Problem A
A Totient Quotient
Time Limit: 1 Second, Memory Limit: 2G

For a positive integer k, Euler's totient function $\phi(k)$ is defined as the number of positive integers less than or equal to k and relatively prime to k. For example, $\phi(9) = 6$, $\phi(24) = 8$, and $\phi(1) = 1$. (As a reminder, the greatest common divisor (gcd) of two positive integers a and b is the greatest positive integer that divides both a and b. Two positive integers are relatively prime if their gcd is 1.)

Euler's product formula gives the value of $\phi(k)$ in terms of the prime factorization of k. For a prime p, let $\nu_p(k)$ be the highest power of p which divides k (so for example, $\nu_2(48) = 4$, $\nu_3(48) = 1$, and $\nu_5(48) = 0$). If k is a product of powers of prime factors, $k = \prod_{i=1}^{j} p_i^{\nu_{p_i}(k)}$ (where the product only includes primes p_i with $\nu_{p_i}(k) > 0$), then

$$\phi(k) = \prod_{i=1}^{j} \left[(p_i - 1) \left(p_i^{\nu_{p_i}(k)-1} \right) \right].$$

A recent edition of The American Mathematical Monthly (Li et al., *Positive Rational Numbers of the Form* $\phi(m^2)/\phi(n^2)$, 128(2), 2021) proved the following fact about totient quotients: for any pair of positive integers a, b there is a unique pair of positive integers m, n for which:

1. $\frac{a}{b} = \frac{\phi(m^2)}{\phi(n^2)}$;
2. if a prime p divides the product mn, then $\nu_p(m) \neq \nu_p(n)$;
3. $\gcd(m, n)$ is square-free: that is, for every prime p, $\gcd(m, n)$ is not divisible by p^2.

Conditions 2 and 3 guarantee that m and n are the unique smallest pair of positive integers satisfying condition 1.

You'd like to verify this claim numerically. Write a program which takes as input two integers a and b and outputs the corresponding pair m, n.

Input

The only line of input contains two space-separated integers a and b ($1 \leq a, b \leq 10\,000$). These two integers are guaranteed to be relatively prime. Additionally, a and b will be chosen so that output values m and n are less than 2^{63}.

Output

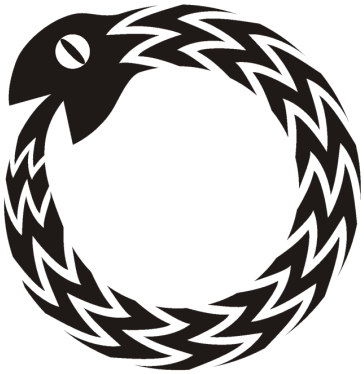
Print the two positive integers m and n satisfying all three of the conditions of The American Mathematical Monthly's theorem, separated by a space. It is guaranteed that $m, n < 2^{63}$.

Sample Input 1	Sample Output 1
9 13	18 13

Sample Input 2	Sample Output 2
19 47	13110 18612

Problem B
Circle of Leaf
Time Limit: 3 Seconds, Memory Limit: 2G

Your friend has given you a rooted tree: a connected graph with N nodes and $N-1$ edges. The nodes of the tree are numbered from 1 to N, with node 1 being the root of the tree and other nodes numbered arbitrarily.



Ouroboros from Wikimedia Commons

However, you recently learned about the Ouroboros, an ancient mythical snake that eats its own tail, signifying a cycle with no beginning and end. You dislike the fact that the tree you were given has a very clear beginning at the root, and clear ends at its leaves, so you decide to completely change the structure of this tree into a new graph which you have named an *Ouroboros Graph*.

To construct this Ouroboros Graph, you take the leaves of the tree (the nodes with no direct children) and draw special "leaf" edges that connect every leaf directly to the root. **If there is already an edge connecting a leaf to the root, you still add a duplicate edge.**

With this special graph structure, you can now create lots of different trees by removing some subset of edges, and in the spirit of Ouroboros, the leaves and roots can change depending on which subset of edges you remove. How many different trees can you make by removing a subset of edges from the Ouroboros Graph? Two trees are considered different if one tree has an edge that the other tree does not. (If both a regular and a "leaf" edge connect the same pair of nodes, then they are distinguishable from each other and are considered different edges.) Since the number of trees can be large, compute the answer modulo $998\,244\,353$.

Input

The first line of input contains a single integer N ($2 \leq N \leq 200\,000$), the number of nodes in the tree.

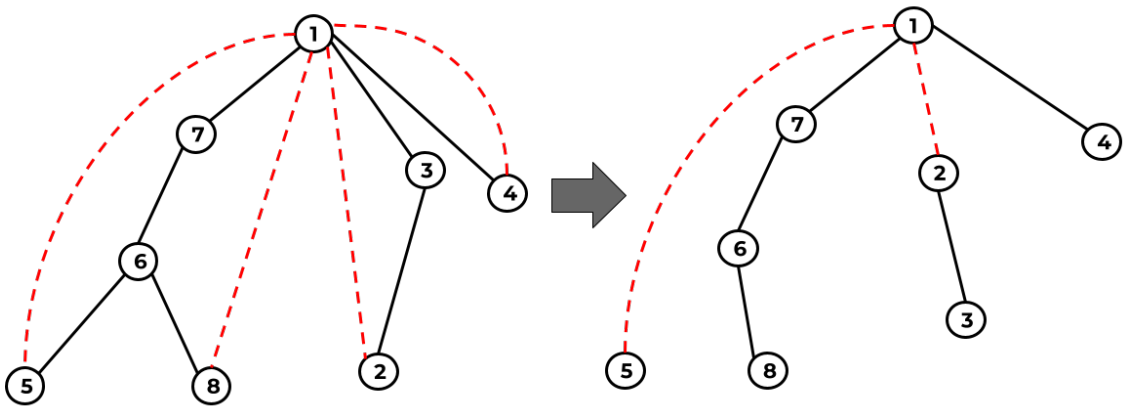
Each of the next $N-1$ lines contains two space separated integers a and b ($1 \leq a, b \leq N$) specifying that an edge exists between parent node a and child node b in the tree. The input graph is indeed guaranteed to be a tree: there is a unique path of edges between every pair of nodes in the graph.

