Problem A. Flag Bearer

Source file name:	Bear.c, Bear.cpp, Bear.java, Bear.py
Input:	`Standard`
Output:	`Standard`

The bears have decided to abandon their solitary lives as predators and hunt together. They have formed two groups on the opposite sides of the farm and are preparing to abduct several smaller piglets, whose fate now seems very uncertain.

To prevent their plans from being revealed, the bears will communicate during the operation at a distance using the semaphore alphabet, which they will further encrypt with Caesar's cipher. Caesar's cipher is defined by a single non-negative integer C. Each letter in the message is encrypted by replacing it with the letter that is C positions further in the alphabet. When the position of the used letter goes beyond the end of the alphabet, counting continues cyclically from the start of the alphabet. So, for example, with $C = 5$, the second-to-last letter in the alphabet is encrypted as the fourth letter in the alphabet. Each letter of the semaphore alphabet is encoded by the position of two lower or upper limbs of a signaling mammal that stands upright, facing the recipient of the message. In the diagram shown below, the center of the diagram represents the torso position of the signaling mammal, and its two limbs are represented by two of the eight possible segments radiating from the center. The assignment between limb positions and letters of the alphabet is provided on a separate page (following the examples).

Now, the bears are going to encrypt their short messages. They are good at semaphore alphabet, but they need some help with Caesar's cipher. Please provide them with this assistance.

Input

The first input line contains two integers N ($1 \leq N \leq 26$) and C ($0 \leq C \leq 25$), representing the length of the word to be encrypted and the constant of the Caesar cipher. Next, there are $9N$ lines, each with 9 characters. Each 9 successive lines represent one character in the message. The characters are "`.`", "`*`", or "`#`". Asterisk "`*`" appears only once in each 9 lines and it corresponds to the center of the semaphore scheme. Hash symbols "`#`" correspond to limbs. Coded words use standard English alphabet with 26 characters.

Output

Print the encrypted message, in the same format as the input.

Example

Input

```
5 9
.........
.........
.........
.........
.###*....
.....#...
......#..
.......#.
.........
.........
.......#.
......#..
.....#...
....*....
...#.....
..#......
.#.......
.........
.........
.......#.
......#..
.....#...
....*....
....#....
....#....
....#....
.........
.........
.......#.
......#..
.....#...
....*....
....#....
....#....
....#....
.........
.........
....#....
....#....
....#....
.###*....
.........
.........
.........
.........
```

Output

```
.........
.........
.........
.........
.###*....
....#....
....#....
....#....
.........
.........
.#.....#.
..#...#..
...#.#...
....*....
.........
.........
.........
.........
.........
.........
.........
.........
....*....
...#.#...
..#...#..
.#.....#.
.........
.........
.........
.........
.........
....*....
...#.#...
..#...#..
.#.....#.
.........
.........
.#.......
..#......
...#.....
....*###.
.........
.........
.........
.........
```

Semaphore alphabet

```
Down        Left/Down   Left        Left/Top
.........   .........   .........   .........
.........   .........   .........   .#.......
.........   .........   .........   ..#......
.........   .........   .........   ...#.....
....*....   ....*....   .###*....   ....*....
....#....   ...#.....   .........   .........
....#....   ..#......   .........   .........
....#....   .#.......   .........   .........
.........   .........   .........   .........

Top         Top/Right   Right       Down/Right
.........   .........   .........   .........
....#....   .......#.   .........   .........
....#....   ......#..   .........   .........
....#....   .....#...   .........   .........
....*....   ....*....   ....*###.   ....*....
.........   .........   .........   .....#...
.........   .........   .........   ......#..
.........   .........   .........   .......#.
.........   .........   .........   .........
```

