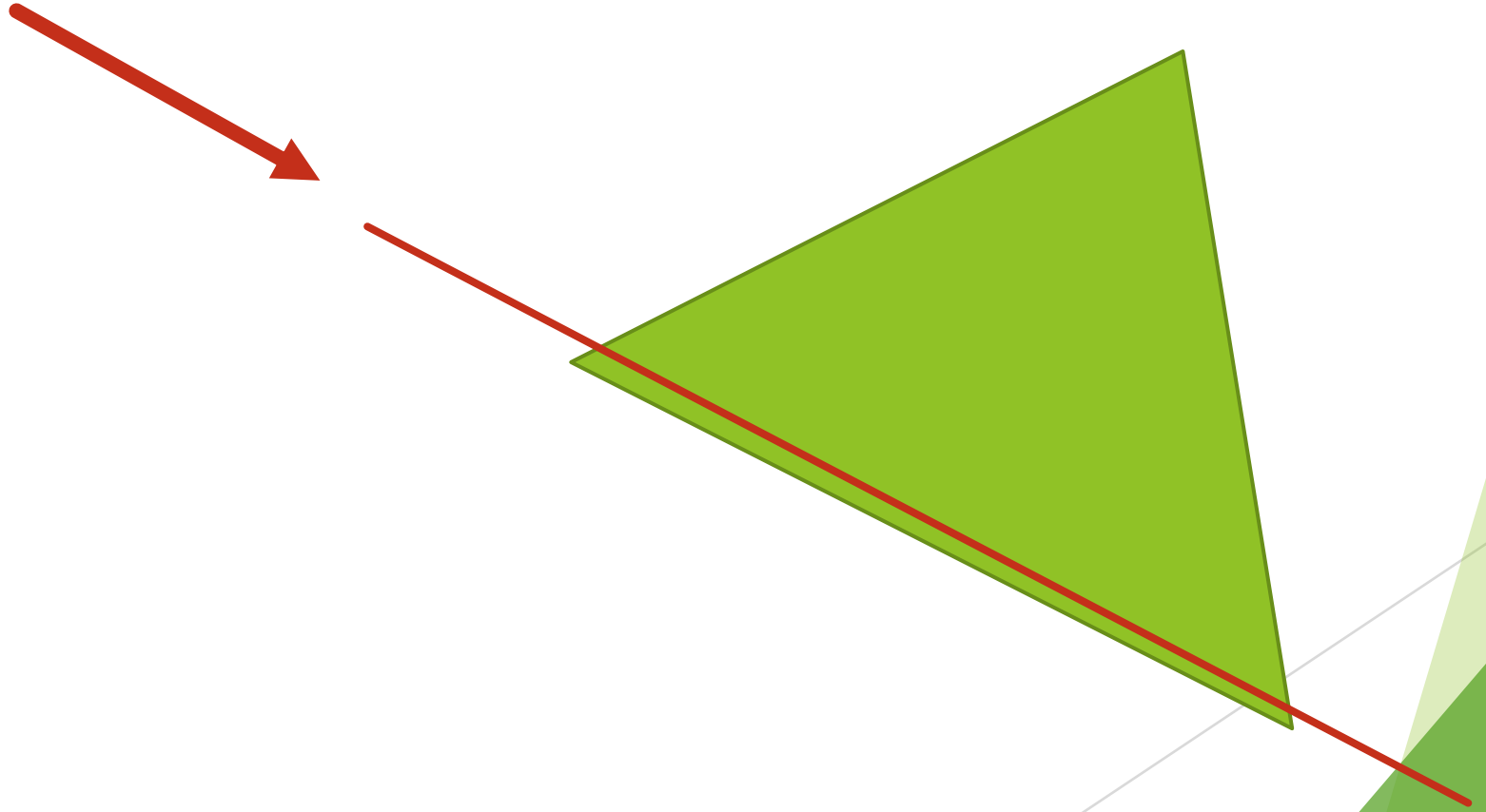
G: Construct One Point

- 実は内側に点があるかの判定は簡単
- ピックの定理
- 三角形の面積，辺上の点の個数
  から内側の点の個数がわかる
- じゃあ復元は？
  - 難しい(びっくり)