Output

Print the number of different trees modulo $998\,244\,353$ that can be created by removing some subset of edges from the input tree's Ouroboros Graph.

Sample Explanation

In the diagram below, the left subfigure illustrates the Ouroboros Graph corresponding to Sample Input 1, with the original edges of the tree drawn in black and the "leaf" edges dashed in red. The tree on the right illustrates one of the 72 possible different trees that can be formed by deleting some subset of edges from the Ouroboros Graph: in this case, original edges 6–5 and 1–3 and "leaf" edges 1–8 and 1–4 were deleted.



Sample Input 1	Sample Output 1
8 1 3 3 2 1 4 1 7 7 6 6 5 6 8	72

Problem C
Entrapment
Time Limit: 5 Seconds, Memory Limit: 2G

Entrapment is an asymmetric two-player game that is played on a 3×3 square grid. The two players are called the Runner and the Trapper. The grid squares are numbered from 1 to 9 as depicted below:

1	2	3
4	5	6
7	8	9

Before starting the game, the players agree on the number of rounds that the game will last, and on the starting state of the game board. Up to 8 of the grid squares can be marked as *unavailable*. The players also choose who will be the Runner and who will be the Trapper. The Runner then secretly chooses a starting square from among those that are available (i.e., are not marked as unavailable) but does not tell the Trapper their choice.

Each of the game rounds consists of the following steps, in order:

1. The Trapper publicly chooses some subset of the available squares (the empty set is allowed) and asks the Runner, "Are you currently in any of these squares?"
2. The Runner answers truthfully whether or not they are in any of the chosen squares.
3. The Trapper publicly chooses exactly one available square. That square becomes unavailable for the rest of the game. (The Runner might currently reside in that square; if so, nothing special happens.)
4. The Runner secretly moves from their current square to an available orthogonally-adjacent square. If no such square exists, the Runner announces that they are trapped and the Trapper wins the game.

If the Runner has not been trapped by the end of the last round, they prove to the Trapper that they answered all questions truthfully by revealing their choice of starting square and the move that they made during each round. The Runner then wins the game.

Because the Runner's initial choice of square is secret, as are all of their subsequent moves, the Runner is allowed to "cheat" by not truly committing to a square. At the end of the game, if the Runner can produce a choice of starting square and subsequent moves that do not result in being trapped and are consistent with the answers to the Trapper's questions during each round, that is enough for the Runner to win the game.

Interaction

This is an interactive problem. Given the number of game rounds and the set of squares that are initially marked unavailable, determine whether the Runner or the Trapper would win assuming optimal play, and then prove it by playing as that role against the judge. The judge will obey all game rules, but may or may not play optimally.

Interaction starts by reading a line of 2 space-separated integers R and U ($1 \leq R \leq 9$, $0 \leq U \leq 8$, $R + U \leq 9$): the number of rounds in the game and the number of squares that are unavailable at the start of the game.

Next, if $U > 0$, read a line of U space-separated integers s ($1 \leq s \leq 9$): the labels of the squares that are unavailable at the start of the game. Please refer to the diagram above for how the squares in the grid are labeled. The U labels are guaranteed to be distinct.

Determine whether the Runner or Trapper would win the game with optimal play, given the starting board and number of game rounds. Print a line of output with the string `Runner` if the runner wins with optimal play, and the string `Trapper` otherwise. You will play as that role for the rest of the game; please see the appropriate section below for further instructions on how to interact with the judge in that role.

For the Runner, repeat the following steps R times:

- Read a line of input with a single integer N: the size of the subset of available squares that the Trapper has chosen to ask about. N is guaranteed to be between 0 and the number of available squares left on the board, inclusive.
- If $N > 0$, read a line of N space-separated integers ℓ ($1 \leq \ell \leq 9$) listing the labels of the squares in the Trapper's chosen subset. The labels are guaranteed to be distinct and all of the chosen squares are guaranteed to be available.
- Print a line of output containing either the string `Yes` or the string `No`. The former informs the trapper that you are currently in one of the chosen squares; the latter informs the trapper that you are not.
- Read a line with a single integer i ($1 \leq i \leq 9$), the label of the square that the Trapper marks as unavailable. It is guaranteed that square i is a formerly-available square.
- Print a line with the string `Free` to inform the Trapper that you have secretly moved to an orthogonally-adjacent available square and are ready to proceed to the next round. If there are no orthogonally-adjacent squares available, you must print `Trapped` instead and exit; your submission will be judged incorrect for having failed to elude the Trapper.

