Problem A. Fancy Antiques

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

You are hosting a fancy party for fancy friends. And, like any fancy party, you need to buy some fancy antiques to put up around the venue (your house).

There is a set of n fancy antiques that you need to buy. And there is a set of m fancy antique shops in the city. Because these antiques are extremely rare, each fancy antique can only be found at a single fancy antique shop. However, the fancy antique shops can also sell "knock-off" (duplicate) versions of some of the antiques. And of course, for any fancy antique, there is only a single fancy antique shop in the city that holds a knock-off version of that antique (this is to maintain the rareness of the antiques). The shop that sells the original is not always the same shop that holds the knock-off.

It turns out that even though you can tell the difference, most people cannot tell the original version from the knock-off version of any given antique. And, because the shops can get away with it, sometimes the knock-off is more expensive than the original! Since the party is tomorrow, you only have time to visit k shops. You would like to buy one version (either the original or the knock-off) of each of the n antiques.

Suppose that there are three shops, and three antiques we would like to buy.

- Antique #1 sells for 30 at shop #1. Its knockoff sells for 50 at shop #2.
- Antique #2 sells for 70 at shop #2. Its knockoff sells for 10 at shop #3.
- Antique #3 sells for 20 at shop #3. Its knockoff sells for 80 at shop #1.

Suppose you only have time to go to two shops. You can go to shops 1 and 3. You can buy the original of antique 1 with cost 30 at shop 1, the original of antique 3 with cost 20 at shop 3, and the knock-off of antique 2 at shop 3 with cost 10. The total cost to buy these items is 60, which is the minimum possible.

If you only have time to visit one shop, then it is impossible. You cannot buy a version of all three items by visiting a single shop.

Given the costs of the antiques/knock-offs at the shops, what is the minimum total cost to buy one version of each antique?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input will consist of three space-separated integers: n, m, and k ($1 \leq n \leq 100$, $1 \leq k \leq m \leq 40$). The next n lines will each have four space separated integers, a, p, b and q, describing an antique, where:

- a is the index of the shop that sells the original version of the antique ($1 \leq a \leq m$)
- p is the price of the original version of the antique at shop a ($1 \leq p \leq 10^7$)
- b is the index of the shop that sells the knock-off version of the antique ($1 \leq b \leq m$)
- q is the price of the knock-off version of the antique at shop b ($1 \leq q \leq 10^7$)

Output

If it is possible to collect all of the antiques while visiting no more than k stores, then output the minimum cost. If it is not possible, output -1.

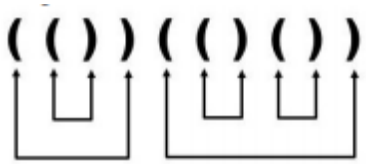
Example

Input	Output
3 3 2 1 30 2 50 2 70 3 10 3 20 1 80	60
3 3 1 1 30 2 50 2 70 3 10 3 20 1 80	-1

Problem B. Alternative Bracket Notation

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

Balanced closed bracket or parenthesis statements are ones where each opening bracket is matched with a closed bracket later in the string.



Notice how each closed parenthesis matches to the most recent unmatched open parenthesis.

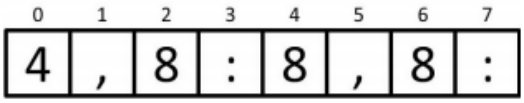
Define an alternative bracket notation as follows: each bracket pair corresponds to a header in the form of "*start,end*:" where *start* and *end* are indices of the new string itself! The index *start* is the index of the character immediately after the ':', and *end* is the index past the last header corresponding to the last bracket pair contained in this bracket pair. By taking a substring(*start, end*) of the new notation, you get an alternative bracket sequence describing all of the pairs of brackets contained by the brackets corresponding to the "*start,end*:"! Since an empty pair of brackets has nothing inside, in their header, *start* and *end* will be the same.