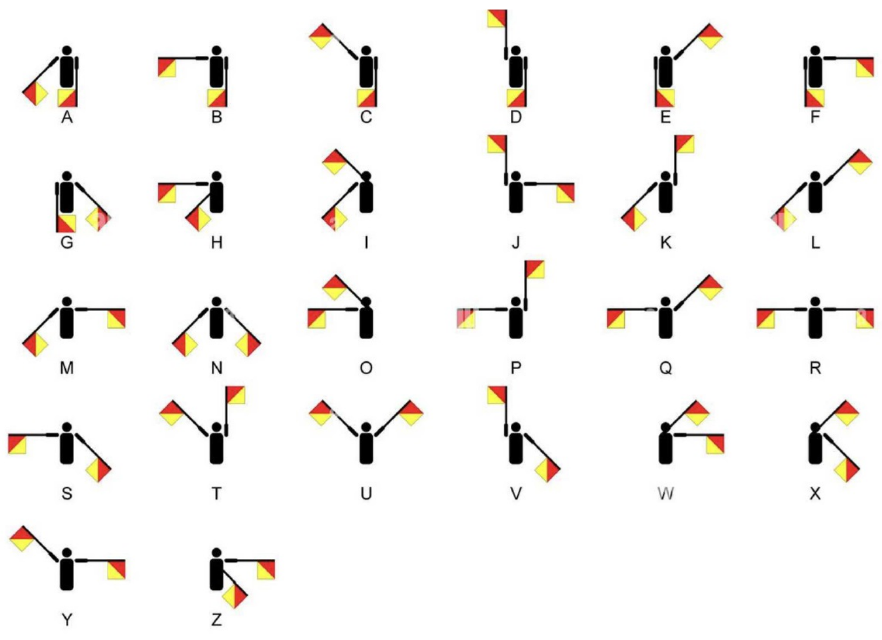


Figure 1: Semaphore Alphabet

Problem B. Cowpproximation

Source file name:	Cow.c, Cow.cpp, Cow.java, Cow.py
Input:	`Standard`
Output:	`Standard`

There are cows in a 2 by 2 kilometer field. The cows want to have a group meeting, and all of them must attend! It is up to you to plan the meeting to take place as soon as possible.

To model this problem, we approximate each cow as a circle, and for our purposes, there are no collisions – cows can pass through each other freely.

The i-th cow is represented by its initial position $(x_i,\ y_i)$ and its radius r_i, all distances are measured in meters. You can tell each cow to travel in any direction from its initial position at a constant speed from 0 to 1 meter per second. The meeting takes place as soon as all cows occupy the same point in the plane, i.e., there exists a point contained within all circles representing the cows. Find the minimum time in seconds required for the cows to meet.

Input

The first input line contains a single integer N $(1 \leq N \leq 10^3)$, representing the number of cows. Each of the following N lines contains three space separated integers x_i, y_i, r_i, describing a cow, where $-10^3 \leq x_i, y_i \leq 10^3$ and $1 \leq r_i \leq 10^3$.

Output

Output the minimum time T in seconds after which all the cows meet. Your answer will be considered correct if it has an absolute or relative error at most 10^{-5}. Formally, let T_{OPT} be the optimal value. Then your answer is considered correct if either $|T - T_{OPT}| \leq 10^{-5}$ or $\left|\frac{T - T_{OPT}}{\max\{1,\ T_{OPT}\}}\right| \leq 10^{-5}$ holds.

Example

Input	Output

2
-10 3 2
10 3 4
	7.0000000

3
-4 -4 1
6 0 3
0 8 5
	3.7039181

Explanation

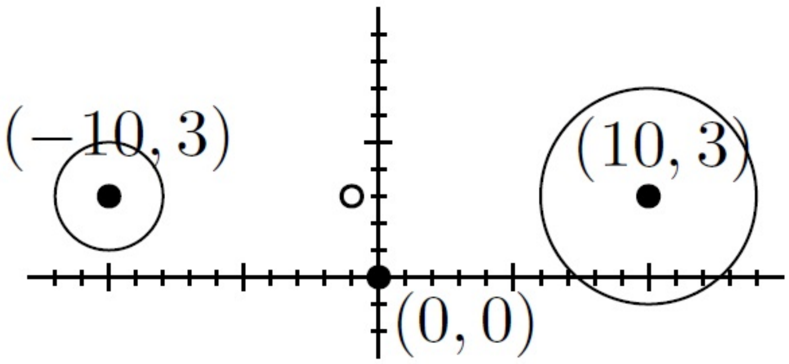


Figure 1: Visualization of example input 1. The empty dot represents the meeting location.