After you have played R rounds of the game according to the protocol above, print a line with $R + 1$ space-separated integers. The first integer is the label of your chosen starting square; each of the next R integers are the labels of the squares onto which you moved at the end of each of the R rounds. Your moves must be legal and must be consistent with the answers you gave to the Trapper's queries during each round of play. After printing this line, your program must exit.

For the Trapper, repeat the following steps R times:

- Print a line with a single integer N: the size of the subset of available squares that you would like to ask the Runner about.
- If $N > 0$, print a line of N space-separated integers listing the available squares to ask the Runner about. You may list the labels in any order, but the labels must be distinct and must refer to available squares.
- Read a line of input containing a single string: `Yes` if the Runner is in one of your chosen squares, or `No` otherwise.
- Print a line with a single integer i: the square that you are marking unavailable. The label i must be a valid currently-available square.
- Read a line with a single string: `Free` if the Runner has moved to an available square, or `Trapped` if they were unable to do so. After reading the word `Trapped`, you have won the game, and your program must exit. If you read the word `Free` at the end of the Rth round, your program must also exit, though your submission will be judged incorrect since you have failed to trap the Runner.

The judge is guaranteed to answer all questions truthfully.

Notes

Do not forget to flush the output stream after each line that you print and to cleanly exit after the interaction is done. Please also make sure that you follow the above interaction protocol exactly regarding what information to print on which line of output: for example, if the protocol requires you to print a list of space-separated integers on a single line, the judge **will not** accept each integer on its own line.

If the judge receives invalid or unexpected input, it will print -1 and then immediately exit. Your program must detect this error report and cleanly exit in order to receive a Wrong Answer verdict. If your program blocks waiting for further interaction from the judge, or tries to interpret the -1 as a game move, you may receive a different rejected verdict (such as Time Limit Exceeded or Runtime Error) instead of Wrong Answer.

You have been provided with a command-line tool for local testing. The tool has comments at the top to explain its use.

Read	Sample Interaction 1	Write
3 6 1 2 3 7 8 9		
	Trapper 2 4 5	
Yes		
	5	
Free		
	0	
No		
	6	
Trapped		

Read	Sample Interaction 2	Write
2 0		
	Runner	
7 3 1 2 8 9 4 5		
	Yes	
5		
	Free	
4 4 6 7 8		
	Yes	
7		
	Free 5 4 1	

Problem D
Geometry Rush
Time Limit: 1 Second, Memory Limit: 2G

You are playing the summer's hottest rhythm-based action platformer—Geometry Rush! The game is played on a 2D plane. Your character begins at $(0, 0)$ and every second must move at a 45-degree angle either up-right or down-right, which takes your character from position (x, y) to $(x+1, y+1)$ or $(x + 1, y - 1)$ respectively. You can change which direction you move every second, but not in between moves. There are obstacles protruding from the floor and ceiling that you must dodge. You win the game if, after w seconds, you reach the line $x = w$ without having touched any obstacles on the way.

The play area extends vertically from $y = -h$ to $y = h$. Obstacles are two polygonal curves: one curve starts at $(0, h)$ and ends at (w, h) and represents a ceiling of varying height. The x values of the vertices of this curve are non-decreasing, and the y values lie between $-h$ and h inclusive. A second polygonal curve starts at $(0, -h)$ and ends at $(w, -h)$ and represents the floor. Its vertices satisfy similar constraints.

Your character is a point of negligible extent: you can move from position (x, y) to $(x + 1, y \pm 1)$ so long as the line segment between your start and end position does not intersect either obstacle. (Exactly touching either polygonal curve counts as intersecting an obstacle, and loses the game.)

You have played *a lot* of games of Geometry Rush. To keep the game interesting, you have started to set challenges for yourself. For example: you win the game no matter where you cross the $x = w$ goal line. But for what maximum value of y can you win the game by crossing at (w, y) without touching any obstacles on the way? For what minimum value? Compute these numbers.

Input

The first line of the input contains four space-separated integers n, m, w, and h. The first two integers ($3 \le n, m \le 10^5$) are the number of vertices in the ceiling and floor polygonal curves, respectively. The second two integers ($3 \le w, h \le 10^5$) are the width and height of the play area, as described above.

The next n lines each contain two space-separated integers x and y ($0 \le x \le w$; $-h \le y \le h$): the coordinates of the vertices of the ceiling polygonal curve, in order from left to right. It is guaranteed that the first vertex is at $(0, h)$ and the last vertex is at (w, h).

The next m lines each contain two space-separated integers x and y ($0 \le x \le w$; $-h \le y \le h$): the coordinates of the vertices of the floor polygonal curve, in order from left to right. It is guaranteed that the first vertex is at $(0, -h)$ and the last vertex is at $(w, -h)$.

For both polygonal curves: the x coordinates are non-decreasing, all vertices are distinct, and the curve does not self-intersect. Neither curve intersects $(0, 0)$. (The floor and ceiling curves might intersect each other, in which case the game is unwinnable.)

Output

If it is impossible to win the game, print `impossible`. Otherwise, print two space-separated integers: the minimum and maximum y values that the player could reach at $x = w$ without losing the game by touching an obstacle along the way.

