Problem A
Acorn Quarrels
Time Limit: 1 Second, Memory Limit: 2G


Paul Danese, CC BY-SA 4.0, via Wikimedia Commons

There are N squirrels storing acorns in a tree. The (botanical) oak tree is also a graph-theoretic tree: a connected graph with N vertices labeled from 1 to N and $N-1$ undirected edges. Each squirrel sits at a different vertex of the tree and two squirrels are *neighbors* if their vertices share an edge.

In ascending order of vertex label starting with the squirrel at vertex 1, each squirrel steals one acorn from the neighboring squirrel that currently has the most acorns. If multiple neighbors are tied for having the most acorns, the squirrel steals one acorn from each of them!

To limit the fallout of these shenanigans, you want to distribute acorns to the squirrels so that each squirrel begins with at least N acorns (so that no squirrel runs out of acorns due to thefts) and ends with the same number of acorns after all N squirrels are done stealing from each other as they had originally. It can be shown that such a distribution exists where every squirrel begins with at most $2N-1$ acorns. Find any distribution satisfying these conditions.

Input

The first line of input contains an integer N ($2 \leq N \leq 10^5$), the number of vertices (and squirrels).

Each of the following $N-1$ lines contains two space-separated integers u and v ($1 \leq u, v \leq N, u \neq v$), indicating that an edge exists between vertices u and v. There is at most one edge between any pair of vertices, and the edges form a tree.

Output

Print N space-separated integers $d_1, d_2, \ldots, d_N$ satisfying $N \leq d_i \leq 2N-1$, where d_i is the number of acorns you would like to distribute to the squirrel at vertex i. Your solution will be accepted if every squirrel ends with the same number of acorns as they started with after all N thefts have occurred. It can be proved that such a distribution of acorns always exists.

Sample Input 1

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

Sample Output 1

```
6 5 7 7 9
```

Sample Input 2

```
8
5 4
3 7
8 4
4 7
5 2
1 3
6 4
```

Sample Output 2

```
14 15 10 9 13 14 14 14
```

Problem B
Boss Rush
Time Limit: 5 Seconds, Memory Limit: 2G

Franklin is playing the latest trendy timed-action video game and is facing a boss rush: a trying gauntlet where he must defeat N monsters (the bosses) to survive. His only ability, *parry*, is extremely powerful but very difficult to use.

Each boss attacks Franklin once every d seconds; however, the bosses have their own *starting delay* before they begin their sequence of attacks, so that the boss attacks are staggered. More specifically, if f_i is the starting delay of the i^{th} boss, then that boss will attack at seconds

$$f_i,\ f_i + d,\ f_i + 2d,\ \ldots$$

To defend himself, Franklin can parry an attack on the exact second it happens, instantly defeating the boss and ending all of its subsequent attacks. Franklin can only parry one attack at a time: if multiple bosses attack him simultaneously, he can parry at most one of those attacks.

Moreover, after parrying an attack Franklin becomes winded and cannot parry again during the next w seconds. Formally, if Franklin parries an attack at second t, the earliest moment that Franklin could parry another attack is at second $t + w$.

Franklin has plenty of health and is unconcerned about the attacks of the bosses against him, but he'd like to end the fight as quickly as possible. Compute the minimum amount of time it would take Franklin to defeat all N bosses if he parries optimally.

Input

The first line of input contains three space-separated integers N ($1 \le N \le 3 \cdot 10^5$), w ($1 \le w \le 10^9$), and d ($1 \le d \le 10^9$), where N is the number of bosses, w is the number of seconds Franklin must wait after parrying before parrying again, and d is the number of seconds between two attacks by the same boss.

The next line of input contains N space-separated integers f_i ($0 \le f_i < d$), the starting delay of each boss in seconds.

Output

Print the number of seconds it takes Franklin to defeat all N bosses if he parries optimally.

Explanation of Sample Input 1

The first boss attacks Franklin at second 2; Franklin could parry the attack but chooses to bide his time and instead parry the attack of the second boss at second 3. He is now winded and cannot parry again before second 7.

