Problem A. Number Maximization

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

Everybody likes larger numbers since they come across as more powerful, more money, etc.

Given a number, determine the largest number that can be formed using the exact same digits as the given number.

Input

There is only one input line; it contains an integer between 0 and 999.999 (inclusive). Assume that the input number will not have leading 0's. Note, however, that the input can be just the value 0.

Output

Print the largest number that can be formed using the exact same digits as the input number.

Example

Input	Output
2897	9872
15010	51100
99	99

Problem B. Simplified Calendar System

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

Consider a simplified calendar system where each year is 12 months and each month is 30 days, i.e., each year is 360 days. This sure makes the math easy, or does it?

Given a date and what day of the week it is, determine what day of the week a second date is. The second date will be after (timewise) the first date.

Input

There are two input lines. The first input line contains four integers:

- d ($1 \leq d \leq 30$), providing what day of the month the first date is,
- m ($1 \leq m \leq 12$), providing what month of the year the first date is,
- y ($1000 \leq y \leq 2999$), providing what year the first date is, and
- n ($1 \leq n \leq 7$), providing what day of the week the given date is (1 represents Sunday, 2 represents Monday, ..., 7 represents Saturday).

The second input line provides the second date; it contains three integers (similar to the first three integers of the first input line). Again, the second date will be later than the first date (timewise).

Output

Print an integer (between 1 and 7, inclusive) indicating what day of the week the second date is.

Example

Input	Output

`20 11 2021 7`
`29 11 2021`
	`2`

`20 10 1999 5`
`15 4 2002`
	`4`

Problem C. Letter Frequency

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

The most common consonants in English are `R`, `T`, `N`, `S`, and `L`. The most common vowels are `E`, `A`, and `I`. It is definitely fun to find the most common letters.

Given a set of words, find the letter that occurs the most in each position. That is, the most frequent letter at Position 1 when considering all the words, the most frequent letter at Position 2 when considering all the words, etc. If more than one letter is the most frequent for a particular position (i.e., ties for max at that position), print those letters in alphabetical order.

Input

The first input line contains an integer, n ($1 \leq n \leq 20$), indicating the number of words. Each of the next n input lines contains a word. Assume that each word contains 1-30 lowercase letters and it starts in column one.

Output

Print each position and the letter(s) that occur the most in that position, following the format illustrated in Example Output. Note that each output line starts with a number, immediately followed by a colon (‘`:`’), followed by a space, followed by a letter. If there are multiple letters for an output line, they are separated by exactly one space.

Example

Input	Output
5 this is an example ink	1: i 2: n 3: a i k 4: m s 5: p 6: l 7: e
3 this is longlonglong	1: i l t 2: h o s 3: i n 4: g s 5: l 6: o 7: n 8: g 9: l 10: o 11: n 12: g

Problem D. Pseudo Pseudo Random Numbers

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

It turns out that if numbers were truly random, then each possible bit string (string of 0's and 1's) of length n would be equally likely. For example, 111111 would be just as likely as 010110 to occur. Unfortunately, most people believe that any time they see the same bit over and over again, that the process can't be truly random.

You are in charge of generating random bit strings of length n for use in a video game. However, the producer of the game has asked you to remove all possibilities where there are **more than *k*** 0's or 1's in a row. For example, if $n = 4$ and $k = 2$, then the 10 valid bit strings would be 0010, 0011, 0100, 0101, 0110, 1001, 1010, 1011, 1100, and 1101 (the other 6 strings of 4 bits either have more than two 0's in a row or more than two 1's in a row, so they are not valid).

Given the values of n and k, determine the number of n bit strings that do not contain any runs of 0's or 1's of length greater than k.

Input

There is only one input line; it contains two integers: n ($2 \leq n \leq 20$), indicating the length of the bit string for the problem, and k ($1 \leq k \leq n$), indicating the maximal length of a run of 0's or 1's for the bit strings to be created.

Output

Print the number of valid bit strings of length n that do not contain any runs of the same bit of length greater than k.

