MEMSQL Start[c]UP 3.0

MemSQL Start[c]UP 3.0 - Round 2 (onsite finalists)

A. Save the problem!

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Attention: we lost all the test cases for this problem, so instead of solving the problem, we need you to generate test cases. We're going to give you the answer, and you need to print a test case that produces the given answer. The original problem is in the following paragraph.

People don't use cash as often as they used to. Having a credit card solves some of the hassles of cash, such as having to receive change when you can't form the exact amount of money needed to purchase an item. Typically cashiers will give you as few coins as possible in change, but they don't have to. For example, if your change is 30 cents, a cashier could give you a 5 cent piece and a 25 cent piece, or they could give you three 10 cent pieces, or ten 1 cent pieces, two 5 cent pieces, and one 10 cent piece. Altogether there are 18 different ways to make 30 cents using only 1 cent pieces, 5 cent pieces, 10 cent pieces, and 25 cent pieces. Two ways are considered different if they contain a different number of at least one type of coin. Given the denominations of the coins and an amount of change to be made, how many different ways are there to make change?

As we mentioned before, we lost all the test cases for this problem, so we're actually going to give you the number of ways, and want you to produce a test case for which the number of ways is the given number. There could be many ways to achieve this (we guarantee there's always at least one), so you can print any, as long as it meets the constraints described below.

Input

Input will consist of a single integer A ($1 \le A \le 10^5$), the desired number of ways.

Output

In the first line print integers N and M ($1 \le N \le 10^6$, $1 \le M \le 10$), the amount of change to be made, and the number of denominations, respectively.

Then print M integers $D_1, D_2, ..., D_M$ ($1 \le D_i \le 10^6$), the denominations of the coins. All denominations must be distinct: for any $i \ne j$ we must have $D_i \ne D_j$.

If there are multiple tests, print any of them. You can print denominations in atbitrary order.

Examples

input
18
output
30 4 1 5 10 25

input
3
output
20 2 5 2

input
314
output
183 4 6 5 2 139

B. Ordering Pizza

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

It's another Start[c]up finals, and that means there is pizza to order for the onsite contestants. There are only 2 types of pizza (obviously not, but let's just pretend for the sake of the problem), and all pizzas contain exactly S slices.

It is known that the i-th contestant will eat s_i slices of pizza, and gain a_i happiness for each slice of type 1 pizza they eat, and b_i happiness for each slice of type 2 pizza they eat. We can order any number of type 1 and type 2 pizzas, but we want to buy the minimum possible number of pizzas for all of the contestants to be able to eat their required number of slices. Given that restriction, what is the maximum possible total happiness that can be achieved?

Input

The first line of input will contain integers N and S ($1 \le N \le 10^5$, $1 \le S \le 10^5$), the number of contestants and the number of slices per pizza, respectively. N lines follow.

The i-th such line contains integers s_i, a_i, and b_i ($1 \le s_i \le 10^5$, $1 \le a_i \le 10^5$, $1 \le b_i \le 10^5$), the number of slices the i-th contestant will eat, the happiness they will gain from each type 1 slice they eat, and the happiness they will gain from each type 2 slice they eat, respectively.

Output

Print the maximum total happiness that can be achieved.

Examples

input
3 12 3 5 7 4 6 7 5 9 5

output
84

input
6 10 7 4 7 5 8 8 12 5 8 6 11 6 3 3 7 5 9 6

output
314

Note

In the first example, you only need to buy one pizza. If you buy a type 1 pizza, the total happiness will be $3 \cdot 5 + 4 \cdot 6 + 5 \cdot 9 = 84$, and if you buy a type 2 pizza, the total happiness will be $3 \cdot 7 + 4 \cdot 7 + 5 \cdot 5 = 74$.

C. Gotta Go Fast

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You're trying to set the record on your favorite video game. The game consists of N levels, which must be completed sequentially in order to beat the game. You usually complete each level as fast as possible, but sometimes finish a level slower. Specifically, you will complete the i-th level in either F_i seconds or S_i seconds, where $F_i < S_i$, and there's a P_i percent chance of completing it in F_i seconds. After completing a level, you may decide to either continue the game and play the next level, or reset the game and start again from the first level. Both the decision and the action are instant.

Your goal is to complete all the levels sequentially in at most R total seconds. You want to minimize the expected amount of time playing before achieving that goal. If you continue and reset optimally, how much total time can you expect to spend playing?

Input

The first line of input contains integers N and R $(1 \le N \le 50, \sum F_i \le R \le \sum S_i)$, the number of levels and number of seconds you want to complete the game in, respectively. N lines follow. The ith such line contains integers F_i, S_i, P_i $(1 \le F_i < S_i \le 100, 80 \le P_i \le 99)$, the fast time for level i, the slow time for level i, and the probability (as a percentage) of completing level i with the fast time.