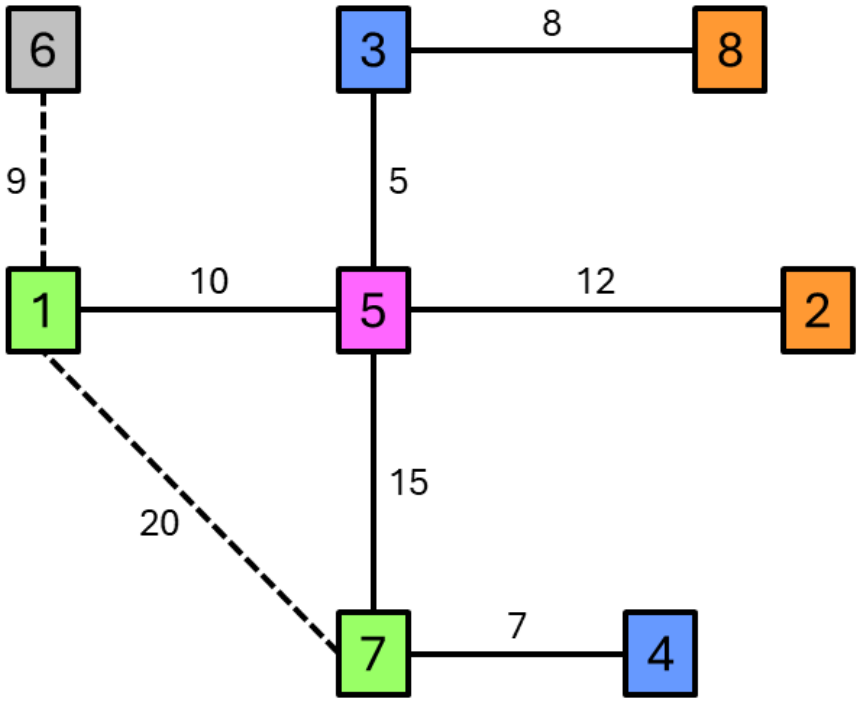
The third boss attacks Franklin at second 8 and Franklin parries the attack. He is winded again and cannot parry until second 12: just in time to parry the first boss's second attack, which ends the fight.

Sample Input 1	Sample Output 1
3 4 10 2 3 8	12

Sample Input 2	Sample Output 2
3 4 10 2 3 9	13

Problem C
Cable Pruning
Time Limit: 2 Seconds, Memory Limit: 2G

You are helping to remodel a data center to make room for more GPUs. Over the years, the data center has become cluttered with superfluous network cable, and you have been asked to clean up the mess and reclaim as much unused cable as possible.



An illustration of Sample Input 1. Servers in the same coupled pair are the same color. Cables to remove are indicated with a dashed line.

The data center has N servers and N network cables that link one server to another. Each cable has a length in feet. Traffic flows through network cables in both directions and the data center is initially connected: for every pair of servers (u, v) there exists a path of network cables from u to v (possibly passing through intermediate servers). You've audited the data center network traffic to discover that only K *coupled pairs* of servers need to communicate with each other. (Note that some servers might not be part of any coupled pair, or might be part of two or more coupled pairs.)

You now need to remove as much total length of cable as possible from the data center while keeping all coupled pairs connected to each other: for each such pair of servers (a, b) there must exist a path from a to b passing through original network cables that you've kept in place.

Find the minimum total length of cable that must be kept in place to satisfy this constraint.

Input

The first line of input contains two space-separated integers N ($3 \leq N \leq 10^5$) and K ($1 \leq K \leq 10^5$), the number of servers in the data center and number of coupled pairs of servers that you've discovered.

The next N lines describe the network cables originally in the data center. Each line contains three space-separated integers u_i, v_i ($1 \leq u_i, v_i \leq N, u_i \neq v_i$), and w_i ($1 \leq w_i \leq 10^9$), specifying that a cable connects server u_i to server v_i and has length w_i feet. There is at most one network cable connecting a pair of servers and the graph of servers and cables is connected.

Each of the next K lines contains two space-separated integers a_j and b_j ($1 \leq a_j, b_j \leq N, a_j \neq b_j$) and describe a coupled pair of servers. Each coupled pair must remain connected by a path after you are done removing cable. All coupled pairs are distinct; (a, b) and (b, a) are considered the same and won't both be listed as coupled pairs.

Output

Print a single integer: the minimum total length of network cable (in feet) that must remain in place in order for all K coupled server pairs to stay connected to each other.

Sample Input 1

```
8 3
5 3 5
1 7 20
3 8 8
7 5 15
5 2 12
1 6 9
5 1 10
7 4 7
3 4
8 2
1 7
```

Sample Output 1

```
57
```

Sample Input 2

```
5 1
1 3 3
4 2 4
3 4 2
5 2 2
4 1 6
2 1
```

Sample Output 2

```
9
```

Problem D
Draw Your Deck
Time Limit: 5 Seconds, Memory Limit: 2G

You are playing a single-player game with a deck of cards. The deck has N cards, each with an integer between 0 and K written on it. You shuffle the deck and draw a card, which forms your starting hand. You then play the game by repeatedly choosing and discarding a card from your hand. Each time you do so, you draw as many cards from the top of the deck into your hand as the integer written on the card you just discarded. (If there are not enough cards left in the deck, you draw them all.) You win if you draw all of the cards from the deck and you lose if you run out of cards in your hand when there are still cards left in the deck. Given the contents of the deck, and assuming that all possible shuffles of the deck are equally likely and that you play optimally, what is the probability you win the game?