Example

Input	Output
4 2	10
5 1	2
20 20	1048576

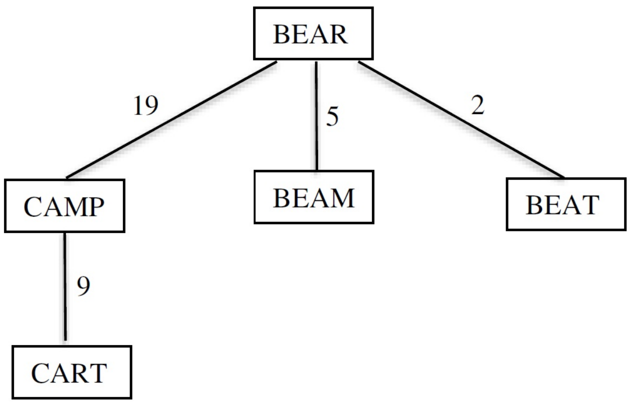
Problem E. Word Tree

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

Given a set of n words, each of the same length, we can form a tree structure between the words by connecting pairs of words with an edge. In particular, we must add exactly $n - 1$ edges and make sure that there is a single simple path from every word to every other word.

We define the cost of an edge between two words as the sum of the costs between each pair of corresponding letters of the two words. We define the cost between two letters as being the absolute value of the difference between their Ascii values. For example, the cost of an edge between "`CAMP`" and "`BEAR`" would be $1 + 4 + 12 + 2 = 19$ because |'`C`' - '`B`'| = 1, |'`A`' - '`E`'| = 4, |'`M`' - '`A`'| = 12, and |'`P`' - '`R`'| = 2.

Here is an illustration of a word tree with the words `BEAM`, `BEAR`, `BEAT`, `CAMP` and `CART`:



Finally, because we like only connecting words that are "similar", our goal is to minimize the value of the maximum edge cost of the tree. In the example above, it is impossible to connect the words in a tree structure where each edge has a cost of less than 19.

Note that there are no constraints on the tree structure, i.e., any word can be listed as the child of any other node. For example, it does not have to be a binary search tree. The structure must, of course, be a tree and the cost is as defined above.

Given a set of n words, each consisting of k uppercase letters, of all possible word trees that could be formed with those words, determine the minimum possible value of the maximum edge cost.

Input

The first input line contains two integers: n ($2 \leq n \leq 1000$), indicating the number of words, and k ($1 \leq k \leq 20$), indicating the length of each of the words. Each of the following n input lines contains one of the words. These words are guaranteed to be distinct, consist of exactly k uppercase letters, and start in column one.

Output

Print the minimum value of the maximum edge cost over all possible word trees for the set of input words.

Example

Input	Output
5 4 BEAM BEAR BEAT CAMP CART	19
6 3 BAN BAR BAT CAN CAP CAR	2

Problem F. House Prices Going Up

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

The house prices have been going up and the Virtual County Tax Collection Agency needs help to keep track of prices.

We assume the houses in Virtual County (VC) are numbered 1 through n, i.e., the house numbers are 1, 2, 3, ..., n. VC keeps track of the house prices as they go up and would like to know the total price for different sections of the county, e.g., the total price for houses 4 through 15.

Input

The first input line contains an integer, n ($1 \leq n \leq 5 \cdot 10^5$), indicating the number of houses in VC. Each of the next n input lines contains an integer (between 1 and 10^9, inclusive) indicating the initial price for a house; first integer is the price for the first house, second integer is the price for the second house, etc.

After the initial price information for all the houses, the input will have a set of transactions (operations) to be processed. This section of the input starts with an integer, t ($1 \leq t \leq 10^5$), indicating the number of transactions. Each of the next t input lines contains a transaction to be processed. There will be two types of transactions:

- Update Transaction: This input line starts with the letter `U` in the first column, followed by one space, followed by a valid house number, followed by a space, followed by an integer (between 1 and 10^9, inclusive), indicating the increase in the price of that house.
- Retrieve Transaction: This input line starts with the letter `R` in the first column, followed by one space, followed by a valid starting house number, followed by a space, followed by a valid ending house number. The total price for this range is being requested. Assume that the ending house number will not be less than the starting house number, i.e., the requested range is valid.

Output

There is no output required for Update transactions. For each Retrieve transaction, output a separate line providing the total price for the range being requested.

