Division 1

Hosted by:

College of Charleston

Florida International University

Kennesaw State University

University of West Florida

Ducks in a Row

Sri is playing a game with ducks, geese, and a magic wand. First he puts all his ducks in a row. Next his friend Srinivas inserts some geese between the ducks at different places. Sri can then use his magic wand to flip some of the ducks and geese.

Each use of his wand can be defined formally:

1) Sri can select some contiguous sequence of ducks and geese.
2) All birds that were ducks before using the wand are now geese.
3) All birds that were geese before using the wand are now ducks.

Sri has an objective to succeed at the game. He must turn the row into at least ***k*** maximal runs of consecutive ducks of length at least ***n***. A maximal run is a sequence of ducks that does not have a duck immediately to its left or right. For example, the following row of birds has 4 maximal runs of ducks of lengths 2, 3, 3, and 1, respectively:

D D G G G G D D D G D D D G D

Sri needs to find the minimum number of wand uses to meet his objective. There can be other maximal runs of consecutive ducks at the end of the game, maybe some of length <***n***, but there must be at least ***k*** of at least length ***n***.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of exactly two lines. The first line will contain two integers ***n*** and ***k*** ($1 \le n, k \le 2{,}000$), where ***n*** is the minimum length of each sequence of ducks that Sri desires, and ***k*** is the minimum number of sequences of ducks that Sri desires. The second line will contain a single string, ***s*** ($1 \le |s| \le 2{,}000$), consisting of only the capital letters **D** and **G**. They represent the row of birds before Sri starts using his magic wand, where **D** is a duck and **G** is a goose.

Output

Output a single integer, the minimum number of times he must use his wand to meet his desired property or **-1** if it is not possible.

Sample Input	Sample Output
2 2 DDDGD	1
2 3 GGDGGDGG	1

Exciting Finish!

You are watching a singing contest featuring ***n*** singers, and the final round has just finished. Each singer currently has a distinct number of points. It is time to determine the final rankings. The judges will give scores to the singers, so that the total points given in the final round across all singers is equal to some fixed number ***x***. More specifically, for each singer, the judges will choose points in the final round so that each is a positive integer, and the final round points for all singers sum to ***x***.

The judges would like to make the final results as exciting as possible. To do this, they will announce the final scores one by one, in nondecreasing order. If there are ties, they will announce the final score of the singer that has the smallest current score. After each score announcements, the rankings will get updated. The final results are *exciting*, if after each announcement of points in the final round, the singer in first place (with the highest number of points) changes. The judges want to avoid any sort of controversy, so they will assign the points so that at every point during the score announcements, there is a unique singer with the highest number of points. There may be ties in the final round scores, but never in the total points.

Consider three contestants (we'll call them A, B and C). Immediately before the final round, A has 3 points, B has 1 point, and C has 4 points. The judges have 12 points to assign in the final round.

Suppose in the final round the judges give A←2, C←3, and B←7. Then:

- A's score (2) is announced, A is the new leader (3+2=5)
- C's score (3) is announced, C is the new leader (4+3=7)
- B's score (7) is announced, B is the new leader (1+7=8)

That's an *exciting* result! The final round scores were announced in nondecreasing order and the leader changes after every announcement, including the first, since C was in the lead before the final round! And, the judges used all 2+3+7=12 points in the final round!

In this scenario, assigning final round points A←3, B←3, C←6 won't work, since there's a tie at the top after the first announcement (A and B both get 3 in the final round, but B is announced first, since B has the lower score before the final round.) The assignment C←1, B←5, A←6 is not *exciting* since the leader does not change after the first assignment.

In this scenario, there are exactly three distinct final rankings possible:

- C, B, A announced in order A←2, B←5, C←5
- B, C, A announced in order A←2, C←2, B←8 OR A←2, C←3, B←7
- C, A, B announced in order B←4, A←4, C←4

Help the judges determine the number of distinct rankings possible that can be attained by assigning points in the final round such that the final results are *exciting*. Two ways are different if the resulting final ranking is different, regardless of the final round points given by the judges.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two space-separated integers ***n*** ($1 \le n \le 12$) and ***x*** ($1 \le x \le 700$), where ***n*** is the number of singers, and ***x*** is the total number of points that the judges can allocate in the final round. The next line will contain ***n*** integers ***p*** ($1 \le p \le 700$) separated by spaces. These are the current points for each of the ***n*** singers.