Input

The first line of input contains two space-separated integers N and K, where N $(1 \leq N \leq 1\,500)$ is the number of cards in the deck and K $(0 \leq K \leq 3)$ is the largest integer written on any of the cards.

The second line contains $K+1$ space-separated integers a_i $(0 \leq a_i \leq N)$, starting at $i = 0$: the number of cards in the deck with the integer i written on them. It is guaranteed that $a_K > 0$ and that the sum of all of the a_i is N.

Output

Print a real number: the probability that you win if you play optimally. Your answer will be accepted if it differs from the judge solution by an absolute error of at most 10^{-6}.

Sample Input 1	Sample Output 1
4 2 2 0 2	0.3333333333333333

Sample Input 2	Sample Output 2
5 1 3 2	0.0

This page is intentionally left blank.

Problem E
Evil Judges
Time Limit: 1 Second, Memory Limit: 2G

The evil judges preparing the ICPC problem sets are tired of seeing the talented contestants AC their problems and AK ("all kill") their problem sets. They have grown so tired of this that they've started to dislike any strings where `AC` or `AK` appear as subsequences. The judges just found a string s consisting of only the characters `A`, `C`, and `K` and are determined to destroy these subsequences!

In one operation, the judges are able to swap two adjacent characters in the string s. To be more precise, the judges may choose an index i ($1 \leq i < |s|$) and swap s_i and s_{i+1}. The judges are very busy and don't have much time, so they can only do this operation up to M times (independently choosing an index i each time).

The judges' goal is to minimize the number of subsequences (**possibly non-contiguous**) that are `AC` or `AK`. Help them calculate the number of `AC` and `AK` subsequences that remain within s after the judges perform an optimal sequence of up to M swap operations.

Input

The first line of input contains the string s ($1 \leq |s| \leq 2 \cdot 10^5$). The string consists only of the characters `A`, `C`, and `K`.

The second line contains an integer M ($0 \leq M \leq 2 \cdot 10^9$), the maximum number of swaps the judges can perform.

Output

Print the minimum total number of `AC` and `AK` subsequences that remain in s after an optimal sequence of at most M swap operations.

Explanation of Sample Input 1

In Sample Input 1, the string s initially has seven different `AC` subsequences and four different `AK` subsequences. One optimal choice of five swaps yields the new string `ACCCAAKA` which only has three `AC` subsequences and three `AK` subsequences left.

Sample Input 1	Sample Output 1
ACAACCAK 5	6

Sample Input 2	Sample Output 2
CCKKAKCKCK 1000000000	0

Sample Input 3	Sample Output 3
AAAAAAAAACCCCCCCCC 13	68

Problem F
Friend Meetup
Time Limit: 3 Seconds, Memory Limit: 2G

A group of friends live happily on a 2D Manhattan grid, which has a horizontal road $y = a$ running through it for every integer a and a vertical road $x = b$ for every integer b. Each friend is located at the intersection of two roads and has a walking speed (in grid units per second). They can only travel by moving along the roads at those speeds.

Life on the grid gets boring, so pairs of friends sometimes like to meet up. They do so by moving towards each other along routes that cause them to meet at a common point as quickly as possible. (This point does not have to be at the intersection of two roads; but does have to lie on a road, of course.) They would like to know: over all possible pairs of friends, what's the longest it could take a pair of friends to meet up?

Input

The first line of input contains an integer N ($2 \le N \le 2 \cdot 10^5$), the number of friends.

Each of the next N lines contains three space-separated integers x, y, and v ($|x|, |y| \le 10^6, 1 \le v \le 10^6$), indicating a friend located at (x, y) who travels at v units per second along the grid.

Output

Print the real number of seconds it would take for a pair of friends to meet up for whom this time is the longest, assuming that each pair takes optimal routes to meet up as quickly as possible. Your answer will be accepted if it differs from the judge solution by relative or absolute error at most 10^{-6}.