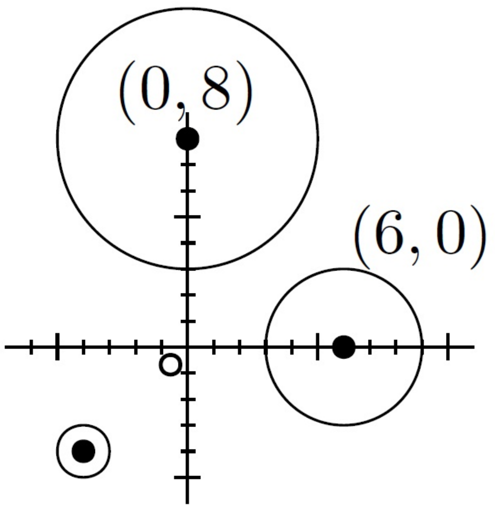


Figure 2: Visualization of example input 2. The empty dot represents the meeting location.

Problem C. Reptile Eggs

Source file name:	Eggs.c, Eggs.cpp, Eggs.java, Eggs.py
Input:	`Standard`
Output:	`Standard`

The farm generates additional income by producing decorated turkey eggs. The farmers, being excessively creative, design various patterns and apply them to the surface of the eggs. The customer orders are handled with the help of specially trained reptile specimen, often it is a Komodo dragon or a member of another suitable reptile species. The reptile had been trained to understand and execute commands in a specific formal language. A customer order is translated into a command for the reptile which then selects a corresponding group of eggs from the production line. The selected eggs are then packed and prepared for shipment.

The command is a regular expression, and the reptile selects such sequence of eggs on the line that exactly matches the given regular expression. The selected sequence does not have to be contiguous, and it does not have to be specified uniquely by the expression. However, the selected sequence always has to be as long as possible, to keep customers satisfied. The reptile recognizes the beginning and the end of the production line, thus the first egg in the selected sequence is not further from the end of the production line than the last egg in the sequence.

Sometimes, the customer's order, and consequently the regular expression, is so demanding that no unempty sequence of eggs can match the expression. In such situation, the reptile is still able to distinguish between two possible scenarios. In the first scenario, an empty sequence containing no eggs can still match the expression. In the second scenario, even an empty sequence of eggs cannot match the expression. In the first scenario, the reptile remains calm, in the second scenario, the reptile gets dangerously annoyed and has to be calmed down.

Check the reptile's operation and, for each given regular expression, determine the maximum possible number of selected eggs in the shipment.

Below is the specification of the regular expressions that control the reptile's activity:

- Each expression is a finite sequence of lowercase letters of the English alphabet, extended with the characters `?`, `*`, `[`, `]`, `(`, `)`.
- Each letter of the English alphabet represents an egg with a specific pattern corresponding to that letter.
- The question mark (`?`) represents one egg with any pattern.
- The brackets `[` and `]` are always correctly paired, and inside them, there may be only lowercase letters of the English alphabet, at least one. This pair of brackets and the letters within them always represent a single egg on the line, one whose pattern matches any letter inside the brackets.
- The asterisk (`*`) replaces any number of repetitions of the sequence immediately preceding the asterisk. This sequence is either a single letter or a question mark or a portion of the expression enclosed in parentheses `()` or square brackets `[]`. The number of repetitions may also be zero.
- Parentheses `(` and `)` are always correctly paired and are always immediately followed by an asterisk. Inside these parentheses, there is always a regular expression adhering to the rules described above, but it does not contain any parentheses.

Input

The first input line contains integer N ($1 \leq N \leq 1000$). The second input line contains a string of N lowercase English letters representing the types of patterns on the eggs on the production line. Each letter corresponds to one egg. The beginning of the production line corresponds to the beginning of the string. The third input line contains integer M ($1 \leq M \leq 1000$). The fourth input line contains a string of M characters representing the regular expression given to the reptile.

Output

Output single integer E, the maximum possible number of eggs in the shipment selected from the given production line, according to the given regular expression. If the regular expression cannot be matched output -1.

Example

Input	Output
3 aba 2 a*	2
3 ctu 4 open	-1
4 food 13 ([ba]g??t*e)*	0
20 alotofeggcellenteggs 14 [only]*(egg)*?	10

Problem D. Fishception

Source file name: Fish.c, Fish.cpp, Fish.java, Fish.py
Input: `Standard`
Output: `Standard`

A farmer marked a large rectangle on his field using four stakes hammered into the ground at the corners. The rectangle had a non-zero area. His plan was to dig a pond for pike there.