Example

Input	Output
5 100 200 150 300 250 8 R 2 4 U 2 20 U 3 40 R 2 4 R 1 5 U 2 10 U 4 100 R 1 5	650 710 1060 1170

Problem G. Which Number

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

Natasha likes counting, but she has been told that a certain set of prime numbers are bad luck. Thus, she skips those numbers and all of their multiples entirely. For example, if 3, 5 and 11 are the primes she is avoiding, then when she starts counting, she'll list the following integers:

1, 2, 4, 7, 8, 13, 14, 16, 17, 19, 23, . . .

You are curious, what is the n^{th} number Natasha will say?

Given the prime numbers whose multiples Natasha avoids, determine the nth number she will say when she starts counting from 1.

Input

The first input line contains two integers: n $(1 \le n \le 10^{17})$, indicating the number for the query, and k $(1 \le k \le 14)$, indicating the number of distinct prime numbers that Natasha avoids when counting (again, the multiples of these primes are avoided as well when counting). The second input line has k distinct prime numbers on it, representing the numbers (and multiples) which Natasha avoids. Assume that the product of all these primes will not exceed 10^{17}, e.g., the list of prime numbers can be $\{2,\ 3,\ 5,\ 11\}$ since their product (330) does not exceed 10^{17} but the list of prime numbers will not be $\{1000000007,\ 1000000009\}$ since their product exceeds 10^{17}. Note that, as illustrated in the Example Input, the primes can be listed in any order (i.e., they are not necessarily listed in increasing order).

Output

Print the n^{th} number Natasha will say.

Example

Input	Output

`11 3`
`3 5 11`
	`23`

`9 4`
`2 7 3 5`
	`37`

Problem H. Maximum Satisfaction

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

UCF is trying to make its COP 2500 class as accessible as possible, by creating s different sections of the course. Of course, it costs money to add sections of a course and UCF requires that each section have at least k students in it.

On the other hand, UCF wants its students to be satisfied. Each student who is going to take COP 2500 has predicted how satisfied they will be if they are placed in each of the different s sections. We assume the students are infallible and that their predictions are perfect. Each prediction is an integer score from 0 to 1000 (0 means not satisfied, 1000 means very satisfied).

Dr. Heinrich is very busy doing some geo-caching, so he would like you to write a program that determines the maximum sum of student satisfiability possible amongst all possible class assignments where each section has at least k students.

Given the number of sections of COP 2500 UCF is offering, the minimum number of students each section must have, and a list for each student of how satisfied they would be in each of the sections of the course, determine the maximum sum of satisfaction of all the students.

Input

The first input line contains three integers: n ($1 \leq n \leq 200$), indicating the number of students, s ($1 \leq s \leq n$), indicating the number of sections of COP 2500, and k ($1 \leq sk \leq n$), indicating the minimum number of students required to be assigned to each section. Each of the following n input lines contain s non-negative integers. The j^{th} value on line i is $a_{i,j}$, the satisfaction rating student i will have in section j.

Output

Print the maximum possible sum of student satisfaction scores within the constraint that each section must have at least k students assigned to it.

Example

Input	Output

`5 2 2`
`10 3`
`6 8`
`9 4`
`11 2`
`12 1`
	`45`

`4 4 1`
`1000 0 0 0`
`0 1000 0 0`
`0 0 1000 0`
`0 0 0 1000`
	`4000`

Explanation

For the first Example input, the maximum possible sum of student satisfaction scores can be achieved by the following assignments:

- Student 1 to Section 1

- Student 2 to Section 2
- Student 3 to Section 2
- Student 4 to Section 1
- Student 5 to Section 1

So, the total satisfaction is: $10 + 8 + 4 + 11 + 12 = 45$

Problem I. Reset

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

Image from pngimg.com

Li and Xiao are involved in a supernatural event in which a crisis will happen at a fixed time in the future. They must complete a number of tasks to prevent the crisis from happening. Once they start a task, they must finish it before switching to another task. Tasks can be completed in any order.

On their first attempt, there may not be enough time for Li and Xiao to complete all the tasks before the crisis hits. The crisis will happen if any task is not completed in time, and Li and Xiao will die. Fortunately, at the moment of their death, the world will be reset: time is rewound to zero and Li and Xiao can have another attempt. All task progress will be lost upon a reset.