Each index takes up as many characters in the string as they do when they are base 10 numbers. (For example, the index 42 will take up 2 characters). The indices in the new string start from 0. All of the indices found in the alternative bracket notation string are absolute indices from the beginning of the new string.

Consider this parenthetical statement: (())

Here is it, in our new, alternate bracket notation: 4, 8 : 8, 8 :

In this example, there are two sets of matching parenthesis, the outer one and the inner one. The outer one appears before the inner one, since the start bracket appears first. So, the header for the outer brackets will appear before the header for the inner bracket. The header 4, 8 : represents the outer bracket, while the header 8, 8 : represents the inner bracket. The substring from the 4th character to 7th character is 8, 8 :, which represents what is contained inside the outer bracket. Note that 5, 11 : 11, 11 : could also be a legitimate alternate notation, but we want the shortest one, which is why 4, 8 : 8, 8 : is the correct answer.



Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The input will consist of a single line, containing a string s, which consists only of open and closed parentheses. The string s will be between 2 and 4000 characters long. There will be no spaces. The string s is guaranteed to be balanced.

Output

Output the string s in our new alternative bracket notation. If there's more than one way to represent s

in the new notation, choose the shortest representation, which will be unique.

Example

Input	Output
(())	4,8:8,8:
()	4,4:
((()))(()))()	5,29:11,17:17,17:23,29:29,29:35,35:

Problem C. Greetings!

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

Your greeting card company makes unique greeting cards. The sizes of these greeting cards vary widely because of the whims of card designers. There are a lot of different types of cards, and each has a specific quantity that you need to manufacture.

Your job is to determine what envelopes to order for these greeting cards. You have a strict limit on the different number of different sizes of envelopes, which may be less than the number of distinct sizes of cards. You need to have envelopes so that every card fits in some envelope, possibly with room to spare, and the amount of waste paper is minimized. Measure the waste paper by the area of the envelope that is in excess of the area of the card, for each card. For example, a 10×4 card in a 10×4 envelope has no wasted paper, but a 10×4 card in a 12×5 envelope has waste of 20. You may not rotate the cards to fit them in the envelopes better.

Suppose that you have 5 types of cards: 10×10 (5 of these), 9×8 (10 of these), 4×12 (20 of these), 12×4 (8 of these), and 2×3 (16 of these).

Now, suppose that you can only buy one type of envelope. Since all cards have to fit in that one envelope size, the smallest envelope size you can use is 12×12, with an area of 144. The wastes by each type of card are $144 - 10 \cdot 10 = 44$, $144 - 9 \cdot 8 = 72$, $144 - 4 \cdot 12 = 96$, $144 - 12 \cdot 4 = 96$, and $144 - 2 \cdot 3 = 138$, respectively. The total waste is $44 \cdot 5 + 72 \cdot 10 + 96 \cdot 20 + 96 \cdot 8 + 138 \cdot 16 = 5836$.

Suppose that you can buy 2 types of envelopes. The best you can do is to put the 10×10, 9×8 and 12×4 cards in 12×10 envelopes, and the 4×12 and 2×3 cards in 4×12 envelopes. That adds up to waste of 1828.

If you can buy 5 types of envelopes, then you can match one envelope type to each card type, and there's no waste!

Given a list of card types and the number of types of envelopes you can buy, what is the smallest amount of wasted paper you can achieve?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of the input will consist of two space-separated integers n and k ($1 \leq n, k \leq 15$), where n is the number of different types of cards, and k is the maximum number of types of envelopes you can order. Each of the following n lines will consist of three integers, describing a type of card. The integers are w, h and q ($1 \leq w, h, q \leq 10^4$), where w is the width of the cards of this type, h is the height of the cards, and q is the quantity of cards of this type.

Output

Output a single integer, representing the smallest possible total amount of wasted paper.