Unfortunately, the next morning, he found out that a new law on the minimum size of the living space for pike was being proposed. He decided to prepare for the new conditions. He pulled the stakes out of the ground and hammered them in at new positions. Because he wanted to use all the already prepared land, he made sure that the previous rectangle was entirely contained within the new, larger one. He was so meticulous, that no point on the boundary of the new rectangle coincided with a point on the boundary of the previous rectangle.

As it often happens, the next morning, he learned that the minimum size of the living space for pike would be even larger than what had been discussed the previous day. So, he repeated the process, pulled the stakes out, and hammered them in at new positions, only to find out the next day that the space needed to be expanded again. This repeated for several days.

Finally, when the time to vote came, the proposed law was unanimously rejected. The farmer then decided to return to the original marked area. He pulled out the stakes, thinking he would simply place them back into the holes left by the stakes. However, he suddenly realized that his entire field was now full of holes from all the constant stake-shifting, and he no longer knew which four points represented the original rectangle. To make matters worse, he received a call from the Land Management Office, urgently requesting the area of the land he planned to use for the pond.

We know the coordinates of all the holes left by the stakes. Help the farmer determine the area of the original, smallest rectangle.

Input

The first input line contains one integer N ($4 \leq N \leq 2 \cdot 10^5$), the number of stakes. N is always a multiple of 4. Each of the next N lines contains two integers x and y ($-10^9 \leq x, y \leq 10^9$), specifying the coordinates of one stake. No two stakes share the same coordinates.

Output

Output one integer specifying the area of the rectangle marked on the first day of the farmer's efforts.

Example

Input	Output

16
0 1
1 0
-1 0
0 -1
2 2
-2 2
2 -2
-2 -2
0 6
0 -6
6 0
-6 0
7 7
-7 7
-7 -7
7 -7
	2

16
-3 7
-2 6
-1 5
0 4
0 3
-1 2
-2 1
-3 0
3 3
3 4
6 2
6 5
9 1
9 6
12 0
12 7
	3

Problem E. Pigpartite Giraffe

Source file name:	Giraffe.c, Giraffe.cpp, Giraffe.java, Giraffe.py
Input:	`Standard`
Output:	`Standard`

There are two fenced areas next to each other: one houses a group of pigs, and the other houses a herd of giraffes. While pigs cannot visit the giraffes, and giraffes cannot visit the pigs, they can communicate over the fence. In fact, they have become such good friends that pigs prefer to talk to giraffes instead of other pigs, and giraffes feel the same way-they would rather talk to pigs than to other giraffes.

This relationship is mutual: if a pig likes to talk to a specific giraffe, that giraffe likes to talk back to that same pig. However, no pig likes to talk to another pig, and no giraffe likes to talk to another giraffe.

If an animal wants to send a message to another animal, even if it doesn't like to talk to it directly, it can do so by passing the message through other animals. Messages can only be passed between animals that like to talk to each other.

To quantify the amount of chatter, we define the *noise level* between two animals as the minimum number of talks required for a message to be passed from the source animal to the target animal. If a message cannot be passed between two animals, the noise level for that pair is 0. For instance, if two animals like to talk directly to each other, their noise level is 1. If the message must pass through one other animal, the noise level is 2, and so on.

We are interested in the total noise level, which is the sum of noise levels for all unordered pairs of different animals. Note that the noise level between two different animals is only counted once.

The problem is, the number of animals keeps increasing. New animals are born by combining the properties of two existing animals of the same kind, referred to as the *parents*. The new animal will belong to the same kind as its parents (pig or giraffe). Additionally, the newborn animal can immediately start talking to other animals.

The newborn animal likes to talk to exactly the animals that like to talk to exactly one of its parents. Along with the initial setup of animals and their talking preferences, you will be given a sequence of queries describing the birth of new animals. After each animal is born, your task is to calculate the total noise level.

Input

The first input line contains three integers A, B, M ($1 \leq A, B \leq 8$, and $0 \leq M \leq A \cdot B$), the number of pigs, the number of giraffes, and the number of initial pairs of animals that like to talk to each other, respectively.

Each of the next M lines each contains two integers a b ($0 \leq a < A$, $0 \leq b < B$), indicating that pig number a likes to talk to giraffe number b.