Output

Output a single integer, which is the number of final rankings possible.

Sample Input	**Sample Output**
3 12 3 1 4	3
12 700 1 2 3 4 5 6 7 8 9 10 11 12	439084800
12 1 1 2 3 4 5 6 7 8 9 10 11 12	0
1 700 123	0

Flipping Out

Here is an interesting way of generating coin flips that looks random, but it's not! First, generate a set of *patterns* (consecutive runs of coin flips) that are either heads or tails. Then, to generate each new coin flip, look back at all of the past coin flips in order, and count the number of times that each pattern occurs. Add them all up, and if that number is even, then the next flip is tails, otherwise the next flip is heads.

Consider these three *patterns*:

```
HTH
THH
T
```

They will generate this sequence of flips: **THHTHTHT**…

1) **T** because at the start there are 0 occurrences of any of the *patterns*, and 0 is even.
2) **H** because there is 1 occurrence of the *pattern* **T**
3) **H** because there is still only 1 occurrence of the *pattern* **T**
4) **T** because there is one **T** and one **THH**, so 2 total
5) **H** because there are 2 **T**s and one **THH**, 3 total
6) **T** because there are 2 **T**s, one **THH** and one **HTH**, 4 total
7) **H** because there are 3 **T**s, one **THH** and one **HTH**, 5 total
8) **T** because there are 3 **T**s, one **THH** and 2 **HTH**s, 6 total (it's OK that the **HTH**s overlap)

Suppose that a single *pattern* was missing from the set of *patterns*. From a sequence of flips, can you determine how many options there are for the missing *pattern*? Note that the missing pattern cannot be a duplicate of one of the given patterns, and also cannot be empty.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing a single integer ***n*** ($1 \le n \le 100{,}000$), representing the number of *patterns*.

The next ***n*** lines will each contain a string consisting only of capital **T**s and capital **H**s. The first ***n***-1 of these strings are the *patterns*, and the last is a sequence of generated flips. The sum of the lengths of all ***n*** strings will be $\le 10^6$. All strings will be unique, and no strings will be empty.

Output

Output a single integer, which is the number of options for the missing *pattern*. Output **-1** if there are infinitely many options.

Sample Input	Sample Output
2 H HTTTT	0
1 THHHHH	1
5 H TTH HHTHT TH TTTHTT	1
5 TTH H T HT THTTHTTH	2

Jumping Haybales

Farmer John's cows have grown lazy and he would like to get them back in shape! He has decided that the best way to do this is by setting up haybales in the field and having the cows jump from the northwest corner of the field to the southeast corner. The field is an ***n*×*n*** square grid. Note that on a standard map, North is up, South is down, East is right and West is left.

A cow can only jump straight east or south, never west or north, or even southeast. They also have a limit ***k*** on how many cells they can jump. They can jump over haybales, empty spaces, or any combination, even if there are no haybales in between, but they cannot land on a haybale. Bessie wants to still be lazy and is interested in the minimum number of jumps to reach the southeast corner of the map from the northwest corner.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers ***n*** and ***k*** ($1 \le n,k \le 2{,}000$), where ***n*** is the size of one side of the square grid, and ***k*** is Bessie's limit on the number of cells she can jump. Each of the next ***n*** lines will contain a string with exactly ***n*** characters. Only the characters '**#**' (a haybale) and '**.**' (an empty space) will appear. The northwest and southeast corners of the field are guaranteed to be empty.

Output

Output a single integer, which is the minimum number of jumps Bessie needs to reach the southeast corner from the northwest corner, or **-1** if Bessie cannot make it.

Sample Input	Sample Output
4 2 .### #... .#.. #.#.	4
3 1 .#. .#. .#.	-1

Long Long Strings

To store DNA sequences your company has developed a general *LongLongString* class that can store strings with a theoretically unlimited number of characters. Individual character can be referenced by index. The leftmost character is at position 1. The class can execute simple programs with three basic operations:

- **insert**(***p***, ***c***) - inserts the character ***c*** at position ***p***. All characters past ***p*** are pushed to the right by 1.
- **delete**(***p***) – deletes the character at position ***p***. All character past ***p*** are pushed to the left by 1.
- **end** – ends the program

Your job is two write a program that compares two string editing programs and determines if they are *different*. They are *not different* if, when applied to **any** string, they produce identical results. Otherwise, they are *different*.