Sample Input 1	Sample Output 1
3 0 0 1 1 1 3 -1 1 4	0.5

Sample Input 2	Sample Output 2
6 970000 560000 3 -530000 510000 1 -300000 210000 4 -780000 -180000 1 460000 420000 5 890000 600000 9	622500.0

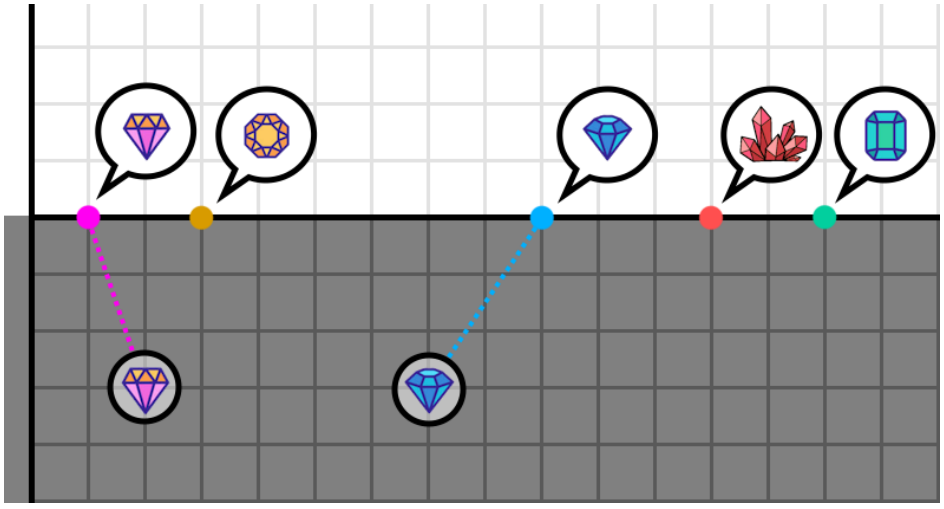
Problem G
Gemstone Dowsing
Time Limit: 3 Seconds, Memory Limit: 2G

Rockhounding is the hobby of searching for valuable rocks and gemstones in the natural environment. As a national-level rockhound yourself, you're determined to find the most valuable gem possible!

You've found an area of land whose cross section can be modeled by a 2D Euclidean plane with the ground at the line $y = 0$. Everything below this line is solid stone, and everything above is air. Buried within the stone are N rare gemstones, each at a (potentially non-unique) (x, y) location with $y < 0$.

Some of these gemstones have already been traced by other rockhounds, who have claimed their discovery by publishing their locations (while leaving the stones themselves in place). That still leaves some gemstones for you to find!

To actually find gemstones, you plan to use an *oscilloscope* to detect waveforms emitted by each gem. Each gem has a unique frequency that can be measured from a distance; however, this specific oscilloscope has a quirk in that each time it is used, it only records the frequency emitted by the **closest gem**, using Euclidean distance. In the case of a tie, it arbitrarily picks a frequency emitted by one of the closest gems.



An illustration of Sample Input 1. Gem icons underground represent previously discovered gemstones, and points on the surface indicate your oscilloscope readings.

You've just used the oscilloscope N times at various **unique** locations $(x_j, 0)$ on the surface of the Earth. You recorded these locations along with the frequency f_j detected by the oscilloscope at that location. Interestingly enough, you've noticed that the frequency of **every gemstone appears exactly once** in your records.

Of the gems that haven't yet been discovered by other rockhounds, you'd like to find the most valuable one. And of course, the deeper the gem, the more valuable!

A *plausible configuration* of the gemstones is a placement of each undiscovered gemstone on the 2D Euclidean plane that satisfies the following conditions:

- every gemstone is underground ($y < 0$);
- for every oscilloscope reading of frequency f_j at location $(x_j, 0)$, no gemstone is closer to $(x_j, 0)$ in Euclidean distance than the gemstone with frequency f_j.

You wish to calculate, for each yet-undiscovered gemstone, the deepest possible (most negative) y-coordinate of that gem among all plausible configurations of gemstones.

Input

The first line contains two space-separated integers N ($2 \le N \le 10^5$) and K ($1 \le K \le N - 1$): the number of buried gemstones (and oscilloscope readings) and number of those gemstones that have already been discovered by other rockhounds, respectively.

Each of the next K lines contains three space-separated integers x_i ($|x_i| \le 10^6$), y_i ($-10^6 \le y_i < 0$), and f_i ($1 \le f_i \le N$), describing the location and frequency of a gemstone that has already been discovered. It is possible that two or more gemstones occupy the same location. It is guaranteed that the f_i values are unique.

Finally, each of the last N lines contains two space-separated integers x_j ($|x_j| \le 10^6$) and f_j ($1 \le f_j \le N$), describing an oscilloscope reading. It is guaranteed that the x_j values are unique, that they are listed in increasing order, and that every gemstone (discovered and undiscovered) has exactly one oscilloscope reading. It is also guaranteed that there is at least one plausible configuration of gemstones.

Output

For each of the $N - K$ undiscovered gemstones, print a line with an integer and a negative real number: the frequency f_ℓ of the gemstone and the deepest (most negative) possible y-coordinate y_ℓ of that gemstone among all plausible configurations of the gemstones. It can be proved that this deepest possible y-coordinate exists for every undiscovered gemstone and is negative and finite.