The next line contains integer Q ($1 \leq Q \leq 10^5$), the number of new animals to be born. Each of the following Q lines describes birth of one animal, and it contains a character X and two integers p and q. The character X is either "`A`" or "`B`", indicating that the newborn is either a pig or a giraffe, while p and q are the numbers of its two parents. The newborn animal is assigned the smallest unused positive number within its kind. It is guaranteed that p and q always refer to existing animals.

Output

Output Q lines. On the i-th line, print the total noise level after i-th birth of an animal.

Example

Input	Output
4 4 4 0 0 1 1 2 2 3 3 2 A 0 1 A 0 1	22 30
3 3 5 0 0 1 0 1 1 2 1 2 2 3 B 0 1 B 3 2 B 3 4	42 61 82

Explanation

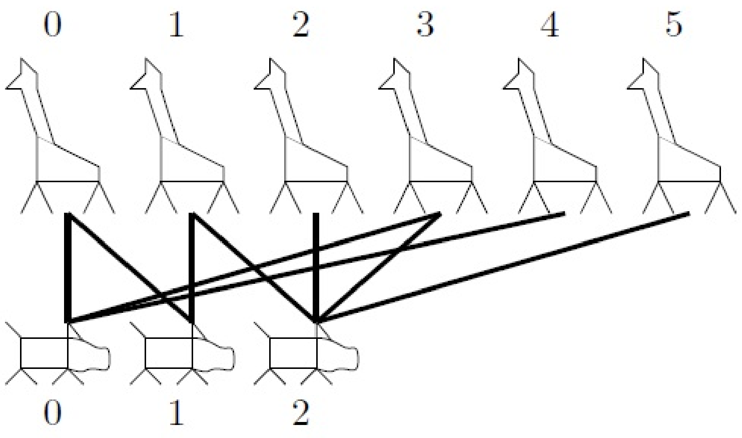


Figure 1: Example Input 2 after all queries are done.

Problem F. Hamster

Source file name:	Hamster.c, Hamster.cpp, Hamster.java, Hamster.py
Input:	`Standard`
Output:	`Standard`

The hamster lives in a rectangular enclosure with floor divided into square fields of the same size, forming a grid. In each field, several larvae are dwelling. The hamster decided to walk around the enclosure, collect the larvae into his bowl, and bring them home for later culinary use. Additionally, he decided to place a trap for common starlings on each field so that they would no longer fly in and steal the larvae. The hamster starts in the northwest corner of the enclosure. From each visited field, he collects the larvae and puts them in the bowl, places a trap on the field, and moves to the next field, which shares a side with the field he is just leaving. The hamster will finish his journey in the southeast corner of the enclosure. But there is a catch, the hamster cannot return to fields he has already visited, otherwise he would get caught in one of his own traps.

We know how many larvae are in each field. Find out the maximum number of larvae the hamster can collect on his journey and avoid all traps.

Input

First line contains two numbers N, M ($2 \leq N, M \leq 1000$), the number of rows and the number of columns in the grid of fields in the hamster's enclosure. Then follow N lines, each contains M numbers representing numbers of larvae in particular fields in the enclosure. Each line represents one row in the grid. The northwest corner of the grid corresponds to the first number in the first of these lines, the southeast corner of the grid corresponds to the last number in the last of these lines.

The number of larvae on any field is between 0 and 1000 inclusive.

Output

Output a single number, the maximum number of larvae the hamster can collect.

Example

Input	Output

2 2
1 2
3 4
	8

3 4
2 2 4 0
1 3 1 0
2 5 3 1
	24

Problem G. Pray Mink

Source file name:	Mink.c, Mink.cpp, Mink.java, Mink.py
Input:	`Standard`
Output:	`Standard`

Aaron, the mink, has two loves: biscuits and prime numbers. Growing up on our farm, he developed a special fondness for both. In the kitchen, he has cookie cutters shaped like the digits 0 through 9. He fills them with dough, bakes the digits, and arranges them in a row on a tray to form a whole positive number.

Recently, Aaron managed to bake and arrange a prime number. After admiring it, he got hungry and ate one of the digit-biscuits. Without rearranging the remaining digits, he simply shifted them left or right to fill the gap. To his delight, the new number was still prime!

Now, Aaron is curious to see how many prime numbers he can find in a row through this process. Each time, he eats a digit, shifts the remaining ones, and checks if the new number is prime. If leading zeros appear after eating a digit, he happily eats those as well right away. However, if at any point the resulting number is not prime, the process ends immediately.