Example

Input	Output
5 1 10 10 5 9 8 10 4 12 20 12 4 8 2 3 16	5836
5 2 10 10 5 9 8 10 4 12 20 12 4 8 2 3 16	1828
5 5 10 10 5 9 8 10 4 12 20 12 4 8 2 3 16	0

Problem D. Programming Team

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

UpCoder is looking to assign their best employees to a team tasked with designing their new and improved website, and they're looking to you to help them form the team. There are n potential candidates. The CEO is employee number 0, and the candidates are all assigned employee numbers ranging from 1 through n. Each candidate is recommended by an employee with a smaller employee number. Each candidate can be described by three numbers (in addition to their employee number): their negotiated salary, their expected productivity, and the number of the employee who recommended them.

You would like to assign exactly k candidates out of the n total candidates to the team. The total value that you can get from these candidates is the sum of their productivities divided by the sum of their salaries. Note that you may only assign a candidate to the team if their recommender is also part of the team, or is the CEO. So, at least one candidate that you assign needs to have the CEO as a reference. The CEO handles the business aspect of the company, so s/he will not be counted as part of the k candidates chosen for the team.

Find the maximum total value your team can provide given these constraints.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of the input will consist of two space separated integers k and n ($1 \leq k \leq n \leq 2500$), where k is the size of the team you must form, and n is the total number of employee candidates. Each of the following n lines will hold three space-separated integers describing an employee. Employee 1 will be described first, then employee 2, and so on. The three integers are s, p and r, where s ($1 \leq s \leq 10000$) is the employee's salary, p ($1 \leq p \leq 10000$) is the employee's productivity, and r ($0 \leq r < i$) is the employee number of the employee who recommended this candidate (where i is the employee number of this candidate).

Output

Output a single real number, which represents the maximum total value you can achieve forming a team of k employees, subject to the constraints of the problem. Output this number to exactly three decimal places, rounded (standard 5 ↑ /4 ↓ rounding).

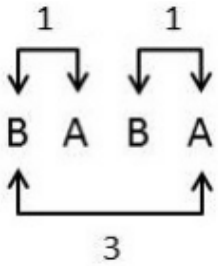
Example

Input	Output
1 2 1000 1 0 1 1000 1	0.001
2 3 1 100 0 1 200 0 1 300 0	250.000

Problem E. K-Inversions

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

You are given a string s consisting only of upper case letters A and B. For an integer k, a pair of indices i and j $(1 \leq i < j \leq n)$ is called a k-inversion if and only if $s[i] = B$, $s[j] = A$ and $j - i = k$.
Consider the string **BABA**. It has two 1-inversions and one 3-inversion. It has no 2-inversions.



For each k between 1 and $n - 1$ (inclusive), print the number of k-inversions in the string s.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The input will consist of a single line with a string s, which consists of only upper case **A**s and **B**s. The string s will be between 1 and 1000000 characters long. There will be no spaces.

Output

Output $n-1$ lines, each with a single integer. The first line's integer should be the number of 1-inversions, the second should be the number of 2-inversions, and so on.