You may print these lines in any order. Your answer will be accepted if each depth value you assign to an undiscovered gemstone differs from the judge solution by relative or absolute error at most 10^{-5}.

Sample Input 1	Sample Output 1
5 2 7 -3 4 2 -3 1 1 1 3 3 9 4 12 5 14 2	2 -7.615773 3 -3.162278 5 -5.830952

Sample Input 2	Sample Output 2
3 2 3 -3 1 3 -3 2 0 1 2 2 5 3	3 -3.605551

This page is intentionally left blank.

Problem H
Heist of the Century
Time Limit: 4 Seconds, Memory Limit: 2G

After planning the most elaborate heist to steal The Crown Jewel of Count Monte Carlo, you have encountered the final obstacle in your path: a locked safe. However, you have trained for this very moment and honed your safecracking abilities.

The safe has N dials, each of which can be set to any integer value from 1 to $2N$. Count Monte Carlo has configured the safe with a correct, secret value for each dial. To attempt to open the safe, you set each dial to a value of your choice, and then turn the safe door handle. If every single dial is set to its correct secret value, you will feel no resistance and the door will swing open immediately.

Of course, opening the safe by randomly guessing all of the right secret values is highly unlikely to succeed. However, as an expert safecracker, even if your guess is wrong you feel some resistance when you attempt to open the door and you can use that knowledge to help decipher the correct secret values. If a dial has secret value h and the dial is set to d when you try to open the door, the dial will exert resistance $|h - d|$ to turning the door handle. You can feel the *maximum* resistance over all of the dials. (Note that if this value is 0, you have successfully opened the safe and completed your heist!)

Unfortunately, the security team has been made aware of your presence and they are closing in on your location. You are able to make one attempt at opening the safe per second, but they are $4N$ seconds away! Can you complete the heist before you get caught?

Interaction

This is an interactive problem. Interaction begins by reading an integer N ($1 \leq N \leq 500$) from standard input, the number of dials on the safe. Your program may make up to $4N$ attempts to open the safe door by specifying a value to try for each safe dial. You will then be told the resistance you feel from the door handle after each attempt.

To make an attempt, print a line containing N space-separated integers $d_1, d_2, \ldots, d_N$ ($1 \leq d_i \leq 2N$), the values you propose for each dial. Then read a single integer from standard input representing the resistance that you felt from the door handle, $\max_i |h_i - d_i|$, where h_i is the (unknown to you) secret value of dial i. If the resistance is 0, you have opened the safe and your program should terminate. Otherwise, if you have attempts remaining, you may try again.

If you run out of attempts, your program should exit cleanly (though it will be judged incorrect for failing to crack the safe in time to escape the guards).

The secret value of each dial was configured by Count Monte Carlo in advance of the heist and will not change in response to your cracking attempts.

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 resistance value, 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 explaining its use.

Read	Sample Interaction 1	Write
3		
		1 1 1
5		
		3 4 5
1		
		4 5 6
0		

Problem I
I Don't Miss Pennies
Time Limit: 2 Seconds, Memory Limit: 2G

In November 2025, the United States minted its last pennies (one-cent coins). Canada has not minted pennies since 2012.

Billions of pennies remain in circulation, but pennies are expected to fade from circulation over time. Stores are expected to continue to price items to the cent, as credit card transactions are processed to the cent. However, for cash transactions, stores are expected to round to the nearest nickel (five cents). Specifically, if the final digit of a total purchase ends in 3, 4, 8 or 9 cents, the total will be rounded up; if it ends in 1, 2, 6 or 7 cents, it will be rounded down. Transactions ending in 0 or 5 cents are not rounded.

You realize that this provides an opportunity. If you pay cash and rearrange and group your purchases appropriately, you may be able to pay slightly less for everything you buy! Given the prices of the individual items you wish to buy in cents, determine the maximum amount you could save by paying solely in cash and rearranging and grouping your purchases optimally, compared to the non-rounded total price you would pay if you bought all of the items with a credit card.

Input

The first line of input contains a single integer N ($1 \leq N \leq 3 \cdot 10^5$), the number of items you intend to purchase.

The next line contains N space-separated integers p_i ($1 \leq p_i \leq 3\,000$), the prices of the items in cents.

Output

Print a single integer: the difference between the total credit card price without rounding and the lowest total cost that can be obtained by paying cash and grouping purchases optimally.