Today, Aaron baked a new set of digits and formed a new number, not necessarily prime. Aaron's goal is to see the maximum number of prime numbers in a row, starting with the initial number and continuing for as long as possible.

Your task is to compute the maximum possible number of occurrences of prime numbers in this process.

Input

The input consists of a single line with integer N ($1 \leq N \leq 10^9$), the number baked by Aaron.

Output

Output one integer specifying the maximum number of prime numbers that Aaron can see through his process.

Example

Input	Output
773	3
300007	2
11	1

Problem H. Ornithology

Source file name:	Ornithology.c, Ornithology.cpp, Ornithology.java, Ornithology.py
Input:	`Standard`
Output:	`Standard`

On the outskirts of the town, close to the farm, there stand two parallel power lines, separated by a narrow dirt road. The power lines were a favorite resting spot for a variety of birds.

Today, on this cool autumn morning, the lines are filled with a group of birds. On the first line there are n consecutive positions for birds numbered 0, 1, . . ., $n - 1$. On the second power line there are also n positions numbered in the same manner.

Initially, every position of the first line is occupied by some birds (possibly zero) and there are no birds on the second power line. Each bird has its desired position to fly to on the second line. No two birds on the same position on the first line share the same desired position.

At one moment, all the birds at once will decide to fly to their desired positions. Every bird flies along the straight line segment connecting its initial and desired position.

It can happen that some pairs of birds crash into each other during their flight. This can happen when their corresponding line segments cross. We shall call such unordered pair of birds *dangerous* pair.

For example, if a bird on position 2 wants to go to position 1 and the bird on position 1 wants to go to position 2, their paths cross.

The birds will not collide if their paths have the same desired position (on the second line) or if they start from the same position (on the first line). In other words a pair of birds with the same initial or desired positions is not considered a dangerous pair.

The task is very simple. Compute the number of dangerous pairs of birds.

Input

First line of input contains an integer n ($1 \le n \le 2 \cdot 10^5$) representing the number of positions on both power lines.

Each of the next n lines describe the desired positions of the birds. The i-th line starts with an integer p_i ($0 \le p_i \le n$) representing the number of birds on position i.

Then, there are p_i distinct numbers $q_{i,1}, \ldots, q_{i,p_i}$ ($0 \le q_{i,j} \le n - 1$) representing the desired places of the p_i birds.

It is guaranteed that the sum $\sum_{i=0}^{n-1} p_i$ does not exceed $2 \cdot 10^5$.

Output

Output exactly one line containing one integer – the number of dangerous pairs of birds.

Example

Input	Output

3
2 1 2
1 0
1 1
	3

Problem I. P||k Cutting

Source file name:	Pork.c, Pork.cpp, Pork.java, Pork.py
Input:	`Standard`
Output:	`Standard`

The farmer needs some pork to be packed. This may take several days, so the farmer has allocated N possible days when the packing can be performed. The whole packing process needs to be completed in a sequence of consecutive days with at least one day. This sequence of days, also called an interval, may not necessarily contain all allocated days.

The farmer has called the bulls and donkeys to help and to do the entire packing job. The bulls and donkeys do not work for free, so the farmer assigned to each day a particular number of cookies he is willing to pay, if the animals work on that day.

The bulls, not being the smartest animals, protested against using such advanced mathematics as summing several numbers of cookies in several consecutive days. They insist on being paid a fixed amount of K cookies, no matter when or for how long they work.

On the other hand, the donkeys considered the sum to be a too ordinary mathematical operation and required that they are paid the bitwise OR of the numbers of cookies offered for the individual days.

Bitwise OR of two given integers is computed by taking their binary representation and keeping the digit 1 in the result on exactly those positions where at least one of the given two integers has digit 1. For instance, bitwise OR of 9 and 5 is 13.

Furthermore, the bulls and donkeys are willing to work only when they are both paid the exact same total amount. The farmer, finding himself unwilling to deal with the animals any longer, agreed to their conditions and wants to know how many options does he have for scheduling the work. As such, your task is to count the total number of distinct non-empty intervals of days on which the animals are willing to work. Counted intervals may overlap.

Input

The first input line contains two integers N, K ($1 \leq N \leq 4 \cdot 10^5,\ 0 \leq K \leq 10^9$), the total number of days in which pork can be packed on the farm, and the amount of cookies the bulls want to be paid.