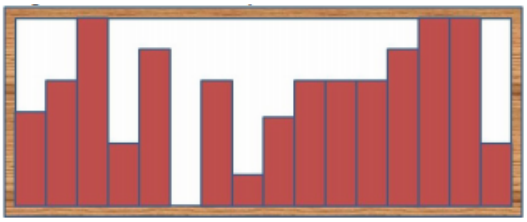
Example

Input	Output
BABA	2 0 1
BBBBBAAAAA	1 2 3 4 5 4 3 2 1

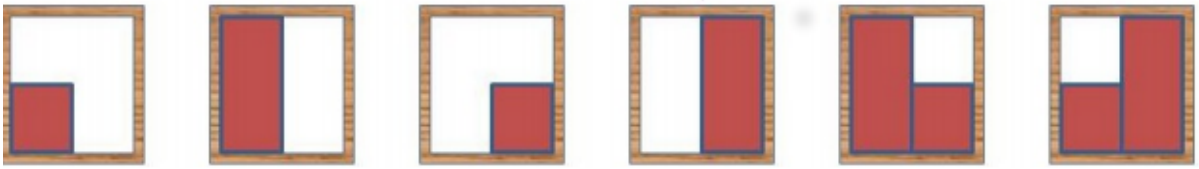
Problem F. Mountain Scenes

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

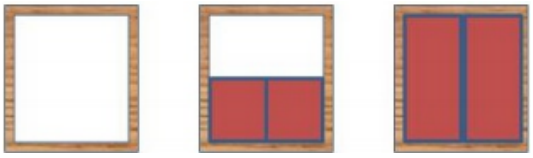
An artist begins with a roll of ribbon, one inch wide. She clips it into pieces of various integral lengths, then aligns them with the bottom of a frame, rising vertically in columns, to form a mountain scene. A mountain scene must be uneven; if all columns are the same height, it's a plain scene, not a mountain scene! It is possible that she may not use all of the ribbon.



If our artist has 4 inches of ribbon and a 2×2 inch frame, she could form these scenes:



She would not form these scenes, because they're plains, not mountains!



Given the length of the ribbon and the width and height of the frame, all in inches, how many different mountain scenes can she create? Two scenes are different if the regions covered by ribbon are different. There's no point in putting more than one piece of ribbon in any column.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The input will consist of a single line with three space-separated integers n, w and h, where n ($0 \leq n \leq 10000$) is the length of the ribbon in inches, w ($1 \leq w \leq 100$) is the width and h ($1 \leq h \leq 100$) is the height of the frame, both in inches.

Output

Output a single integer, indicating the total number of mountain scenes our artist could possibly make, modulo $10^9 + 7$.

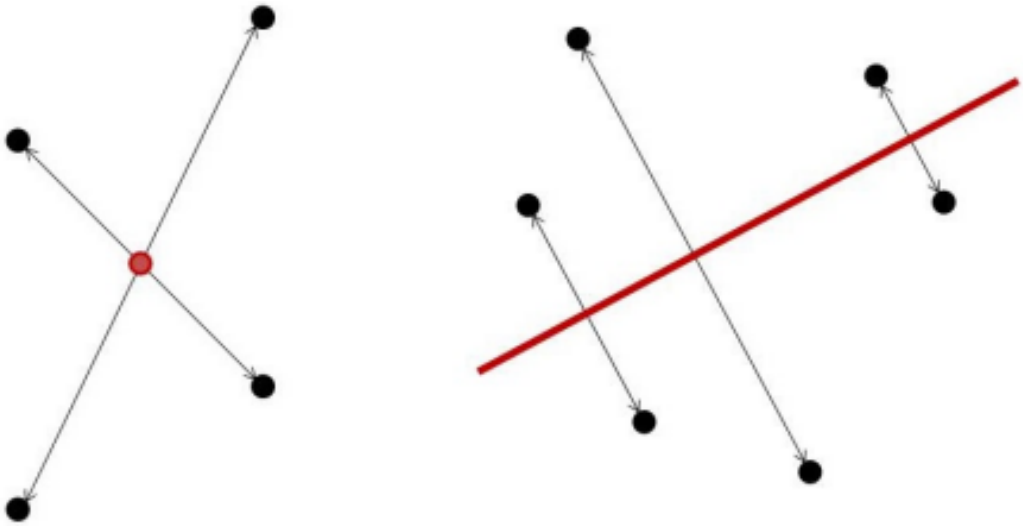
Example

Input	Output
25 5 5	7770
15 5 5	6050
10 10 1	1022
4 2 2	6

Problem G. Symmetry

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

You are totally bored with nothing to do. You notice a pattern of spots on the wall in front of you and begin to dwell on them. There is no obvious pattern of symmetry. With time this becomes very grating, and you contemplate adding more spots to satisfy your quest for balance. For this exercise you are to resolve this situation with a computer program.