Explanation of Sample Input 1

For the first sample, one optimal grouping of items is

$$\{78, 999, 350\}, \{59, 173, 2995, 350\}, \{882, 298\}, \{1096\}, \{497\}.$$

The prices of each group are, respectively, $1427, 3577, 1180, 1096, 497$ and thus the total difference is

$$(1427 + 3577 + 1180 + 1096 + 497) - (1425 + 3575 + 1180 + 1095 + 495) = 7.$$

You save 7 cents by optimally grouping the items into several cash transactions instead of paying for everything with a credit card.

Sample Input 1	Sample Output 1
11 78 59 90 999 173 882 1096 2995 298 497 350	7

Sample Input 2	Sample Output 2
2 199 299	-2

Problem J
Jelly Fusion
Time Limit: 3 Seconds, Memory Limit: 2G

As an evil scientist, you have figured out how to create bigger and scarier jellies in your secret laboratory. Arranged around your circular lab bench is a *cyclic array* of N rectangular jellies, each with a certain height and width. Note that the first and last jellies are adjacent.

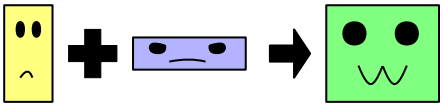


Illustration of jelly fusion.

You can fuse two adjacent parent jellies together to produce a combined jelly with height equal to the maximum of the two parent jelly heights and width equal to the maximum of the two parent jelly widths. The fused jelly replaces the parents at their spot in the cyclic array (which is now one element shorter) and could in principle be fused again with one of its two neighbors.

You want to maximize the sum of the areas of your jellies by performing this fusion operation any number of times. After all, the more total area your jellies can cover, the more cities you can conquer!

Input

The first line of input contains an integer N ($1 \leq N \leq 3 \cdot 10^5$), the number of jellies initially on your lab bench.

The next N lines describe these jellies, in order. The i^{th} line contains two space-separated integers h_i and w_i ($1 \leq h_i, w_i \leq 10^6$), the height and width of the i^{th} jelly.

Output

Print a single integer: the sum of the areas of the jellies that remain in your cyclic array after you perform any number of fusion operations to optimally maximize this sum.

Sample Input 1	Sample Output 1
2 6 3 2 7	42

Sample Input 2

```
4
1 5
10 10
5 1
6 7
```

Sample Output 2

```
152
```

Sample Input 3

```
5
6 2
5 1
1 5
2 7
1 2
```

Sample Output 3

```
67
```

Sample Input 4

```
1
380385 222650
```

Sample Output 4

```
84692720250
```

Problem K
Kindergarten Revisited
Time Limit: 4 Seconds, Memory Limit: 2G

In kindergarten, one of the most time consuming activities was cutting out shapes from a piece of paper with safety scissors. Let's look at a simplified model of this task: you start with an infinitely large sheet of paper with N disjoint axis-aligned rectangles drawn on it, and cuts are infinitely long straight lines. A cut must not "nick" any rectangle: it must not pass through any point strictly on the interior of any rectangle. (Cuts that pass *exactly* along a rectangle edge or through a rectangle corner are allowed.) When you cut a piece of paper, the paper falls apart into two different pieces of paper that you continue cutting independently of each other (future cuts on one piece do *not* affect any other pieces).

Your goal is to make a sequence of cuts so that each rectangle ends up on its own piece of paper (since after that it's pretty easy to cut out each rectangle exactly).

Determine the minimum number of cuts (not necessarily axis-aligned) needed to cut out the rectangles in this way. If the task is impossible, print `impossible` instead.

Input

The first line of input contains an integer N ($1 \leq N \leq 100$), the number of rectangles.

Each of the next N lines describe one rectangle. Each line contains four space-separated integers x_1, y_1, x_2, and y_2 ($|x_1|, |y_1|, |x_2|, |y_2| \leq 10^9$, $x_1 < x_2$, $y_1 < y_2$), where (x_1, y_1) is the bottom-left corner of the rectangle and (x_2, y_2) is the top-right corner of the rectangle.

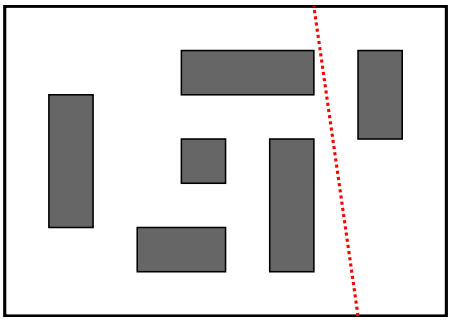
The rectangles are guaranteed to be disjoint: no two rectangles intersect at any common point, including on their edges or corners.