The second line contains N integers a_i ($0 \leq a_i \leq 10^9$), the offered number of cookies on the i-th day.

Output

Output single integer P, the number of intervals in which the animals are willing to work together.

Example

Input	Output

`5 3`
`1 2 1 2 3`
	`11`

Problem J. Rabid Rabbit

Source file name:	Rabbit.c, Rabbit.cpp, Rabbit.java, Rabbit.py
Input:	`Standard`
Output:	`Standard`

Famous magician Leonardo from Pisa is about to visit the farm and amuse the staff with his tricks. There is a long row of rabbit hutches along one wall of the farm yard. Each hutch is permanently occupied by some number of rabbits. To perform his tricks on rabbits, the magician needs to exploit the row of rabbit hutches. However, the farm is planning maintenance, and they can offer the magician only one contiguous segment of the row of the hutches. To help Leonardo, they compiled a list of segments. Leonardo will choose one segment from the list and the hutches outside the segment will be inaccessible to him due to the maintenance.

Before Leonardo chooses a suitable segment, he has to consider his additional restrictions. Each day, Leonardo will perform one trick on the chosen segment. Fundamental part of each trick is a pair of rabbit hutches selected in the segment in such a way that when the total number of rabbits in both hutches is calculated, it is equal to one of the so-called positive Fibonacci numbers. Each day, the calculated Fibonacci number has to be different from all Fibonacci numbers calculated in the previous days, otherwise the principle of magician's tricks could be cracked and publicized. When Leonardo cannot perform more tricks, he has to leave the farm.

Help Leonardo to determine how many days at most he can stay on the farm, for each given segment of rabbit hutches.

The so-called positive Fibonacci numbers form an infinite sequence of integers. It starts with integers 1 and 2, and each next entry in the sequence is the sum of two entries immediately preceding this entry in the sequence.

Input

The first input line contains two integers N, Q $(1 \leq N, Q \leq 10^5)$, the number of rabbit hutches in the row, and the number of entries in the list of segments compiled by the farm. The second line contains N space-separated integers A_0, A_1, …, A_{N-1} $(1 \leq A_i \leq 10^9)$, specifying the number of rabbits dwelling in each hutch, starting from the beginning of the row. The following Q lines represent the list of segments available to Leonardo. Each segment is specified on one line which contains two integers P, R $(0 \leq P < R \leq N-1)$ denoting the number of the first and the last hutch in the hutch segment. The hutches in the row are numbered by consecutive integers 0, 1, …, $N-1$.

Output

Output Q lines, each with a single integer specifying the number of days Leonardo can spend on the farm performing his tricks, if the respective segment in the input list will be available to him.

Example

Input	Output

`8 4`
`7 3 2 10 3 4 1 2`
`0 5`
`1 2`
`4 5`
`4 7`
	`2`
	`1`
	`0`
	`2`

Problem K. Fellow Sheep

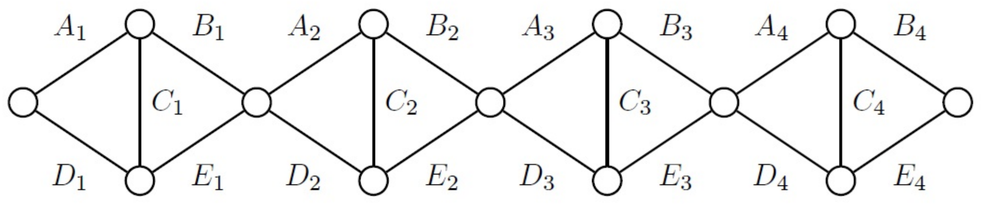
Source file name:	Sheep.c, Sheep.cpp, Sheep.java, Sheep.py
Input:	`Standard`
Output:	`Standard`

The sheep are fleeing from the wolf that has entered their pasture. To save themselves, they must escape through fenced paths designated for sheep shearing. The paths are connected to each other and eventually lead to a single exit into the farmyard, where the sheep that escape the wolf will be safe. The paths are arranged in a special formation, consisting of several segments of the same shape, with each segment containing 5 paths. The entry point of each segment, except for the first, is also the exit point of the previous segment.