Given an array of spots with coordinates in the range from -20000 to 20000, determine the fewest additional spots needed to generate a pattern with some symmetry. The symmetry can be around a point or across a line. If the symmetry is around a point, the point does not need to be a spot in the data, or even a point with integral coordinates. If the symmetry is across a line, the line may be at any angle. The coordinates of the additional spots may or may not be within the -20000 to 20000 limits.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input will consist of a single integer n ($1 \leq n \leq 1000$) indicating the number of spots. Each of the next n lines will hold two space-separated integers x and y ($-20000 \leq x, y \leq 20000$), which are the coordinates of a spot. The locations of all spots are guaranteed to be unique.

Output

Output a single integer, indicating the smallest number of spots which need to be added so that all of the spots are symmetric about some point, or about some line.

Example

Input	Output
4 0 0 1000 0 0 1000 1000 1000	0
11 0 0 70 100 24 200 30 300 480 400 0 100 0 200 0 400 100 0 300 0 400 0	6

Problem H. Jewel Thief

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

The grand museum has just announced a large exhibit on jewelry from around the world. In the hopes of his potential future prosperity, the world-renowned thief and master criminal Edward Terrenando has decided to attempt the magnum opus of his career in thievery.

Edward is hoping to purloin a large number of jewels from the exhibit at the grand museum. But alas! He must be careful with which jewels to appropriate in order to maximize the total value of jewels stolen.

Edward has k knapsacks of size 1, 2, 3, up to k, and would like to know for each the maximum sum of values of jewels that can be stolen. This way he can properly weigh risk vs. reward when choosing how many jewels to steal. A knapsack of size s can hold items if the sum of sizes of those items is less than or equal to s. If you can figure out the best total value of jewels for each size of knapsack, you can help Edward pull off the heist of the century!

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input will consist of two space-separated integers n and k, where n ($1 \leq n \leq 1000000$) is the number of jewels in the exhibit, and k ($1 \leq k \leq 100000$) is the maximum size of knapsack available to Edward. The next n lines each will describe a jewel. Each line will consist of two space-separated integers s and v, where s ($1 \leq s \leq 300$) is the size of the jewel, and v ($1 \leq v \leq 10^9$) is its value. Each jewel can only be taken once per knapsack, but each knapsack is an independent problem.

Output

Output k integers separated by whitespace. The first integer should be the maximum value of jewels that will fit in a knapsack of size 1. The second should be the maximum value of jewels in a knapsack of size 2, and so on.

Example

Input	Output
4 9 2 8 1 1 3 4 5 100	1 8 9 9 100 101 108 109 109
5 7 2 2 3 8 2 7 2 4 3 8	0 7 8 11 15 16 19
2 6 300 1 300 2	0 0 0 0 0 0