Sample Input 1

```
4 4 5 5
0 5
0 2
5 2
5 5
0 -5
0 -2
5 -2
5 -5
```

Sample Output 1

```
-1 1
```

Sample Input 2

```
4 4 6 5
0 5
0 2
6 2
6 5
0 -5
0 -2
6 -2
6 -5
```

Sample Output 2

```
0 0
```

Sample Input 3

```
3 3 7 5
0 5
5 -1
7 5
0 -5
2 1
7 -5
```

Sample Output 3

```
impossible
```

Sample Input 4

```
4 3 5 5
0 5
0 2
5 2
5 5
0 -5
3 -1
5 -5
```

Sample Output 4

```
-1 1
```

This page is intentionally left blank.

Problem E
Humans vs AI
Time Limit: 5 Seconds, Memory Limit: 2G

In the world of rising AI, James is scared of losing his job. So, when his boss asks him to evaluate a new AI model to see how well it performs compared to humans, he wants to make it look as bad as possible.

To test the AI, James conducts a sequence of N trials where a human and an AI are given the same task and then scored based on their performance on the task. He is then going to send the results of some non-empty contiguous subsequence of these trials to his boss and quietly delete the rest.

Let a_i and h_i be the performance of the AI and human on trial i, respectively. James's boss evaluates the AI on a sequence of trials by calculating two total scores: one for the humans, and one for the AI. Both scores are initially 0. For each trial i where $h_i \geq a_i$, the boss awards the humans $h_i - a_i$ points. For each trial where $h_i < a_i$, the AI earns $a_i - h_i$ points. If the humans' total score is greater than or equal to the AI's total score times some constant k (to account for humans needing food, water, and a desk), James's boss declares that the humans outperform the AI.

James plans to send his chosen subsequence of test results through email to his boss. There is, however, one complication: since AI is already all-knowing and all-pervasive, it intercepts this email and may swap the value of h_i and a_i for one trial i of its choice. (It doesn't want to swap more than one trial result—James might notice!)

Count how many non-empty contiguous subsequences of trial results James could send his boss with the guarantee that humans will be declared to outperform the AI, even if the AI swaps the result of up to one trial.

Input

The first line of input contains two space-separate integers: N ($1 \leq N \leq 2 \cdot 10^5$), the total number of trials James conducted, and k ($1 \leq k \leq 100$), the multiplier James's boss will apply to the AI's total score to determine whether humans outperform AI.

The second line contains N space-separated integers $h_1, h_2, \ldots, h_N$ ($0 \leq h_i \leq 10^6$), the performance of the humans on each of the N trials.

The third line contains N space-separated integers $a_1, a_2, \ldots, a_N$ ($0 \leq a_i \leq 10^6$), the performance of the AI on the N trials.

Output

Print the number of non-empty contiguous trial subsequences for which James's boss would declare that humans outperform AI, even if the AI swaps the result of up to one trial.

Sample Input 1	Sample Output 1
10 2 3 5 7 6 8 6 4 5 2 6 2 4 6 5 4 3 3 6 3 4	4

Sample Input 2	Sample Output 2
7 1 4 3 2 1 7 6 5 4 2 3 1 7 6 5	11

Problem F
Mob Grinder
Time Limit: 5 Seconds, Memory Limit: 2G

In a certain popular sandbox video game, one can build a structure called a *mob grinder*. A mob grinder consists of an $N \times M$ rectangular grid of tiles. Monsters, also known as "mobs," appear continuously at random places on the grid. The goal of a mob grinder is to move all of the monsters to the top-right tile in the grid, no matter where they originally appear. To accomplish this goal, each tile (except for the top-right tile) has a conveyor belt on it with a specified direction (up, right, down, or left). A monster on a conveyor belt gets moved to the orthogonally adjacent tile in the direction specified by the conveyor belt orientation.

Your job is to place a conveyor belt on each tile (other than the top-right corner) so that no matter where a monster appears on the grid, it will get moved to the top-right corner after a finite amount of time, without ever leaving the bounds of the grid. However, there is a limit on how many conveyor belts you can use of each orientation: your final design must have exactly U conveyor belts going up, R going right, D going down, and L going left.

You are asked to design multiple mob grinders, each with a specification of how many conveyor belts of each type you are allowed to use. Design a valid mob grinder that meets each specification, if possible.

Input

The first line of input contains an integer T ($1 \leq T \leq 10^5$): the number of mob grinders you need to design.

Each of the next T lines of input contains six space-separated integers that describe one mob grinder specification. The first two integers, N and M, ($1 \leq N, M$ and $N \cdot M \leq 10^5$) are the number of rows and columns in the grid, respectively. The last four, U, R, D, L ($0 \leq U, R, D, L$ and $U + R + D + L = (N \cdot M) - 1$), are the number of times you must use each conveyor belt orientation in your design.

It is guaranteed that the sum of $N \cdot M$ over all T mob grinders does not exceed 10^5.

Output

Print T mob grinder designs, one for each specification. Separate consecutive designs with a single empty line.

If it is impossible to construct a valid mob grinder respecting the given constraints for the given specification, print `impossible`. Otherwise, print an $N \times M$ grid of ASCII characters. The top-right tile must be a `*`. Every other character in the grid must be either `U`, `R`, `D`, or `L`, representing the orientation of the conveyor belt on that grid tile.

This problem is whitespace-sensitive. You *must* separate each mob grinder design with exactly one empty line (containing just a newline character). You *must not* print an empty line, or any other extraneous output, after the last mob grinder design (though the last line of output must be terminated with a newline). Please see the Sample Output for examples of how to correctly format your mob grinder designs.