Output

Print the total expected time. Your answer must be correct within an absolute or relative error of 10^{-9}.

Formally, let your answer be a, and the jury's answer be b. Your answer will be considered correct, if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-9}$.

Examples

input
1 8 2 8 81
output
3.14

input
2 30 20 30 80 3 9 85
output
31.4

input
4 319 63 79 89 79 97 91 75 87 88 75 90 83
output
314.159265358

Note

In the first example, you never need to reset. There's an 81% chance of completing the level in 2 seconds and a 19% chance of needing 8 seconds, both of which are within the goal time. The expected time is $0.81 \cdot 2 + 0.19 \cdot 8 = 3.14$.

In the second example, you should reset after the first level if you complete it slowly. On average it will take 0.25 slow attempts before your first fast attempt. Then it doesn't matter whether you complete the second level fast or slow. The expected time is $0.25 \cdot 30 + 20 + 0.85 \cdot 3 + 0.15 \cdot 9 = 31.4$.

D. Buy Low Sell High

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You can perfectly predict the price of a certain stock for the next N days. You would like to profit on this knowledge, but only want to transact one share of stock per day. That is, each day you will either buy one share, sell one share, or do nothing. Initially you own zero shares, and you cannot sell shares when you don't own any. At the end of the N days you would like to again own zero shares, but want to have as much money as possible.

Input

Input begins with an integer N ($2 \le N \le 3 \cdot 10^5$), the number of days.

Following this is a line with exactly N integers $p_1, p_2, ..., p_N$ ($1 \le p_i \le 10^6$). The price of one share of stock on the i-th day is given by p_i.

Output

Print the maximum amount of money you can end up with at the end of N days.

Examples

input
9 10 5 4 7 9 12 6 2 10
output
20

input
20 3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4
output
41

Note

In the first example, buy a share at 5, buy another at 4, sell one at 9 and another at 12. Then buy at 2 and sell at 10. The total profit is $-5-4+9+12-2+10=20$.

E. Hex Dyslexia

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Copying large hexadecimal (base 16) strings by hand can be error prone, but that doesn't stop people from doing it. You've discovered a bug in the code that was likely caused by someone making a mistake when copying such a string. You suspect that whoever copied the string did not change any of the digits in the string, nor the length of the string, but may have permuted the digits arbitrarily. For example, if the original string was $0abc$ they may have changed it to $a0cb$ or $0bca$, but not abc or $0abb$.

Unfortunately you don't have access to the original string nor the copied string, but you do know the length of the strings and their numerical absolute difference. You will be given this difference as a hexadecimal string S, which has been zero-extended to be equal in length to the original and copied strings. Determine the smallest possible numerical value of the original string.

Input

Input will contain a hexadecimal string S consisting only of digits 0 to 9 and lowercase English letters from a to f, with length at most 14. At least one of the characters is non-zero.

Output

If it is not possible, print "NO" (without quotes).

Otherwise, print the lowercase hexadecimal string corresponding to the smallest possible numerical value, including any necessary leading zeros for the length to be correct.

Examples

input
f1e
output
NO

input
0f1e
output
00f1

input
12d2c
output
00314

Note

The numerical value of a hexadecimal string is computed by multiplying each digit by successive powers of 16, starting with the rightmost digit, which is multiplied by 16^0. Hexadecimal digits representing values greater than 9 are represented by letters: $a = 10$, $b = 11$, $c = 12$, $d = 13$, $e = 14$, $f = 15$.

For example, the numerical value of $0f1e$ is $0 \cdot 16^3 + 15 \cdot 16^2 + 1 \cdot 16^1 + 14 \cdot 16^0 = 3870$, the numerical value of $00f1$ is $0 \cdot 16^3 + 0 \cdot 16^2 + 15 \cdot 16^1 + 1 \cdot 16^0 = 241$, and the numerical value of $100f$ is $1 \cdot 16^3 + 0 \cdot 16^2 + 0 \cdot 16^1 + 15 \cdot 16^0 = 4111$. Since $3870 + 241 = 4111$ and $00f1$ is a permutation of $100f$, $00f1$ is a valid answer to the second test case.

F. Egg Roulette

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

The game of Egg Roulette is played between two players. Initially $2R$ raw eggs and $2C$ cooked eggs are placed randomly into a carton. The shells are left on so there is no way to distinguish a raw egg from a cooked egg. One at a time, a player will select an egg, and then smash the egg on his/her forehead. If the egg was cooked, not much happens, but if the egg was raw, it will make quite the mess. This continues until one player has broken R raw eggs, at which point that player is declared the loser and the other player wins.