For example:

- [**delete**(1) **delete**(2) **end**] and [**delete**(3) **delete**(1) **end**] are *not different*
- [**delete**(2) **delete**(1) **end**] and [**delete**(1) **delete**(2) **end**] are *different*.
- [**insert**(1,X) **delete**(1) **end**] and [**end**] are *not different*.
- [**insert**(14,B) **insert**(14,A) **end**] and [**insert**(14,A) **insert**(15,B) **end**] are *not different*
- [**insert**(14,A) **insert**(15,B) **end**] and [**insert**(14,B) **insert**(15,A) **end**] are *different*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of exactly two programs, one after the other. Each program will end with an **end** statement, and each will be no longer than **2,000** instructions. Each line of each program will consist of exactly one of:

`I` ***p*** ***c***

or

`D` ***p***

or

`E`

Where ***p*** ($1 \le p \le 10^{10}$) is a position in the string, and ***c*** is a single capital letter (**A**..**Z**). `I` means **insert**, `D` means **delete**, and `E` means **end**.

Output

Output a single integer, **1** if the programs are *different*, and **0** of they are *not different*.

Sample Input	Sample Output
D 1 D 2 E D 3 D 1 E	0
D 2 D 1 E D 1 D 2 E	1
I 1 X D 1 E E	0
I 14 B I 14 A E I 14 A I 15 B E	0
I 14 A I 15 B E I 14 B I 15 A E	1

Move Away

Tommy has just completed college and is looking for his first job. A priority in his life is living close to his friends, but he wants to live as far away from his parents as possible.

You are given the locations of Tommy's friends and the maximum distance he would be willing to live away from each friend. You also know that Tommy's parents live at (0, 0) in the coordinate plane. Determine how far Tommy can live from his parents. (There will always be at least one point meeting these requirements.)

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with a single integer ***n*** ($1 \le n \le 50$), representing the number of friends Tommy has. The next ***n*** lines will each contain three integers: ***x***, ***y*** ($-1{,}000 \le x, y \le 1{,}000$) and ***d*** ($1 \le d \le 1{,}000$), representing the (***x***, ***y***) coordinate of his friend and the maximum distance ***d*** he is willing to live away from that friend.

Output

Output a single decimal number on a single line, equal to the maximum distance he can live from his parents while still being close enough to all of his friends. Output this number to exactly 3 decimal places, rounded.

Sample Input	Sample Output
4 1 0 1 0 1 1 -1 0 1 0 -1 1	0.000
2 -1 0 1000 2 0 1000	999.999

Rainbow Roads

Your city has decided to spice up its image – by painting its roads different colors!

Your city will paint a road the same uniform color between two intersections if there are no intersections in between (for our purposes, we'll refer to dead ends and cul-de-sacs as intersections), but along its full length, a road may be painted many different colors. Interestingly, there is exactly one path along its roads between any two intersections in the city.

The city council wants to label some intersections as *Super* intersections, and put up signs designating them so. They consider a path a *Rainbow* if there are no intersections along the path where the road in and the road out are the same color. An intersection is a *Super* intersection if the path from that intersection to every other intersection is a *Rainbow*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line of input containing a single integer n ($1 \le n \le 50{,}000$) which is the number of intersections. The intersections are numbered 1..n.

Each of the next n-1 lines will contain three integers, a, b and c ($1 \le a,b,c \le n$, $a \ne b$), which describe a road between intersection a and intersection b with color c. It is guaranteed that the given roads satisfy the constraint that there is exactly one path between any pair of intersections. The roads are two-way roads, so a road from a to b also goes from b to a.

Output

On the first line, output a single integer indicating the number of *Super* intersections. On the following lines, output a list of integers, one per line. These are the *Super* intersections. Print them in numerical order, smallest to largest.