Sample Input 1

```
2
4 3 5 3 1 2
1 2 0 1 0 0
```

Sample Output 1

```
RR*
URU
UDU
ULL

R*
```

Sample Input 2

```
3
3 3 0 0 0 8
2 2 0 2 0 1
1 1 0 0 0 0
```

Sample Output 2

```
impossible

impossible

*
```

Problem G
Most Scenic Cycle
Time Limit: 7 Seconds, Memory Limit: 2G

The government of the Independent Country of Problem Creators (ICPC) finally saved enough money to construct high speed rail infrastructure. The new rail system has V stations and E bidirectional direct railway lines that each connect two stations together. The head of ICPC Rail Infrastructure Planning, Skib E. Dee, has seen enough programming problems about tree-topology transportation networks in other countries to know that such a network would be a recipe for disaster: a single broken railway line would split the network into disconnected pieces and disrupt travel for days. Instead, Dee decided that the ICPC rail network will be **robustly connected**: every pair of stations s_1, s_2 must be connected by at least two paths which do not share any direct railway lines, and do not share any railway stations other than s_1 and s_2 themselves.

Of course, ICPC cannot afford to build too many redundant railway lines. To balance efficiency and resiliency, Dee has also designed the network to be **regionally connected**. A cycle is a non-empty path from a station to itself which doesn't repeat any railway station (apart from the first station, which must repeat exactly once as the last station of the cycle). In order for the network to be regionally connected, there must exist a set $\mathcal{F}$ of $E - V + 1$ *regional cycles* satisfying three properties:

- every direct railway line in the transportation network belongs to at least one regional cycle;
- if two regional cycles share any direct railway lines, then all railway lines and stations shared by those cycles lie along a connected path;
- for each subset f of $\mathcal{F}$, at most $|f| - 1$ pairs of regional cycles in f share any direct railway lines.

To promote the new high speed rail, Dee needs to create a timelapse video of a train travelling around some cycle in the railway network. Each direct railway line has a (possibly negative) scenic value representing how nice the view out the train window is along that line. Dee wants to send the train around whichever cycle maximizes the sum of scenic values of the direct railway lines on the cycle—compute this maximum possible sum. (The most scenic cycle that Dee is looking for does **not** have to be a regional cycle.) In order to ensure this cycle is not boring, it must traverse at least two direct railway lines, and must not repeat any railway station (apart from the first station, which must repeat exactly once as the last station of the cycle).

Input

The first line of input contains two space-separated integers V ($2 \leq V \leq 2 \cdot 10^5$) and E ($V \leq E \leq 4 \cdot 10^5$), the number of stations and direct railway lines in the rail network, respectively.

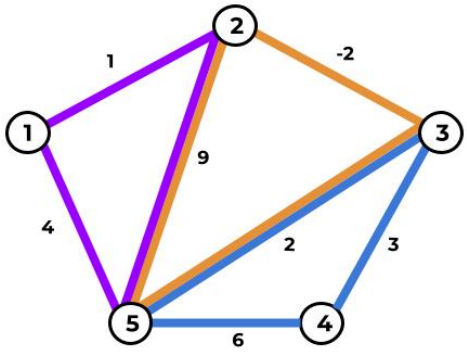
The next E lines of input describe the direct railway lines. Each line contains three space-separated integers a, b, and s ($1 \leq a, b \leq V$; $-10^9 \leq s \leq 10^9$), signifying that a direct railway line exists between stations a and b with scenic value s. No direct railway line connects a station to itself, but **multiple direct railway lines might exist between the same two stations**. It is guaranteed that the input graph will be both *robustly connected* and *regionally connected*.

Output

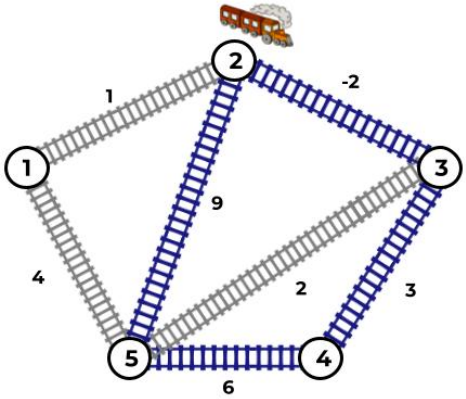
Print the sum of scenic values around the cycle in the railway network that maximizes this sum.

Sample Explanation

For the railway network in Sample Input 2, one possible choice for the regional cycles in $\mathcal{F}$ are $1 \to 2 \to 5 \to 1$, $2 \to 5 \to 3 \to 2$, and $3 \to 4 \to 5 \to 3$ (pictured on the left). The most scenic cycle (pictured on the right, in blue) has a scenic value sum of $9 + 6 + 3 - 2 = 16$.



Sample 2 Regional Cycles



Sample 2 Most Scenic Cycle

Sample Input 1

```
6 9
1 2 9
2 3 9
3 4 9
3 4 -9
4 1 9
1 5 1
5 6 1
6 2 1
3 4 8
```

Sample Output 1

```
36
```

Sample Input 2

```
5 7
1 2 1
2 3 -2
3 4 3
4 5 6
5 1 4
5 3 2
2 5 9
```

Sample Output 2

```
16
```

This page is intentionally left blank.