The order in which players take turns can be described as a string of 'A' and 'B' characters, where the i-th character tells which player should choose the i-th egg. Traditionally, players take turns going one after the other. That is, they follow the ordering "ABABAB...". This isn't very fair though, because the second player will win more often than the first. We'd like you to find a better ordering for the players to take their turns. Let's define the unfairness of an ordering as the absolute difference between the first player's win probability and the second player's win probability. We're interested in orderings that minimize the unfairness. We only consider an ordering valid if it contains the same number of 'A's as 'B's.

You will also be given a string S of length $2(R+C)$ containing only 'A', 'B', and '?' characters. An ordering is said to match S if it only differs from S in positions where S contains a '?'. Of the valid orderings that minimize unfairness, how many match S?

Input

The first line of input will contain integers R and C ($1 \le R, C \le 20, R+C \le 30$).

The second line of input will contain the string S of length $2(R+C)$ consisting only of characters 'A', 'B', '?'.

Output

Print the number of valid orderings that minimize unfairness and match S.

Examples

input
1 1 ??BB
output
0

input
2 4 ?BA??B??A???
output
1

input
4 14 ????A??BB????????????AB????????????
output
314

Note

In the first test case, the minimum unfairness is 0, and the orderings that achieve it are "ABBA" and "BAAB", neither of which match S. Note that an ordering such as "ABBB" would also have an unfairness of 0, but is invalid because it does not contain the same number of 'A's as 'B's.

In the second example, the only matching ordering is "BBAAABABABBA".

G. Flowers and Chocolate

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

It's Piegirl's birthday soon, and Pieguy has decided to buy her a bouquet of flowers and a basket of chocolates.

The flower shop has F different types of flowers available. The i-th type of flower always has exactly p_i petals. Pieguy has decided to buy a bouquet consisting of exactly N flowers. He may buy the same type of flower multiple times. The N flowers are then arranged into a bouquet. The position of the flowers within a bouquet matters. You can think of a bouquet as an ordered list of flower types.

The chocolate shop sells chocolates in boxes. There are B different types of boxes available. The i-th type of box contains c_i pieces of chocolate. Pieguy can buy any number of boxes, and can buy the same type of box multiple times. He will then place these boxes into a basket. The position of the boxes within the basket matters. You can think of the basket as an ordered list of box types.

Pieguy knows that Piegirl likes to pluck a petal from a flower before eating each piece of chocolate. He would like to ensure that she eats the last piece of chocolate from the last box just after plucking the last petal from the last flower. That is, the total number of petals on all the flowers in the bouquet should equal the total number of pieces of chocolate in all the boxes in the basket.

How many different bouquet+basket combinations can Pieguy buy? The answer may be very large, so compute it modulo $1000000007 = 10^9 + 7$.

Input

The first line of input will contain integers F, B, and N ($1 \le F \le 10, 1 \le B \le 100, 1 \le N \le 10^{18}$), the number of types of flowers, the number of types of boxes, and the number of flowers that must go into the bouquet, respectively.

The second line of input will contain F integers $p_1, p_2, ..., p_F$ ($1 \le p_i \le 10^9$), the numbers of petals on each of the flower types.

The third line of input will contain B integers $c_1, c_2, ..., c_B$ ($1 \le c_i \le 250$), the number of pieces of chocolate in each of the box types.

Output

Print the number of bouquet+basket combinations Pieguy can buy, modulo $1000000007 = 10^9 + 7$.

Examples

input
2 3 3 3 5 10 3 7
output
17

input
6 5 10 9 3 3 4 9 9 9 9 1 6 4
output
31415926

Note

In the first example, there is 1 way to make a bouquet with 9 petals $(3 + 3 + 3)$, and 1 way to make a basket with 9 pieces of chocolate $(3 + 3 + 3)$, for 1 possible combination. There are 3 ways to make a bouquet with 13 petals $(3 + 5 + 5, 5 + 3 + 5, 5 + 5 + 3)$, and 5 ways to make a basket with 13 pieces of chocolate $(3 + 10, 10 + 3, 3 + 3 + 7, 3 + 7 + 3, 7 + 3 + 3)$, for 15 more combinations. Finally there is 1 way to make a bouquet with 15 petals $(5 + 5 + 5)$ and 1 way to make a basket with 15 pieces of chocolate $(3 + 3 + 3 + 3 + 3)$, for 1 more combination.

Note that it is possible for multiple types of flowers to have the same number of petals. Such types are still considered different. Similarly different types of boxes may contain the same number of pieces of chocolate, but are still considered different.