Li and Xiao can choose to research a task and gain experience in the task. For each full second Li and Xiao spend researching a task (they cannot spend a partial second on research), their completion time for that task will be reduced by some number of seconds. When the time is reduced to zero, Li and Xiao can complete the task instantly. Upon a reset, their experience from task research remains intact, and in their next attempt they will be able to complete the tasks more quickly. However, there is a constraint that each task can be researched at most once in each attempt.

Witnessing the crisis repeatedly (and therefore dying repeatedly) is not fun. Li and Xiao therefore want to minimize the number of resets they go through before they eventually prevent the crisis.

Input

The first line of input contains two integers n ($1 \leq n \leq 2 \cdot 10^5$) and c ($1 \leq c \leq 10^9$), where n is the number of tasks that Li and Xiao must complete, and c is the number of seconds until the crisis occurs.

Each of the next n lines contains two integers t and d ($1 \leq d \leq t \leq 10^9$) which describe a task, where t is the initial time required to complete the given task. If Li and Xiao research the task for one second, the amount of time to complete it is reduced by d; if this would cause the task completion time to become negative, it is reduced to 0.

Output

Output a single integer, the minimum number of resets required to complete all tasks and prevent the crisis.

Example

Input	Output

3 5
17 5
5 2
15 4
	3

2 1345
1344 1
10 10
	0

Problem J. Rafting Trip

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

You are planning a rafting trip. The terrain can be viewed as a grid. Each cell is either land, or has part of a river flowing through it in one of the four directions: north, south, east, or west. Some land cells contain a sightseeing spot.

You can choose any river cell as the starting point of your rafting trip. Once your raft reaches a river cell (including the starting cell), it will follow the water direction of that cell and move to an adjacent cell or exit the grid.

You can visit a nearby sightseeing spot if your raft reaches a river cell that is adjacent to it, including the starting cell. (Cell adjacency includes horizontal and vertical neighbors, but not diagonal neighbors.) Each sightseeing spot can be visited at most once.

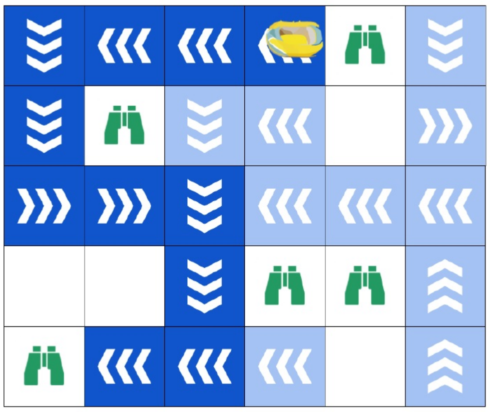


Illustration of the first example case. The optimal rafting trip starts at the cell with the raft and visits 4 sightseeing spots (marked by binoculars). The river cells reached along the trip are highlighted in dark blue.

Your rafting trip stops when your raft moves onto a land cell, exits the grid, or enters a river cell that it has reached before. Note that if the raft ends at a land cell, you cannot visit the sightseeing spots adjacent to that land cell.

What is the maximum number of sightseeing spots you can visit in a single rafting trip if you choose your starting cell optimally?

Input

The first line of input contains two integers r and c ($2 \leq r,\ c \leq 500$); the terrain grid has r rows and c columns.

Each of the next r lines contains c characters describing one row of the terrain grid. A dot '`.`' denotes a land cell without a sightseeing spot. A hash '`#`' denotes a land cell that contains a sightseeing spot. River cells are denoted by '`^`' (north), '`v`' (south), '`>`' (east), or '`<`' (west). There is at least one river cell in the grid.

Output

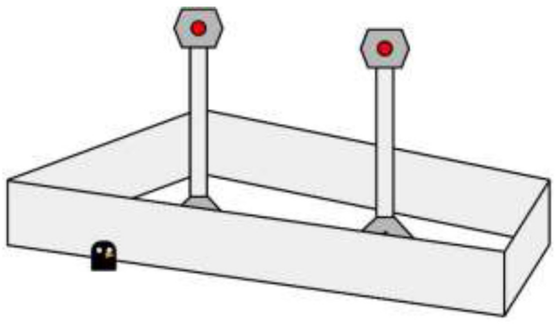
Output a single line with a single integer, which is the maximum number of sightseeing spots you can visit in a single rafting trip.