Sample Input	Sample Output
8 1 3 1 2 3 1 3 4 3 4 5 4 5 6 3 6 7 2 6 8 2	4 3 4 5 6
8 1 2 2 1 3 1 2 4 3 2 7 1 3 5 2 5 6 2 7 8 1	0
9 1 2 2 1 3 1 1 4 5 1 5 5 2 6 3 3 7 3 4 8 1 5 9 2	5 1 2 3 6 7
10 9 2 1 9 3 1 9 4 2 9 5 2 9 1 3 9 6 4 1 8 5 1 10 5 6 7 9	4 1 6 7 9

Security Badges

You are in charge of the security for a large building. The building has a map, consisting of rooms, and doors between the rooms. Each door has a security code, which consists of a range of numbers, specified by a lower bound and an upper bound. Each employee has a uniquely numbered security badge. Only a security badge with a number within a door's range can go through that door.

Your boss wants a quick check of the security of the building. Given a starting room and a destination room, how many security badge numbers can go from the start to the destination?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing three integers integer ***n*** ($1 \le n \le 1{,}000$), ***m*** ($1 \le m \le 5{,}000$) and ***k*** ($1 \le k \le 10^9$), where ***n*** is the number of rooms, ***m*** is the number of doors, and ***k*** is the number of badges. The rooms are numbered 1..***n*** and the badges are numbered 1..***k***.

The next line will contain two integers, ***s*** and ***d*** ($1 \le s,d \le n$), which indicate the starting room and destination room.

Each of the next ***m*** lines will contain four integers, ***a***, ***b*** ($1 \le a,b \le n$, $a \ne b$), ***min*** and ***max*** ($1 \le min \le max \le k$) describing a door, where the door from room ***a*** to room ***b*** (and not back), and the badges range for the door is ***min***..***max***, inclusive.

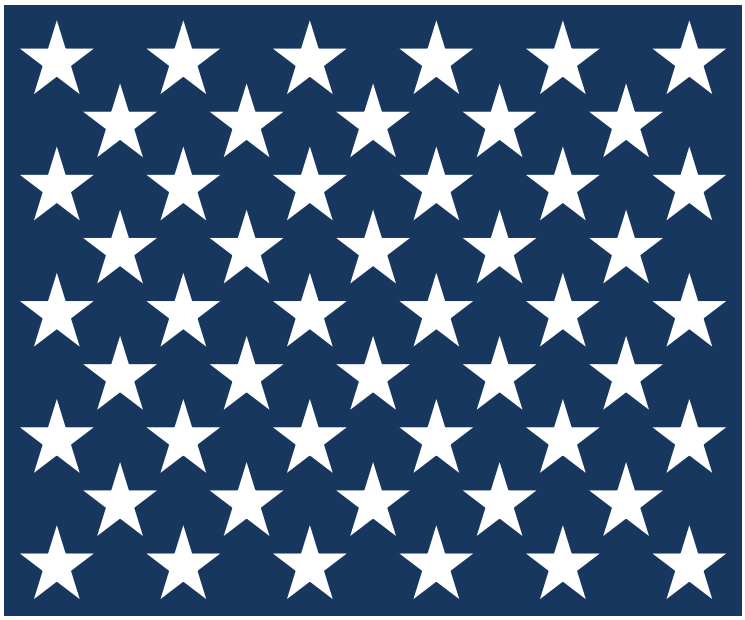
Output

Output a single integer, which is the number of badges that can go from the start room to the destination room.

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

Star Arrangements

The recent vote in Puerto Rico favoring US statehood has made flag makers very excited. An updated flag with 51 stars rather than the current 50 would cause a huge jump in US flag sales. The current pattern for 50 stars is five rows of 6 stars (30), interlaced with four offset rows of 5 stars (20).



This pattern has some appealing properties: adjacent rows differ by no more than one star. This star arrangement can be represented uniquely in a compact notation by the first two rows' star counts: **6 5**.

A 51-star flag can have three rows of 9 stars, interlaced with three rows of 8 stars (27 + 24 = 51), or **9 8**. If Guam were to also become a state, a 52-star flag could have 13 rows of 4 stars, or **13 13** (because there are 13 stars in each of the first 2 rows).