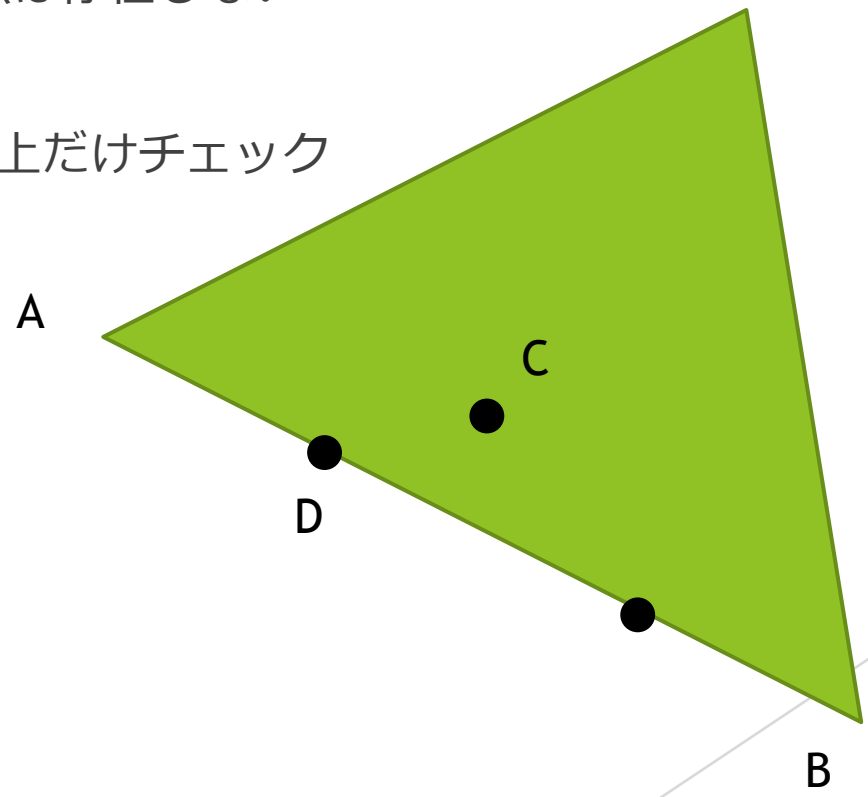
G: Construct One Point

- 実はこの線上だけチェックすれば良い(とは？)