The entry point of the first segment is in the pasture, and the exit point of the last segment leads into the farmyard. The sheep always run in the same direction along one path, and it never happens that two sheep run along the same path in opposite directions. The sheep cannot jump over the fence of the path. The sheep's escape is further complicated by the fact that halfway along each path there is an automatic gate that allows a certain number of sheep to pass through before closing itself, making the path impassable for other sheep. However, the wolf can jump over the gates on the paths, making him extremely dangerous. We assume (and the sheep assume as well) that the wolf can catch only those sheep which are no longer able to reach the farmyard due to the closed gates.

Determine how many sheep can escape from the wolf at most, given that for each gate it is known how many sheep it will let through.

The diagram shows the paths arranged in a formation of 4 segments.



Input

The first input line contains a single integer N ($1 \leq N \leq 10^5$), the number of segments in the path formation. Each of the next N lines contains 5 space separated integers A_i, B_i, C_i, D_i, E_i ($1 \leq A_i, B_i, C_i, D_i, E_i \leq 10^8$), the numbers of sheep which can pass through the corresponding gate in the i-th segment before the gate closes. The letters A, B, C, D, E correspond to the notation in the image above. Segments are linked in the same order as they appear in the input. The entry of the first segment is in the pasture, the last segment leads into the farmyard.

Output

Output a single integer, the maximum number of sheep which can escape the wolf.

Example

Input	Output

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

Problem L. Watchdogs

Source file name:	Watchdogs.c, Watchdogs.cpp, Watchdogs.java, Watchdogs.py
Input:	`Standard`
Output:	`Standard`

In a small town called Kocourkov, surrounded by hills and woods, a strange problem has appeared. The town, known for its cozy pubs and friendly people, had always been peaceful – until a problem with mice showed up.

But these weren't just any mice. They were smart, quick and seemed to be everywhere at the same time. They had taken over the town's favorite pubs, running through the rafters, nibbling on crumb, and causing a lot of trouble.

In Kocourkov, there are N places and exactly $N-1$ roads and it is possible to reach any place from any other place in Kocourkov via the roads. Each road connects a pair of places.

The town council held an emergency meeting in the largest pub U Nacpané Mysi. They noticed that the mice are running across the city in a very particular pattern. Each mouse has exactly two lairs scattered across the city and runs only between its two lairs. Each lair is located in one of the places. It has also been observed that the mice are vulnerable about halfway through their usual path. Formally, if mouse's lairs are in places A and B then the mouse is vulnerable at place C if and only if $|d(C, A) - d(C, B)| \leq 1$ and C is on the path from A to B. In this context, $d(X, Y)$ denotes the distance between places X and Y measured in the number of roads on a path from X to Y. Therefore, there may be more than one place where a mouse is vulnerable and a mouse might also be vulnerable in its own lair. After long discussion, the ultimate solution was proposed. The town will place watchcats around the town. A watchcat can get rid of a mouse if the watchcat is on a place where the mouse is vulnerable. One watchcat can get rid of any number of mice. Watchcats are lazy, each of them will always sit on its chosen place, and it will not move away from it.

As purchasing watchcats is not cheap, the town council wants to minimize the amount of watchcats to buy. Help them with this task.

Input

First line of input consists of two numbers N, K ($2 \leq N \leq 10^5,\ 0 \leq K \leq 10^5$), the number of places in Kocourkov, and the number of mice present in Kocourkov.

The next $N-1$ lines describe the road network in Kocourkov. Each of these lines contains two integers U, V ($0 \leq U, V \leq N-1$) indicating that the places U and V are connected by a direct road.

The next K lines describe the pairs of lairs of the mice. On each of these lines, there are two numbers P, Q ($0 \leq P, Q \leq N-1$) representing the fact that a mouse has lairs in the places P and Q. It is guaranteed that $P \neq Q$.

Output

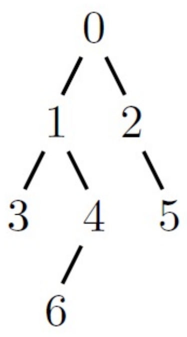
Output a single number representing the minimal number of watchcats to distribute in Kocourkov which will guarantee that every mice is eventually caught.

Example

Input	Output

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

Explanation



In the example input, it is optimal to place the watchcats at places 4, 1, 2. The first mouse is caught by the cat at place 4. The second and the third mouse are caught by the cat at place 1. The last two mice are caught by the cat at place 2.