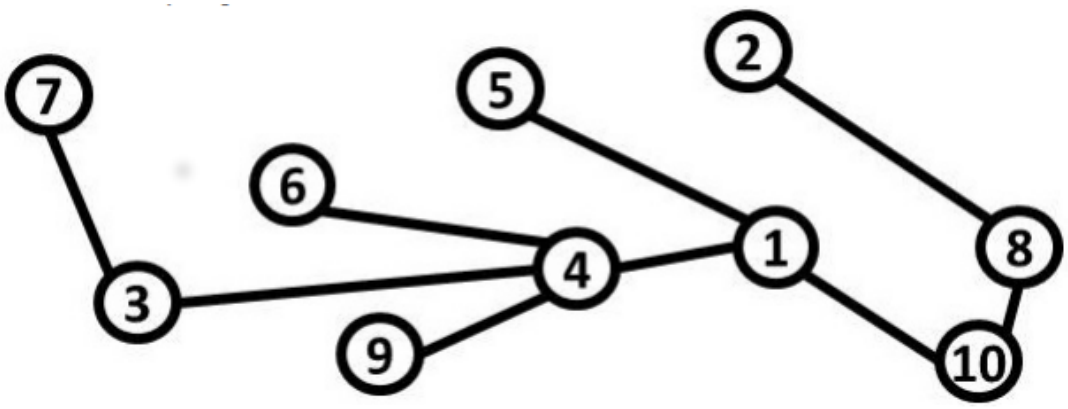
Problem I. Tourists

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

In Tree City, there are n tourist attractions uniquely labeled 1 to n. The attractions are connected by a set of $n-1$ bidirectional roads in such a way that a tourist can get from any attraction to any other using some path of roads.

You are a member of the Tree City planning committee. After much research into tourism, your committee has discovered a very interesting fact about tourists: they LOVE number theory! A tourist who visits an attraction with label x will then visit another attraction with label y if $y > x$ and y is a multiple of x. Moreover, if the two attractions are not directly connected by a road the tourist will necessarily visit all of the attractions on the path connecting x and y, even if they aren't multiples of x. The number of attractions visited includes x and y themselves. Call this the length of a path.

Consider this city map:



Here are all the paths that tourists might take, with the lengths for each:

$$1 \to 2 = 4, \quad 1 \to 3 = 3, \quad 1 \to 4 = 2, \quad 1 \to 5 = 2, \quad 1 \to 6 = 3, \quad 1 \to 7 = 4,$$
$$1 \to 8 = 3, \quad 1 \to 9 = 3, \quad 1 \to 10 = 2, \quad 2 \to 4 = 5, \quad 2 \to 6 = 6, \quad 2 \to 8 = 2,$$
$$2 \to 10 = 3, \quad 3 \to 6 = 3, \quad 3 \to 9 = 3, \quad 4 \to 8 = 4, \quad 5 \to 10 = 3$$

To take advantage of this phenomenon of tourist behavior, the committee would like to determine the number of attractions on paths from an attraction x to an attraction y such that $y > x$ and y is a multiple of x. You are to compute the **sum** of the lengths of all such paths. For the example above, this is: $4+3+2+2+3+4+3+3+2+5+6+2+3+3+3+4+3 = 55$.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input will consist of an integer n ($2 \leq n \leq 200000$) indicating the number of attractions. Each of the following $n-1$ lines will consist of a pair of space-separated integers i and j ($1 \leq i < j \leq n$), denoting that attraction i and attraction j are directly connected by a road. It is guaranteed that the set of attractions is connected.

Output

Output a single integer, which is the sum of the lengths of all paths between two attractions x and y such that $y > x$ and y is a multiple of x.

Example

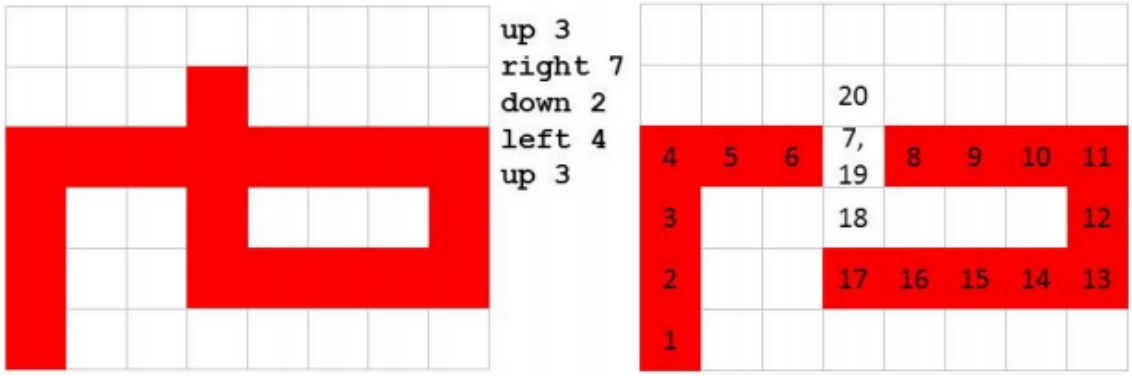
Input	Output
10 3 4 3 7 1 4 4 6 1 10 8 10 2 8 1 5 4 9	55

Problem J. Whiteboard

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

Mr. Turtle loves drawing on his whiteboard at home. One day when he was drawing, his marker dried out! Mr. Turtle then noticed that the marker behaved like an eraser for the remainder of his drawing.