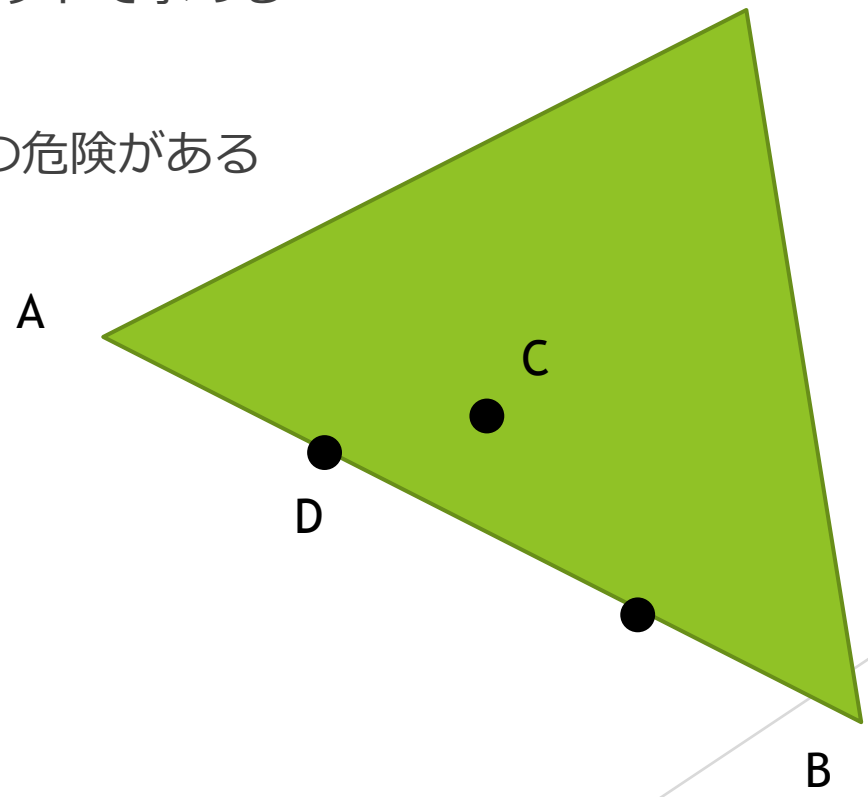
G: Construct One Point

- AC上, CB上, ACBの内側に点が無いような点Cが存在する
- すると三角形ACDの辺上，内側に点は存在しない
- よって三角形ACDの面積は0.5
- 面積が0.5になるようにずらした線上だけチェック



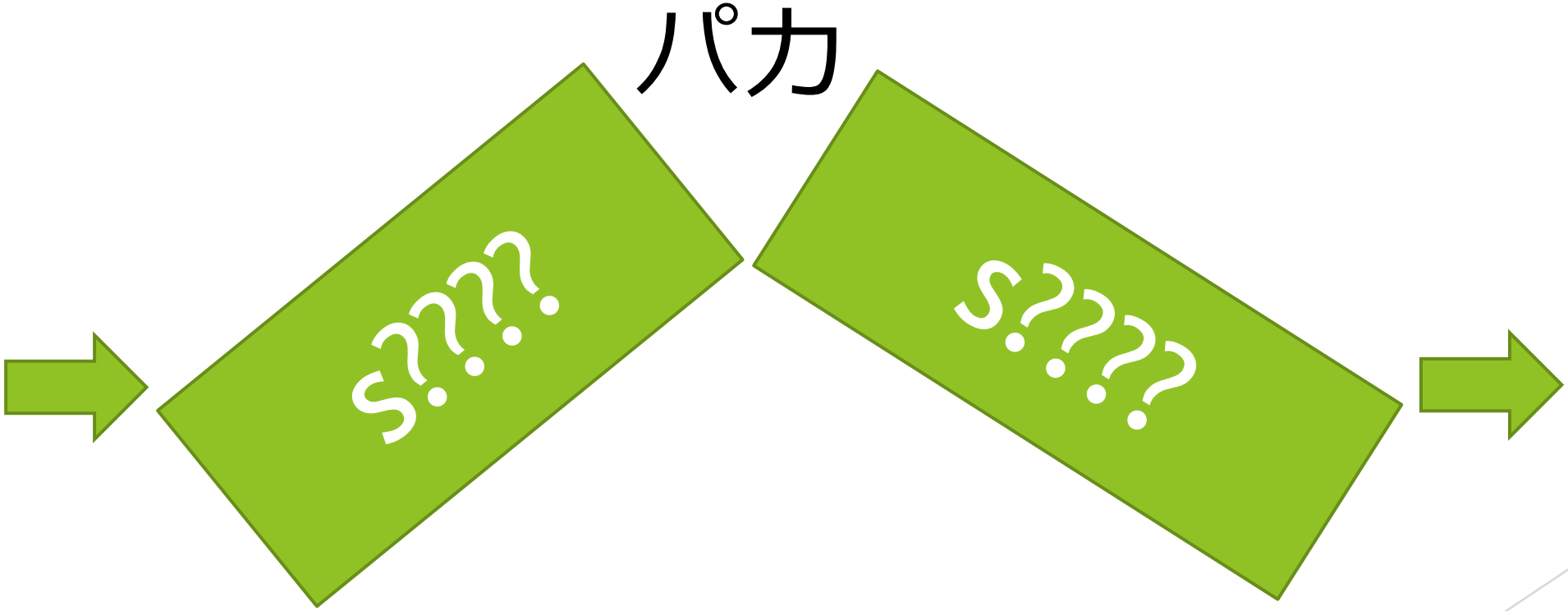
G: Construct One Point

- 面積が0.5になるようにずらした線上だけチェック
- この線上のうち1点は拡張ユークリッドで求める
- あとは二分探索で適当に
- あらゆるところでオーバーフローの危険がある



