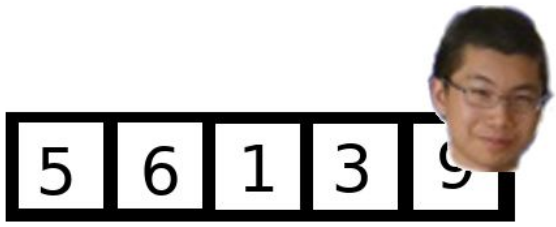
Problem A: Ben Queue

Filename: `benq`
Time limit: `1 second`



A new data structure has recently been created called a Ben Queue. This queue is so fast that it can actually see into the future. Companies from all around the world have already started to utilize this amazing data structure to make more money. Given some kind of resource that the company needs, the Ben Queue will output that resource at the optimal time such that it will be the most affordable for the company. This works for cryptocurrencies, stocks, even lemonade! Of course, the company still has to pay for the resource. For the time that the company will be using the Ben Queue, they would like to always have enough of that resource to be able to scrape by.

Implement the Ben Queue.

Input

The first line of input has a single positive integer, n ($n \leq 10^3$), representing the number of days that the company will be using the Ben Queue for. The second line of input has n space-separated integers, with the i^{th} integer c_i ($0 \leq c_i \leq 10^7$) representing how much the resource costs on day i in dollars. The third line of input has n space-separated integers, with the i^{th} integer d_i ($0 \leq d_i \leq 10^7$) representing how much of the resource was consumed on day i. The company starts with no resources. When buying the resource on any day, they must do so at the beginning of the day.

Output

Output a line containing the minimum amount in dollars that the company will have to pay for their resources.

Samples

Input	Output
3 1 2 3 3 2 1	6
5 4 6 3 7 1 5 1 10 15 8	107

Problem B: Cookie Calories

Filename: `calories`
Time limit: `1 second`

At Insomnia Cookies, one can design their own ice cream sandwich, which consists of two cookies and some ice cream. Due to government regulations, Insomnia Cookies is required to give calorie counts for each of their items. For the ice cream sandwich, the calorie count may vary, depending on which two cookies and which ice cream are chosen to form the sandwich. The government requirement in this case is simply to list the minimum and maximum number of calories possible amongst all possible ice cream sandwiches the user may create.

Due to the popularity of the ice cream sandwich, Insomnia Cookies is thinking about rolling out a new product that is similarly designed, where the user gets to choose n ingredients, and each ingredient can be chosen from several choices. To help them with their design, they would like you to write a program that calculates the minimum and maximum calorie counts of a build your own menu item, where, for each ingredient, the calorie counts of each possibility are given.

Input

The first line of input has a single positive integer, n ($n \leq 10$), representing the number of different items that comprise the build your own item. n lines follow, with information about choices of each ingredient. The i^{th} of these lines starts with a positive integer k_i ($k_i \leq 10$), representing the number of options the user has to select from for item i. This is followed by k_i integers, each representing the number of calories of one of the choices for the i^{th} ingredient. All calorie counts of individual choices for ingredients will be in between 0 and 2,000, inclusive.

Output

On a line by itself, output two space separated integers: the minimum number of calories of any item the user could create, and the maximum number of calories of any item the user could create.

Samples

Input	Output
2 3 110 200 150 2 300 700	410 900
3 4 50 80 100 0 2 250 100 3 140 70 100	170 490

Problem C: Farmer John's Problems

Filename: `problems`
Time limit: `1 second`

Given that Farmer John lives the simple life of a farmer, he sure does get himself caught up into an unusually high number of difficult problems! His problems are so well-known that many, many competitive programming problem writers write about them. It's gotten to the point where many students dread helping Farmer John out.

As it turns out, Farmer John is fictitious (sorry to burst your bubble), so you really shouldn't blame him. Instead, you should blame the problem writers! Unfortunately, more than one person writes about Farmer John, so there's no way to know for certain, who to blame. Instead, the best you can do is calculate a probability that a given problem about Farmer John was written by a particular problem writer.

For the purposes of this question, view problems being added to a database over time. At any given moment, if there was a contest, we could pull one of the problems from the database. We assume that each of the problems is equally likely to be chosen. Write a program to simulate this situation. Namely, your program will read through a sequence of operations, either adding a problem about Farmer John to the database by a particular author, or a query about the probability that an arbitrary problem about Farmer John is by a particular author. For each query, you are to calculate the desired probability.

Input

The first line of input has a single positive integer, n ($n \leq 1000$), representing the number of operations for the input case. This is followed by n lines, one per each operation, in the order they occur. An operation to add a problem to the database will have the following format:

```
1 NAME
```

where 1 indicates that a problem about Farmer John is being added to the database and NAME is the name of the problem author adding the problem. NAME will be in between 1 and 20 characters long and only consist of uppercase letters. An operation to query a particular problem author will have the following format:

```
2 NAME
```

where 2 indicates a query about the author with the given name that follows. NAME will be in between 1 and 20 characters long and only consist of uppercase letters. It's guaranteed that the first operation out of the n operations will be a type 1 operation

Output

For each query of type 2, output, on a line by itself, a fraction p/q, in lowest terms, of the probability that a randomly chosen problem from the Farmer John Problem Database is written by the queried author.