Mr. Turtle has a picture in his head of how he wants his final drawing to appear. He plans out his entire drawing ahead of time, step by step. Mr. Turtle's plan is a sequence of commands: *up*, *down*, *left* or *right*, with a distance. He starts drawing in the bottom left corner of his whiteboard. Consider the 6×8 whiteboard and sequence of commands in the first diagram. If the marker runs dry at timestep 17, the board will look like the second diagram (the numbers indicate the timestep when the marker is at each cell). Note that it will make a mark at timestep 17, but not at timestep 18.



Mr. Turtle wants to know the earliest and latest time his marker can dry out, and he'll still obtain the drawing in his head. Can you help him? Note that timestep 0 is the moment before the marker touches the board. It is valid for a marker to dry out at timestep 0.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The input will start with a line with 3 space-separated integers h, w and n ($1 \leq h, w, n \leq 1000000$, $w \cdot h \leq 1000000$) where h and w are the height and width of the whiteboard respectively, and n is the number of commands in Mr. Turtle's plan.

The next h lines will each consist of exactly w characters, with each character being either '#' or '.' . This is the pattern in Mr. Turtle's head, where '#' is a marked cell, and '.' is a blank cell. The next n lines will each consist of a command, of the form "direction distance", with a single space between the direction and the distance and no other spaces on the line. The direction will be exactly one of the set `{up, down, left, right}`, guaranteed to be all lower case. The *distance* will be between 1 and 1000000 inclusive. The commands must be executed in order. It is guaranteed that no command will take the marker off of the whiteboard.

Output

Output two integers, first the minimum, then the maximum time that can pass before the marker dries out, and Mr. Turtle can still end up with the target drawing. Neither number should be larger than the last timestep that the marker is on the board, so if the marker can run to the end and still draw the target drawing, use the last timestep that the marker is on the board. If it's not possible to end up with the target drawing, output -1 -1.

Example

Input	Output
6 8 5 ....#.... ######## #..#...# #..##### #..... up 3 right 7 down 2 left 4 up 3	20 20
6 8 5 .... ###.#### #.....# #..##### #..... up 3 right 7 down 2 left 4 up 3	17 17
3 3 2 ... .#. ... up 2 right 2	-1 -1
2 2 4 .. .. up 1 right 1 down 1 left 1	0 1

Problem K. YATP

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

This is Yet Another Tree Problem. You are given a tree, where every node has a penalty and every edge has a weight. The cost of a simple path between any two nodes is the sum of the weights of the edges in the path, plus the product of the penalties of the endpoint nodes. Note that a path can have 0 edges, and the cost of such a path is simply the square of the penalty of the node.

For each node, compute the smallest cost of any path starting at that node. The final answer is the sum of all of these minimum costs.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input will consist of a single integer n ($1 \leq n \leq 200000$), which is the number of nodes. The next line will consist of n space-separated integers p ($1 \leq p \leq 1000000$), which is the penalty of each node, in order. Each of the following $n-1$ lines will consist of three space-separated integers i, j and w ($1 \leq i \leq n$, $1 \leq j \leq n$, $i \neq j$, $1 \leq w \leq 1000000$), specifying an edge between nodes i and j with weight w.

Output

Output a single integer, which is the sum of all of the lowest cost paths for each node

Example

Input	Output
5 9 7 1 1 9 3 2 8 5 2 10 4 3 10 2 1 2	63