D: Knapsack And Queries = D. Knapsack and Queries

D: Knapsack And Queries



D: Knapsack And Queries

- Queueをstack * 2で実装する
- Queueと違ってstackなら常にDPテーブルを持てる(挿入するたびにその時点のDPテーブルを保持)
- 2つのDPテーブルに対して高速にクエリは処理できる(スライド最小値)

F: XOR Tree = E. XorTree

For a vertex v, define b_v as the XOR of all values assigned to edges incident to v. If we perform an operation on the path between two vertices u and v with the value xm we change b_u and b_v to $b_u XOR x$ and $b_v XOR x$, respectively. Also, note that all b_v will be zero if and only if all edges are assigned zeroes. Thus, the problem can be restated as follows:

> You are given a sequence of integers $b_1, b_2, .., b_N$ (here, all integers are up to 15). In each operation, you choose i, j, x, and XOR x into b_i and b_j. How many operations do you need to make all integers zeroes?

Now, consider a graph with N vertices $1, 2, .., N$. Initially, this graph doesn't have edges. Whenever you perform an operation for indices i and j, add an edge between vertices i and j.

After we perform all operations, this graph will have several connected components. When you perform an operation between i and j, the value $b_i XOR b_j$ doesn't change. Thus, the XOR of all values in a single connected component never changes.

Therefore, we want to add minimum possible number of edges to this graph such that in each connected component, the XOR of all b_i is zero. Obviously, each connected component should be a tree if we want to minimize the number of edges. In this case, the total number of edges is N minus the number of connected components.

Thus, we can again restate the problem as follows:

> You are given a sequence of integers $b_1, b_2, .., b_N$ (here, all integers are up to 15). Divide these numbers into disjoint set, such that in each set the XOR of all values is zero. The answer is N minus the maximum possble number of sets we get.

First, we should make sets as follows:

- If we have a zero, we should create a set with this single zero.
- If we have two identical numbers, we should create a set with these two numbers.

After these process, we will have at most 15 elements. Now, we can compute the maximum number of sets in a simple $O(3^k)$ DP, where k is the number of remaining elements.

E: Antennas on Tree `= F. Antennas on Tree`

Consider two vertices s, t in the tree. When can we distinguish these vertices?

If the distance between them is odd, we can always distinguish them as long as we have at least one antenna (by checking the parity of distance from the antenna). Otherwise, let u be the midpoint between s and t. We see s, t and antennas from u. If the direction of at least one antenna is the same as the direction of s or t, we can distinguish them.

Thus, the condition can be restated as follows:

> For each vertex v, the following holds. Remove v from the tree, and suppose that we get k subtrees. Then, at least $k-1$ of the subtrees must contain antennas.

Now, we'll describe the solution. In case the tree is a path, the answer is one: we can put an antenna on one of the leaves. Otherwise, a vertex r with degree at least 3 exists. Make it the root of the tree.

The condition can be again restated as follows:

> For each vertex v in the rooted tree, the following holds. If v has k children, at least $k-1$ of k subtrees whose roots are children of v must contain antennas.

This is because, if v is not r, v always contain at least one antenna in the "parent direction". (Otherwise, the condition above won't be satisfied for the root r).

Now we can compute the answer by a simple DP. Define $dp[v]$ as the smallest number of vertices we must choose from the subtree rooted at v, when we want to satisfy the conditions above for all vertices in this subtree.

J: Rectangles `= G. Rectangles`

The 2D version of the problem is easy. Consider two adjacent rectangles. If these two rectangles are not "aligned" well, we can uniquely determine the positions of more rectangles, and we will eventually fill a entire row or a column. Thus, there is a line parallel to one of coordinate axis that doesn't split any rectangles. It's not hard to count such patterns.

Now, let's solve the original 3D problem. Assume that A, B, C are multiples of a, b, c, respectively (otherwise the answer is zero). We call a pattern "trivial" if there is a plane that doesn't split any cuboids. We can count the number of trivial patterns easily. The number of patterns that can be cut by a plane can be reduced to the 2D case. Do not forget to avoid double-counting by using inclusion-exclusion principle. For example, you should subtract the number of patterns that can be cut by planes of two directions.