Problem H
Ornaments on a Tree
Time Limit: 4 Seconds, Memory Limit: 2G

You're helping your friend decorate their Christmas tree! Funnily enough, the places to put your ornaments on their Christmas tree can be represented by a (graph-theoretic) tree with nodes labeled 1 to N, with node 1 being the root of the tree and other nodes numbered arbitrarily. You have an infinite supply of ornaments of every non-negative integer weight (including 0), and you must place exactly one ornament on each node of the tree.

However, your friend has some restrictions on how they want their tree decorated. First, they have strong opinions about which ornament must go on some of the tree nodes; you are only allowed to choose decorations on the other nodes. Second, each region of their tree can support only so much weight: if the sum of the weights of the ornaments on a node and all of its immediate children exceeds a constant K, the whole tree will come crashing down!

Your friend wants to know the largest possible total weight of ornaments on their tree, given the above restrictions. Can you help them find out?

Input

The first line of input has two space-separated integers N and K ($1 \leq N \leq 2 \cdot 10^5, 0 \leq K \leq 10^9$), the number of nodes in the tree and the weight constant, respectively.

The next line contains N space-separated integers. The i^{th} integer (starting at $i = 1$) is either -1 or a non-negative integer. If it is -1, you are free to choose any ornament for node i. If it is a non-negative integer w_i ($0 \leq w_i \leq 10^9$), your friend insists you place an ornament with weight w_i on node i.

The next $N - 1$ lines each contain two space-separated integers a and b ($1 \leq a, b \leq N$), indicating that nodes a and b are connected by an edge. The input graph is guaranteed to be a tree: there is a unique path of edges between every pair of nodes in the graph.

Output

If it is impossible to place ornaments on the tree in a way that satisfies all of the constraints described above, print -1. Otherwise, print the maximum possible total weight of the ornaments on the tree, subject to the constraints.

Sample Input 1

```
5 10
-1 2 3 -1 -1
1 2
1 3
2 4
2 5
```

Sample Output 1

```
18
```

Sample Input 2

```
1 5
-1
```

Sample Output 2

```
5
```

Sample Input 3

```
2 5
5 5
1 2
```

Sample Output 3

```
-1
```

Problem I
Polygon Partition
Time Limit: 3 Seconds, Memory Limit: 2G

A simple polygon is a polygon that is not self-intersecting and does not contain any holes. You are given the N vertices of a simple polygon, v_1, v_2, ..., v_N, where $v_i = (x_i, y_i)$, and x_i and y_i are the x-coordinate and y-coordinate of the i^{th} vertex, respectively. The vertices are distinct and given in counterclockwise order (so there is an edge between each pair of consecutive vertices; there is also an edge from v_N back to v_1).

The polygon's boundary does not pass through any *lattice points* (a lattice point is a point where both coordinates are integers). In addition, none of the x_i or y_i values are exactly an integer.

A *semi-integer point* is a point where exactly one of its coordinates is an integer. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ be all of the semi-integer points that lie on the boundary of the polygon. For each semi-integer point p_i in $\mathcal{P}$, let n_i be the floor of the non-integer coordinate of p_i. For a subset $\mathcal{S}$ of $\mathcal{P}$, let $\sigma(\mathcal{S})$ be the sum of the n_i of the points in $\mathcal{S}$ (with $\sigma(\emptyset) = 0$). Does there exist a partition of $\mathcal{P}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ so that the $\sigma(\mathcal{S}_1) = \sigma(\mathcal{S}_2)$?

(Two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ are a partition of $\mathcal{P}$ if $\mathcal{P} = \mathcal{S}_1 \cup \mathcal{S}_2$ and $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$. There are no other restrictions on $\mathcal{S}_1$ and $\mathcal{S}_2$ so long as these two conditions hold and $\sigma(\mathcal{S}_1) = \sigma(\mathcal{S}_2)$. In particular, empty sets are allowed, and the semi-integer points in each set *do not* have to be contiguous around the polygon boundary.)

Input

The first line of input contains one integer N ($3 \leq N \leq 500$), the number of vertices of the polygon.

Each of the next N lines contains two space-separated real numbers x_i and y_i ($-500 < x_i, y_i < 500$): the coordinates of the polygon vertices, in counterclockwise order. Each coordinate will have exactly 6 digits after the decimal point and will not be exactly an integer.

It is guaranteed that the polygon does not self-intersect, that the vertices are distinct, and that the polygon boundary does not pass through any lattice points.

Output

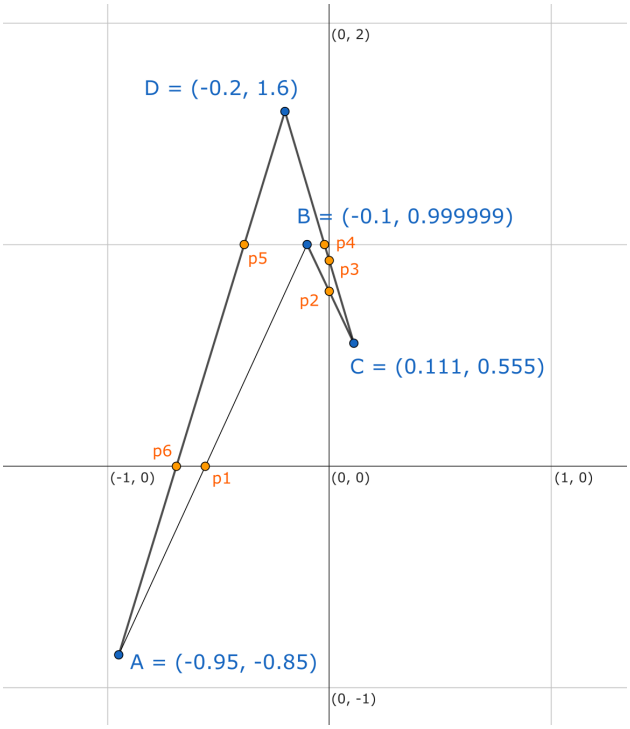
If there is no solution, print -1 and no further output.