A visually appealing star field satisfies these conditions:

1) There are at least 2 rows of stars.
2) All odd numbered rows have the same number of stars.
3) All even numbered rows have the same number of stars.
4) The difference in the number of stars between any two adjacent rows is either always 0, or always 1
5) The first row cannot have fewer stars than the second, nor can it have only 1 star.

Given a number of states, describe all possible appealing star fields in compact notation.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with single integer ***n*** ($3 \le n \le 10^6$), indicating the number of stars.

Output

Output all possible star field arrangements, in compact notation. Output them one arrangement per line, with a single space between the two integers. Output them in increasing order of the first integer. If the first integer of two arrangements is the same, output them in increasing order of the second integer.

Sample Input	**Sample Output**
3	2 1
50	2 1 2 2 3 2 5 4 5 5 6 5 10 10 13 12 17 16 25 25
51	2 1 3 3 9 8 17 17 26 25
52	2 2 4 4 7 6 13 13 26 26

Treasure Map

You have found a treasure map! The map leads you to several gold mines. The mines each produce gold each day, but the amount of gold that they produce diminishes each day. There are paths between the mines. It may take several days to go from one mine to another. You can collect all of the day's gold from a mine when you are there, but you have to move on, you cannot stay for multiple days at the same mine. However, you can return to a mine after leaving it.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers ***n*** ($2 \le n \le 1{,}000$) and ***m*** ($1 \le m \le 1{,}000$), where ***n*** is the number of mines, and ***m*** is the number of paths.

The next ***n*** lines will each describe a mine with two integers, ***g*** ($1 \le g \le 1{,}000$) and ***d*** ($1 \le d \le 1{,}000$), where ***g*** is the amount of gold mined on day 1, and ***d*** is the amount by which the gold haul diminishes each day. For example, if ***g***=9 and ***d***=4, then on day 1, the mine produces 9, on day 2 it produces 5, on day 3 it produces 1, and from day 4 on, it produces 0 (the mines cannot produce negative amounts of gold). The mines are numbered 1..***n*** in the order that they appear in the input, and you start at mine 1 on day 1.

The next ***m*** lines will each describe a path with three integers, **a**, ***b*** ($1 \le a < b \le n$) and ***t*** ($1 \le t \le 100$), where the path goes from mine ***a*** to mine ***b***, and takes ***t*** days to traverse. The paths go in both directions, so that a path that goes from ***a*** to ***b*** can also be used to go from ***b*** to ***a***.

Output

Output a single integer, which is the maximum amount of gold that you can collect.

Sample Input	Sample Output
2 1 10 1 10 2 1 2 1	42
3 2 10 5 3 1 5 1 1 2 1 2 3 1	16
3 3 20 6 8 2 6 1 1 2 1 2 3 1 1 3 1	38
2 1 1 1 10 5 1 2 2	1

Unsatisfying

Usually a computer scientist tries to satisfy some constraints. This time you will try to make some logical statements unsatisfiable.

You will be given a list of logical statements of the following form:

$$p_1 \mid p_2$$
$$\sim p_2 \mid p_3$$
$$p_3 \mid \sim p_4$$

Here '**|**', the disjunctive statement, stands for logical **OR** (the result is **TRUE** if either proposition is **TRUE**, possibly both). '**~**' is negation, forcing the value to be the opposite truth value.

To satisfy a list of logical statements, you must assign truth values (**TRUE** or **FALSE**) to each variable such that all the given statements result as **TRUE**. Your task is to add disjunctive statements to the list to make the list of statements unsatisfiable. But you cannot use the negation symbol!

All disjunctive statements (both those given and ones you add) must have exactly 2 terms. The ones given can use negation, but the ones added cannot.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers ***n*** and ***m*** ($1 \le n,m \le 2{,}000$), where ***n*** is the number of variables and ***m*** is the number of disjunctions. The variables will be numbered 1..***n***.

Each of the next ***m*** lines will contain two integers ***a*** and ***b*** ($1 \le |a|,|b| \le n$), representing the subscript in the variable. A negative value is the negated version of that variable.

Output

Output a single integer, which is the minimum number of disjunctive clauses to add to make the list unsatisfiable. If it is not possible, output -**1**.

Sample Input	Sample Output
2 1 1 2	-1
4 5 1 2 -1 -3 -2 3 3 -4 -2 -3	1