The main challenge is that, in 3D case, there are non-trivial patterns. Let's think how these patterns look like.

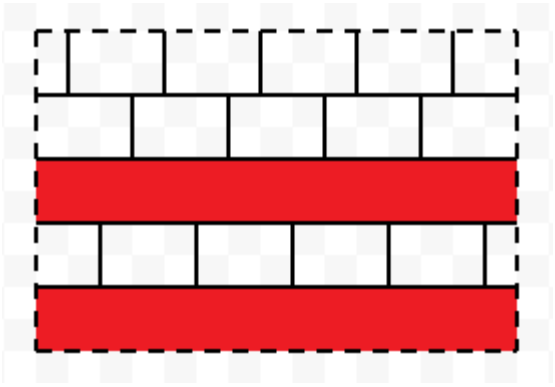
First, for a torus cuboid with parameters (p, q, r) in the statement, write an integer k into the small-cube $\{((p+i) \mod A, (q+j) \mod B, (r+k) \mod C))\}$. This way all small-cubes will contain an integer. Let $v(x, y, z)$ be the integer written on (x, y, z) For each pair (x, y), the sequence $v(x, y, 0), v(x, y, 1), \cdots, v(x, y, C-1)$ will be a cyclic shift of $0, 1, .., c-1, 0, 1, .., c-1, .., 0, 1, .., c-1$. Thus, the values of $v(x, y, 0)$ determines all values of v. Let $v(x, y) = v(x, y, 0)$, and consider an $A \times B$ table whose (i, j)-element is $v(i, j)$.

Now, for a given table, we want to count the number of patterns that are consistent with this table. Fix a pattern that is consistent with the table. In each layer (here a layer means a plane with constant value of z-coordinate), we get a partition of the entire $A \times B$ rectangle into $a \times b$ torus rectangles that corresponds to the pattern. Here, notice that a torus rectangle never contains two cells with different values of v, and the partition at the layer with $v(x, y, z) = 0$ determines everything else.

Thus, we get the following. Let $f(h)$ be the number of ways to partition the set of cells (x, y) such that $z(x, y) = h$ into torus rectangles. Then, the number of patterns that is consistent with the table is $f(0)^{C/c} \times \cdots \times f(c-1)^{C/c}$.

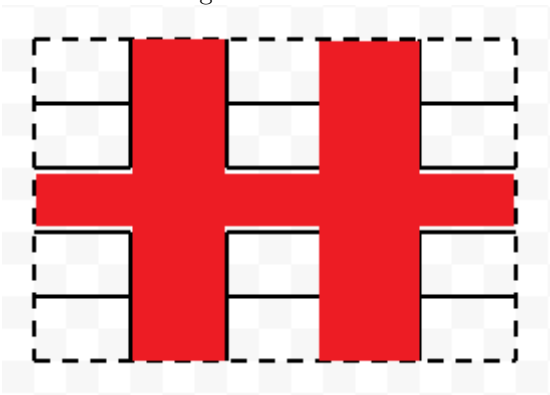
Now, fix an $A \times B$ table (with the values of v). When do we have non-zero number of non-trivial patterns that is consistent with this table?

We call this table "row-aligned" if in each row, all numbers are the same. Similarly, define "column-aligned". If the table is both row-aligned and column-aligned, it means that the table only contains a certain constant, and this corresponds to patterns that can be cut by xy-plane. Since this pattern is trivial, we can ignore it. If the table is row-aligned (or similarly, column-aligned), the table is multiple stripes whose heights are multiples of a (see the picture below). In this case the only way to partition it into torus rectangles is to entirely divide it into stripes with heights a, thus again it corresponds to a trivial pattern. Thus, we assume that this table is not aligned in any directions.



Consider a way to partition this table into torus rectangles of dimensions $a \times b$. Each torus rectangle must contain the same values of v. As we see in the 2D case, this partition is either "horizontal" (i.e., a union of stripes of height h, some stripes are possibly shifted horizontally), or "vertical". If all such ways are "horizontal" (or "vertical"), it corresponds to a trivial pattern. Thus, there must be both horizontal ways and vertical ways to partition it. It means that the entire $A \times B$ table must be splitted into $a \times b$ torus rectangles in the most natural way (i.e., all rectangles are aligned like a grid). Also, since the table is not aligned, there is a unique way to do so. Let's call it "standard partition".