Otherwise, print a single integer M on its own line: the number of semi-integer points in one of the two subsets in a valid partition of $\mathcal{P}$. On the next M lines of output, print the values n_i for the points in that subset, one per line.

If there are multiple valid partitions, you may choose any of them. You may print either of its two subsets, and you may list the subset's n_i values in any order.

Sample Explanation

Sample Input 1 is shown in the image below:



The points of the vertices are labeled A, B, C, D. The semi-integer points are marked in orange and labeled p_i going counterclockwise around the perimeter starting from A. The values n_i of the semi-integer points are, in the same order, $-1, 0, 0, -1, -1, -1$. Any subset of those values that sum to -2 would be accepted as correct. Sample Output 1 shows one possible correct answer.

The boundary of the polygon in Sample Input 2 does not intersect any semi-integer points, so $\mathcal{P}$ is empty, and it can be partitioned into two empty sets each with n_i sum of zero.

Sample Input 1

```
4
-0.950000 -0.850000
-0.100000 0.999999
0.111000 0.555000
-0.200000 1.600000
```

Sample Output 1

```
3
0
-1
-1
```

Sample Input 2

```
3
0.500000 0.700000
0.100000 0.200000
0.800000 0.900000
```

Sample Output 2

```
0
```

Sample Input 3

```
4
-360.000001 -24.000001
-359.999999 -24.000001
-359.999999 -23.999999
-360.000001 -23.999999
```

Sample Output 3

```
2
-25
-360
```

This page is intentionally left blank.

Problem J
Popping Balloons
Time Limit: 15 Seconds, Memory Limit: 2G

The ICPC logo has three colors: blue, yellow, and red. The NAC volunteers have just inflated a huge number of balloons in these colors and arranged them in a line. They next need to sort the balloons by color before they can give them out to contestants.

Unfortunately, due to the Orlando heat, the balloons begin to randomly pop: each second, a random remaining balloon pops (and the volunteers remove the debris from the line). This isn't all bad: maybe if the NAC volunteers wait long enough, the balloons will become sorted by chance? Compute the expected number of seconds until the first time that all blue balloons come before all yellow and red balloons, and all yellow balloons come before all red balloons. (These conditions are satisfied even if they are vacuously true: for example, if there are no blue balloons at all remaining, then it is true that all blue balloons come before all yellow and red balloons.)

Input

The input has one line: a string s ($1 \le |s| \le 2 \cdot 10^5$) where each character is one of `B`, `Y`, or `R` representing blue, yellow, and red respectively —the colors of the initial balloons in the line.

Output

Print the expected number of seconds that elapse before the first time that all blue balloons come before all yellow and red balloons, and all yellow balloons come before all red balloons. Since this number might not be an integer, print it modulo $998\,244\,353$.

Formally, let $p = 998\,244\,353$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{a}{b}$, where a and b are non-negative integers and $b \not\equiv 0 \pmod{p}$. Print the integer x with $0 \le x < p$ and $x \equiv a \cdot b^{-1} \bmod p$.

Sample Explanation

In Sample Input 1, the expected time until the balloon colors first become sorted in the correct order is $\frac{17}{6} = 2 \cdot \frac{1}{6} + 3 \cdot \frac{5}{6}$ seconds: the only way for the balloon colors to be sorted correctly after 2 seconds is if the first two balloons to pop are the yellow and red balloon (in either order). The probability that these balloons pop before either blue balloon is $\frac{1}{6}$. Otherwise (with probability $\frac{5}{6}$) the balloon colors will automatically be sorted after 3 seconds, when there is only one balloon left. Since $6^{-1} \equiv 166\,374\,059 \pmod{p}$, the answer is $17 \cdot 166\,374\,059 \equiv 831\,870\,297 \pmod{p}$.

Sample Input 1	Sample Output 1
RYBB	831870297

Sample Input 2	Sample Output 2
YRBBR	598946615

Problem K
SLA Tomography
Time Limit: 1 Second, Memory Limit: 2G

Stereolithography (SLA) is a 3D printing technique for hardening liquid material into a solid object one layer at a time using a laser. In this problem, we will consider a 2D simplification of SLA where the design of the object being printed can be represented as a rectangular grid of '#' and '.' characters, where '#' represents a grid cell occupied by the object and '.' is empty space. For example, here is a 4×8 design:

```
..#.....
..#..#..
#.#.##..
#.#####.
```

A design does not have to consist of a single connected piece, but except for '#' cells on the bottom row of the design, **each '#' cell must be supported by another '#' cell directly below it**.