Example

Input	Output
5 6	4
v<<<#v	
v#v<.>	
>>v<<<	
..v##^	
#<<<.^	
4 5	2
>v<<.	
^<..#	
#...#	
.#>^#	

Problem K. Security Fence

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



You are the last vestige of hope in a world overrun by penguins. You are going to set up a sonic noise emitter array to keep the deranged beasts away from your camp. You have the perfect location to do so. There is already a fence set up to help keep out the beasts while building your weapon. The fence has been constructed by joining a series of rods embedded in the ground with straight sections of reinforced titanium sheets. Luckily the fence's interior angles between the sheets of titanium are each less than 180 degrees. You cannot adjust the location of the fence's rods or the titanium sheets, but the fence's current location should hopefully suffice for the task at hand.

To complete the sonic noise emitter array, you need to build two towers and there is a minimum distance required between these two towers. You don't want the penguins to interfere in your construction. To utilize the fence as much as possible, the towers need to be as far from the titanium sheets as possible while inside the fenced area.

Given the location of the rods of the fence and the minimum distance required between the two towers, determine the maximum possible distance the towers can be from their closest titanium sheets.

Input

The first input line contains two integers, N and D ($3 \leq N \leq 10^5$ and $1 \leq D \leq 10^6$), representing the number of rods that comprise the fence and the minimum distance required between the two towers. Each of the following N lines contains two integers, the i^{th} of which are x_i and y_i ($-10^7 \leq x_i \leq 10^7$ and $-10^7 \leq y_i \leq 10^7$) representing the x and y coordinate of the i^{th} fence rod. The rod locations are given in a clockwise order. It is guaranteed that both towers can exist in the interior of the fenced region.

Output

Print on a single line by itself a single positive number: the furthest possible distance between the closest titanium sheet and towers. Answers within 10^{-6} absolute of the expected answers will be considered correct.

Example

Input	Output

`3 1`
`0 0`
`0 3`
`3 0`
	`0.6715728752538098`

`4 1`
`0 0`
`0 5`
`4 5`
`4 0`
	`1.9999999999999998`

Problem L. Ranked Choice Spoiling

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

Ranked choice voting, a way of determining the winner of an election, proceeds as follows:

1. Each candidate (identified in this problem as A, B, C, etc.) is placed on the ballot.
2. Each voter ranks the candidates from most to least favorite (e.g., B first, then A, then C last) and submits that ranking as their vote.
3. Once all votes have been received, the first place votes for each candidate remaining on the ballot are tallied. If there is a candidate with more than half of the first place votes, that candidate is declared the winner.
4. If no candidate has more than half of the first place votes, the candidate with the fewest first place votes is eliminated from the ballot, and the process returns to step 3 with the remaining candidates. (If there are multiple such candidates, the loser is the lexicographically last candidate, e.g., C is eliminated before B.)

Spoiling refers to an additional candidate Z entering a race, thereby causing the winner to switch from one existing candidate to another candidate (who is not Z). Proponents of ranked choice voting claim that this type of election is less vulnerable to spoiling than the more common plurality voting (the candidate with the most votes wins).

You are candidate A in an election against one or two other candidates, and you wish to prove them wrong by engineering a candidate Z to enter the election and spoil it in your favor. Suppose you knew every voter's rankings for the existing candidates, and you were capable of engineering a candidate Z such that you had full control over where each voter inserts Z into their rankings. For a given set of such rankings, is it possible to engineer a candidate Z to spoil the election in your favor?

Input

The first line contains two integers n ($1 \leq n \leq 1000$) and k ($2 \leq k \leq 3$), where n is the number of voters and k is the number of existing candidates.

Each of the next n lines contains k space-separated capital letters (A, B, or C) indicating the rankings of each voter from first to last place. Each line will list each candidate exactly once (when $k = 2$, C is not present).

Output

Output a single integer, 1 if it is possible to create a candidate Z who spoils the election in favor of candidate A (or if A would already win the election), 0 otherwise.

Example

Input	Output
5 2	1
B A	
A B	
A B	
B A	
B A	
1 3	1
A B C	
8 3	0
A B C	
B C A	
B C A	
B C A	
C B A	
C B A	
C B A	
C B A	