In order for the pattern to be non-trivial, in at least one layer the partition must be shifted to horizontal direction, and in at least one layer the partition must be shifted to vertical direction. This means that there exists an integer h that dominates some rows and some columns, as in the picture below:



Since we have a standard partition, now we can consider the entire table as an $A/a \times B/b$ table. As we see above, there are h, p, q such that h dominates exactly p rows ($0 < p < A/a$) and q columns ($0 < q < B/b$). In this case, the number of standard partition is 1, the number of horizontally shifted partitions is $b^p - 1$, and the number of vertically shifted partitions is $a^q - 1$. We want to count the number of ways to choose a sequence of C/c partitions such that at least one is horizontally shifted, and at least one is vertically shifted. This value is $(b^p + a^q - 1)^{C/c} - (b^p)^{C/c} - (a^q)^{C/c} + 1$.

To summarize, the solution is as follows:

- Count the number of trivial patterns by inclusion-exclusion.
- Let's count the number of non-trivial patterns. First we fix a standard partition (ab ways) and the value of h (c ways). Then, for each pair (p, q), compute the following values:

- Suppose that we have an $A/a \times B/b$. How many ways are there to fill this tables with integers between 0 and $c-1$, such that exactly p rows are dominated by h and exactly q columns are dominated by h? This can be done by a simple $O(N^4)$ DP ("exclusion principle").
- Compute the value $(b^p + a^q - 1)^{C/c} - (b^p)^{C/c} - (a^q)^{C/c} + 1$.

We compute the product of two values above, compute the sum for all pairs (p, q), and multiply it by a factor of abc.

H: Generalized Insertion Sort `= H. Generalized Insertion Sort`

Assume that each vertex contains a ball, and each ball has an integer written on it. We move balls by operations described in the statement.

Here's our plan. First, we choose an arbitrary leaf, and perform some operations until we assign a correct ball to the chosen leaf. After that, we can ignore this leaf (never perform operations that affect this leaf). By repeating this process from leaves to roots, we can achieve the goal.

To reduce the number of operations, we process some leaves (and other vertices) at once, as follows. We define "leafish vertex" as follows:

- A leaf is a leafish vertex.
- If a vertex has exactly one child, and the child is leafish, is parent is also leafish.

Each time we remove all leafish vertices, the total number of leaves is always halved. It means that we can make the tree empty by repeating the process "remove all leafish vertices "$O(logn)$ times. Thus, if we can assign correct balls to all leafish vertices in $O(n)$ operations, we can achieve the goal in $O(nlogn)$ operations.

How do we do that? Let's color balls. We color the balls that should be assigned to leafish vertices red. Other balls are white.

Then, repeat the following:

- If the ball at the root is red, insert it into a "correct position", and color it red. More specifically, look at the path that contains the destination of this ball. Check balls in the path from bottom to top. Whenever we find a non-blue ball or a blue ball that should be above our current ball in the final position, insert our current ball there.
- Otherwise, call the ball at the root B. Then, find the bottommost ball that is red or white: call it C. Paint B black, and insert it to the position of C.

Here, blue balls represent balls that are assigned to the correct path, in "correct order". At each moment, all blue balls are in leafish vertices, and inside each path, the blue balls are below other balls. Furthermore, the relative order of blue balls are the same as there order in the final position.

How many operations do we need for this process? Since we color red and white balls after the very first operations, the total number of operations for those balls is at most n. Blue balls never appear at the root because all of them are at leafish positions. Black balls may reach the root only after we perform an operation for red balls. Thus, the total number of operations for black balls is at most (the number of leafish vertices), and the total number of operations for all operations is $n+$ (the number of leafish vertices).

The total number of phases is at most 11, and in each phase the number of operations is at most $n+$ (the number of leafish vertices). Thus, we need at most 24000 operations.