Printing an object using SLA proceeds layer-by-layer, starting from the bottom row. First, all cells in the row are flooded with a liquid photosensitive resin. Then a laser sweeps over the row, hardening the resin in all '#' cells into a solid and skipping all '.' cells. Then, leftover liquid to the left of the leftmost '#' and to the right of the rightmost '#' drains away. Other liquid remains trapped. (If there are no '#' cells in the row—which can only happen for rows near the top of the design, after the object has been fully printed—all liquid drains away from the row.) This process then repeats for each subsequent row. For the design above, after printing is complete, resin remains trapped in all of the cells marked with a '~' character below:

```
..#.....
..#~~#..
#~#~##..
#~#####.
```

While manually suctioning the leftover resin from the object, you start to wonder: how much of the original design can be recovered from knowing only how much liquid resin is left over in each row of the design after printing? For the above design, the amount of leftover resin in each row (starting from the top of the design) is 0, 2, 2, 1. Other designs also have the same leftover-resin fingerprint; for example:

```
....
#..#
#..#
#.##
```

Given a list of how many cells of liquid resin are left over in each row (starting from the top row), print the width of the narrowest object design whose rows would contain those amounts of liquid resin after printing. If no such design exists, print `impossible`.

Input

The first line of input contains a single integer $N (1 \leq N \leq 10^5)$, the number of rows in the object design. N lines follow, each containing a single integer x $(0 \leq x \leq 10^9)$, the number of cells of leftover liquid resin in each row of the desired object design (in order from top to bottom).

At least one row will have at least one leftover cell of liquid resin.

Output

Print the width (number of columns) in the narrowest object design whose number of leftover liquid resin cells in each row matches the input. ("Narrowest" means having the smallest possible number of columns). If no such design exists, print `impossible` instead.

Sample Explanation

Sample Input 1 corresponds to the example above. One narrowest-possible design for Sample Input 2 is:

```
#....#.....
######.....
######....#
```

Sample Input 1	Sample Output 1
4 0 2 2 1	4

Sample Input 2	Sample Output 2
3 4 0 4	11

This page is intentionally left blank.

Problem L
Solar Farm
Time Limit: 1 Second, Memory Limit: 2G

You are building a new solar farm. The area in which you are allowed to build is a circular field of radius r, and the solar panels each take up a rectangular space of size $w \times h$. You must place all the panels in the same orientation of your choice in a single rectangular array (so that all of the panels combined exactly form a single rectangle). What is the maximum number of panels that you can fit in this farm?

Input

The first line of input contains a single integer T ($1 \leq T \leq 1\,000$). This is the number of test cases.

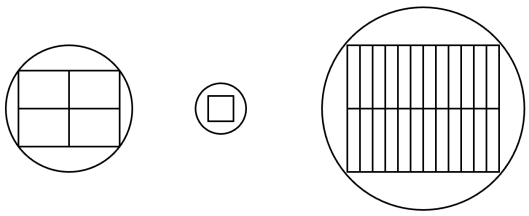
The next T lines of input each represent one test case and consist of three space-separated integers r, w, and h ($1 \leq r, w, h \leq 10^9$): the radius of the field, the width of each solar panel, and the height of each solar panel, respectively.

Output

For each test case, print a line with a single integer: the maximum number of solar panels that can be placed in a solar farm within the circular field.

Sample Explanation

The diagram below illustrates one optimal layout of solar panels for each of the three test cases in Sample Input 1 (from left to right).



Sample Input 1	Sample Output 1
3 5 4 3 2 2 2 8 1 5	4 1 24

Sample Input 2	Sample Output 2
2 500000003 1 600000010 511374200 637192506 100000000	799999999 7

Problem M
This Is Sparta!
Time Limit: 1 Second, Memory Limit: 2G

King Primonidas is putting together a tournament to find the strongest gladiator in all the lands. In total N gladiators have made their way to the Coliseum to bring back honor and glory to their hometowns. Each gladiator starts with a certain amount of vitality. Vitality is similar to health points in that it reflects the amount of damage a gladiator can take, but it also represents the amount of damage they can deal (since energy is needed for strong blows).

The tournament consists of K rounds. Every round, the king arranges the gladiators in a line from least to most remaining vitality, breaking ties randomly. King Primonidas believes that the true strength of a gladiator lies in their ability to take a beating, so he orders the first gladiator (the one with the lowest remaining vitality) to deal their strongest blow to the second gladiator. This blow subtracts the first gladiator's vitality from the second gladiator's vitality. After the second gladiator takes the hit, they deal their strongest blow (using their new, lowered vitality) to the third gladiator, and so on. This process repeats until the second-to-last gladiator deals their strongest blow to the last gladiator (who doesn't get to attack anybody).

Notice that in the above process, a gladiator's vitality can never go below zero. (A gladiator with zero vitality deals a feeble blow of no damage to the next gladiator.)

Print the vitality of each gladiator after K tournament rounds, in order from the first to the last gladiator in line.

Input

The first line of input contains two space-separated integers N ($2 \leq N \leq 10^5$) and K ($1 \leq K \leq 10^{18}$): the number of gladiators and the number of rounds in the tournament.

The next line of input contains N space-separated integers $v_1\ v_2\ \ldots v_N$ ($0 \leq v_i \leq 10^{18}$): the starting vitality of the gladiators.

Output

Print N space-separated integers: the vitality of the gladiators at the end of the Kth round of the tournament, in the order that the gladiators currently stand in line.

Sample Input 1	Sample Output 1
6 3 21 28 24 23 1 12	1 1 3 6 3 10

Sample Input 2	Sample Output 2
3 1000 8 2 10	0 0 2