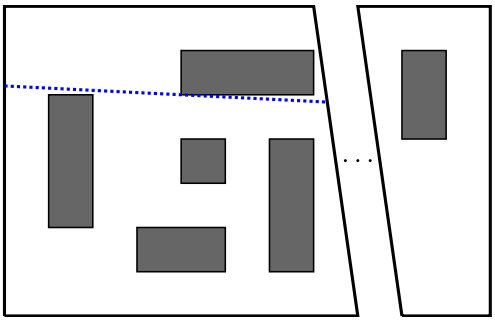
Output

Print the minimum number of cuts needed to separate all rectangles. (Do *not* include additional cuts that would be needed to trim blank paper from around the margins of the rectangles once separated.) If this task is impossible, print `impossible` instead.

Explanation of Sample Input 1

The first two cuts in one possible sequence of cuts that separate the rectangles are shown below. The first cut is drawn in red and the second in blue. Note that the blue cut is not valid before the red cut, as it would have nicked the rectangle on the right hand side.





Sample Input 1

```
6
-1 1 0 4
1 0 3 1
2 2 3 3
4 0 5 3
2 4 5 5
6 3 7 5
```

Sample Output 1

```
5
```

Sample Input 2

```
4
0 -1 1 2
2 -1 5 0
-10 3 3 4
4 1 5 13
```

Sample Output 2

```
impossible
```

Problem L
Leaking Santa's Secrets
Time Limit: 1 Second, Memory Limit: 2G

It's Christmas time! While most workplaces are participating in Secret Santa gift exchanges, your workplace is hatching a more sinister plot: uncovering Santa's secrets.

Santa has a naughty-or-nice list for every human on Earth. Because its contents are so sensitive, the list is written in North-Polish, an arcane language with N letters. For further security, Santa has enciphered this list with a *substitution cipher*: a permutation H of the numbers $1, 2, \ldots, N$ that maps each North-Polish letter i to a distinct letter $H(i)$. In such a cipher, no two letters map to the same target letter—formally, if $i \neq j$, then $H(i) \neq H(j)$—since otherwise Santa would not be able to decipher his list! (Santa can choose to map some letters to themselves, $H(i) = i$, just to be extra tricky.)

Fortunately for you, Santa's server was poorly vibe-coded and is exposed to the public Internet. You and your coworkers hope to hack into Santa's server, decipher his list, and confirm that you're all naughty! (Hackers are always naughty.)

The server was built so that Santa could quickly check his list on the go. After a user connects to the server, it prompts them to input the list of N integers $H(1), H(2), \ldots, H(N)$ encoding the permutation H, verifies that this list is correct, and then deciphers Santa's secret list. Through months of research, you have found a side-channel timing vulnerability. Say you type in a permutation Q. If $H = Q$, then the server instantly grants access. Otherwise, consider a graph on N vertices, and add an edge from each vertex i to vertex $H(Q(i))$. You have discovered that the server's convoluted authentication algorithm will take exactly as many seconds to respond to you with an Access Denied error message as the number of connected components in the resulting graph.

For example, suppose that $N = 4$ and the cipher permutation H is the following:

i	1	2	3	4
$H(i)$	2	3	1	4

If you try to log in to the server with input `4 3 2 1`, since this permutation does not match H and since the graph described above has two connected components (one containing a cycle of edges $1 \to 4 \to 2 \to 1$ and another containing just the self-loop $3 \to 3$), the server will respond with an Access Denied error message after a delay of 2 seconds.

Note that if you try to log in to the server multiple times with different inputs Q, it will authenticate Q each time against the *same* stored H. It does not change H in any way in response to your inputs.

Santa will notice if his server is bombarded with unauthorized requests. You've estimated that you can only make $1\,510$ login attempts before drawing too much suspicion. Can you find an efficient strategy for determining the cipher permutation?

Interaction

This is an interactive problem. Interaction begins by reading an integer N ($1 \leq N \leq 220$) from standard input, the number of letters in North-Polish. The judge is not adaptive: the hidden permutation H is chosen at this time and will not change throughout the rest of the interaction.

To attempt to log in to the server, print a line with N space-separated integers $Q(1), \ldots, Q(N)$, where Q is a permutation of $\{1, 2, \ldots, N\}$. Then read a single integer from standard input, the amount of time in seconds it takes for the server to respond to your input.

If this delay is 0, you have successfully found the cipher permutation H and your program should terminate. If not, this delay is the number of connected components in the graph built according to the procedure described above.

You may attempt to log in at most $1\,510$ times. If you run out of attempts, your program should exit cleanly (though it will be judged incorrect for failing to decipher Santa's naughty-or-nice list).

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 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 the number of components, 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 explaining its use.