By the way, if you want more efficient solution, we can actually do it in $O(n)$ operations. Here's a brief sketch of the solution:

- See operations in the reverse order: we insert a chosen ball into the root.
- We define "broom", a set of vertices that satisfies the following properties. Let x be an arbitrary vertex in the tree, and $c_1, \cdots, c_k$ be a subset of children of x. A broom consists of two parts: a "path part" and a "tree part". The path part is simply a path between the root and x. The tree part is the union of subtrees rooted at $c_1, \cdots, c_k$.
- Consider a broom with p vertices in the path part, and q vertices in the tree part. We call it "balanced" if $2q - c \leq p \leq 4q + c$ (where c is a small constant).
- Prove that a tree always have a balanced broom.
- Take a broom from a tree. Paint all q balls that should be assigned to the tree part of the broom red. Find a way to put all red balls into the path part of the broom, in $O(p + q) = O(q)$ steps.
- Suppose that now all red balls are in the path part of the broom. Find a way to put all red balls into correct positions in $O(p + q) = O(q)$ steps.
- Now, just repeat this and we achieve the goal.

I: ADD DIV MAX RESTORE `= I. ADD, DIV, MAX`

- ADD DIV MAXならsegment tree beatsで解けます(なので += RESTORE)
- 計算量はO((N+Q) logN)です

- (RESTOREは無視すると) f(x) := (a + x) / b型の関数を遅延伝搬segtreeに乗せると，解けそう？
- 素直に書くと落ちます

I: ADD DIV MAX RESTORE

- 合成自体は可能
- 値が大きくなりすぎる
- f(x) = (x + a) / b + cとすれば，cは小さくできる
- bが大きくなりすぎたらf(x) = (x + a) / 10^9 + cへ変換する
- Restoreは更にフラグを1個増やせば遅延伝播可能

I: Simple APSP Problem `= J. Simple APSP Problem`

The black cells will be very sparse, and the vast majority of rows and columns will be entirely white.

Suppose that two adjacent rows, the i-th row and the $i+1$-th row, are entirely white. Let's call the region in the i-th row and above "UP", and the $i+1$-th row and below "DOWN". It's easy to see that the shortest path between two cells crosses an edge (here, we imagine that white cells correspond to vertices and there are edges for each adjacent pair of white cells) between the i-th row and the $i+1$-th row if and only if one of the cells is UP and the other is DOWN.

Thus, if we change the costs of all edges between the i-th row and the $i+1$-th row to zeroes, the answer decreases by the following value:

(The number of white cells in UP region) $\times$ (the number of white cells in DOWN region)

Now, we can "compress" two adjacent empty rows, and we can do the same for columns. By repeating these operations, we can compress the entire grid into a smaller grid with $H, W \leq 2N$, and we can simply solve the problem by BFS in $O(N^4)$.

Note that we should add "weights" to cells after compressions. For example, in the very first compression, if we merge the i-th row and the $i+1$-th row into a single row, the cells of this row should have a weight of 2. Then, in the final grid, the distance between two cells p and q should be added to the answer with the coefficient (the weight of p) $\times$ (the weight of q).

D: Forest `= K. Forest Task`

If $M = N - 1$, the input is a tree, and the answer is 0. Assume that $M < N - 1$.

First, divide the forest into connected components (trees). We want to merge these trees into a single tree by adding edges.

Here, we have the following constraints:

- From each tree, at least one vertex must be chosen (otherwise this tree will be isolated).
- There are $N - M$ trees. Thus, we must add $N - M - 1$ edges. This means that the total number of chosen vertices must be $2(N - M - 1)$.

It turns out that these conditions are sufficient. That is, if a subset of vertices satisfies the conditions above, we can actually make a tree by choosing those vertices. For example, we can connect all trees by repeating the following: choose two components with the biggest number of chosen (and still unused) vertices, and connect them.

The answer is as follows. First, if $N < 2(N - M - 1)$, the answer is "Impossible" (from the second condition). Otherwise, we choose vertices greedily as follows:

- First, from each component, choose a vertex with the smallest cost.
- Then, from remaining vertices, choose vertices from the smallest costs, until we choose $2(N-M-1)$ vertices in total.

These operations can be implemented in $O(NlogN)$ time.