Read	Sample Interaction 1	Write
3		
		1 2 3
1		
		2 1 3
2		
		3 1 2
0		

Problem M
Maki Conveyor Belt
Time Limit: 3 Seconds, Memory Limit: 2G

Alice and Bob are eating at a conveyor belt maki restaurant. (Maki is a kind of sushi). Diners in the restaurant sit around a circular conveyor belt with N positions numbered from 1 to N in clockwise order. Alice sits at position p_A and Bob sits at position p_B.

The restaurant serves M different kinds of maki. There are K different plates set out on the conveyor belt. The i^{th} plate consists of x_i pieces of a single kind of maki and each piece costs c_i coins. The same kind of maki may appear on multiple plates, and have different costs on different plates. No more plates will be added beyond what is already on the conveyor belt and no other customers are present at the restaurant (perhaps they picked a poorly rated maki restaurant...).

There is at most one plate per position. Every second, the conveyor belt rotates clockwise. Formally, if there is a plate at position N, it moves to position 1; and all other plates move from position i to position $i + 1$. When a plate is in front of a diner, they may immediately grab any number of pieces from it, or choose not to grab any. For example, if there is a plate with 5 pieces of maki in front of Alice, she can choose to only grab 2. The diners may grab maki from plates in front of them before the belt first rotates.

Alice and Bob want to return home as quickly as possible to watch their favorite sushi documentary. They know the full layout of the restaurant, and each have a desired number of pieces of each maki type they want to eat in order to be satisfied. Help them determine the minimum time (in seconds) they need to spend in the restaurant and the minimum cost (in coins) for them to become satisfied within that time.

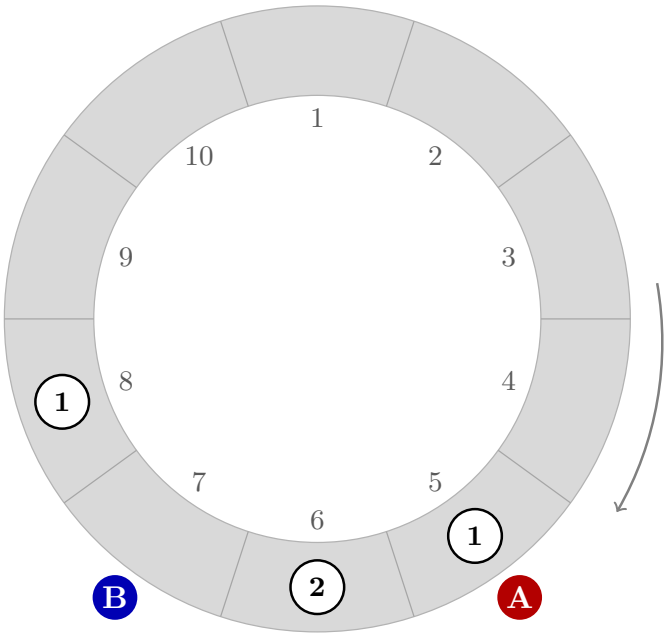
Input

The first line of input contains five space-separated integers N, M, K, p_A, and p_B, where N ($2 \le N \le 10^9$) is the number of conveyor belt positions, M ($1 \le M \le 10^5$) is the number of maki types, K ($1 \le K \le \min(2 \cdot 10^5, N)$) is the number of plates, and p_A and p_B ($1 \le p_A, p_B \le N, p_A \ne p_B$) are Alice's and Bob's positions respectively.

The second line contains M space-separated integers a_i ($0 \le a_i \le 10^6$), where a_i is the number of pieces of maki of type i that Alice wants to eat.

The third line contains M space-separated integers b_i ($0 \le b_i \le 10^6$), where b_i is the number of pieces of maki of type i that Bob wants to eat.

Each of the next K lines describe a plate. The j^{th} line contains four space-separated integers s_j, t_j, x_j, and c_j, where s_j ($1 \le s_j \le N$) is the starting position of the plate, t_j ($1 \le t_j \le M$) is the type of maki on the plate, x_j ($1 \le x_j \le 10^6$) is the number of pieces on the plate, and c_j ($1 \le c_j \le 10^6$) is the cost per piece. All plates have different starting positions (all s_j are distinct).



Initial position of Alice, Bob, and the plates in Sample Input 1.

Output

Print two integers: the minimum time in seconds that Alice and Bob will need to spend in the restaurant and the minimum cost in coins for them to become satisfied within that minimum time. If it is impossible for both of them to ever be satisfied, print `impossible`.

Sample Input 1

```
10 2 3 5 7
3 1
4 1
5 1 9 2
6 2 5 3
8 1 9 7
```

Sample Output 1

```
9 20
```

Sample Input 2

```
5 1 1 2 3
2
2
5 1 3 3
```

Sample Output 2

```
impossible
```