Samples

Input	Output
6 1 BRIAN 1 BRIAN 2 SPENCER 1 SPENCER 1 SPENCER 2 SPENCER	0/1 1/2
11 1 JACOB 2 JACOB 1 ARUP 1 CHARLES 2 ARUP 2 CHARLES 2 JACOB 1 JACOB 2 JACOB 1 ARUP 2 ARUP	1/1 1/3 1/3 1/3 1/2 2/5

Problem D: FJ and Subarray

Filename: `subarray`
Time limit: `1 second`

Farmer John decided to surprise his two favourite cows, Bessie and Elsie, with a special present: an array! Sadly, Bessie and Elsie do not want to share this array, so in a rush FJ has decided that he will take two non-overlapping subarrays, both of length ***k***, and give one to Bessie and one to Elsie, and then save the rest of the array for later.

Even with all the fuss, FJ still wants Bessie and Elsie to be happy, so he wants to pick two ***contiguous*** subarrays such that the sums of the values in the two subarrays are as close as possible. If there are multiple subarrays whose sum differences are minimal, FJ picks the two subarrays that add up to the largest total. Bessie is always given the left chosen subarray; Elsie is given the right.

Input

On the first line, there are two integers ***n*** and ***k*** ($1 \le n \le 1000$ and $2k \le n$). On the next line are ***n*** space-separated integers between 1 and 1000, the contents of the array.

Output

Print an integer representing the sum of elements in Bessie's array and Elsie's array.

Samples

Input	Output
6 2 3 6 7 9 6 6	28
6 3 1 2 3 4 5 6	21
9 1 1 1 6 5 8 7 11 10 9	2

Sample Explanations

In the first sample Bessie gets the subarray ***[2, 3]*** and Elsie gets ***[4, 5]***. (Note that ***[1,3]*** and ***[4,5]*** can't be chosen because they are different lengths and ***[2,3]*** and ***[5,6]*** can't be chosen because they aren't contiguous.)In the second sample Bessie gets ***[1, 3]*** and Elsie gets ***[4, 6]***. This is the only valid arrangement in this case. In the third, Bessie gets ***[1, 1]*** and Elsie gets ***[2, 2]***.

Problem E: Fire Sale 2

Filename: `firesale2`
Time limit: `2 seconds`

There is a fire sale going on at Anya's favorite toy store again! As Arup wants to make Anya happy, he gives her a maximum allowance of k dollars. It is Anya's goal to buy as many toys as possible without spending more than k dollars.

However, the fire sale has certain rules. All of the toys are lined up in a row, and Anya can only buy toys that are in one contiguous sequence. The store owner is mad because how much money he lost last time Anya came into his store. Now, the cost of the sequence of toys is the average of all the toys in the subarray.

Input

The first line of input will contain two space separated integers, n ($1 \le n \le 10^6$), k ($1 \le k \le 10^6$) representing the number of toys in the store and the maximum amount of money Arup is willing to spend. On the following line is n space separated integers, a_i ($1 \le a_i \le 10^6$), the costs of each toy.

Output

On a single line, output the most toys Anya can buy without going over her allowance.

Samples

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

Problem F: Dank Memes

Filename: `memes`
Time limit: `3 seconds`

As you well know, one big problem in the land of dank memes is stolen memes. However, to impress your normie friends you've decided that you're going to start stealing dank memes to send to them. Because you're quite the meme lord, you have a vast selection of memes to pick from. In order to keep up your facade of being original, you want to minimize the chance of getting caught with stolen memes.

The i^{th} meme in your meme library has two associated values: r_i, how recognizable the meme is, and d_i, how dank the meme is. To gain the most meme clout with your pals, you want to choose dank memes such that their total dankness is maximal. You also want to pick a set of stolen memes with total recognizability no larger than R. The total recognizability of a set of memes is simply the sum of the recognizability scores of each of those memes.

Given R, a set of n stolen memes, each with their respective r_i and d_i values, determine the largest total dankness of memes you can share with your normie friends. Remember, you can't use the same meme twice, because then you're just another normie.

Input

The first line of input contains two integers n and R ($1 \le n, R \le 4{,}000$). Line i of the next n lines describes the i^{th} stolen meme with two integers r_i and d_i ($1 \le r_i \le 3{,}000$; $0 \le d_i \le 10^9$), denoting the recognizability and dankness respectively.

Output

Print out a single integer D, the largest total dankness you can achieve by sharing some set of stolen memes with your friends without surpassing R in recognizability.

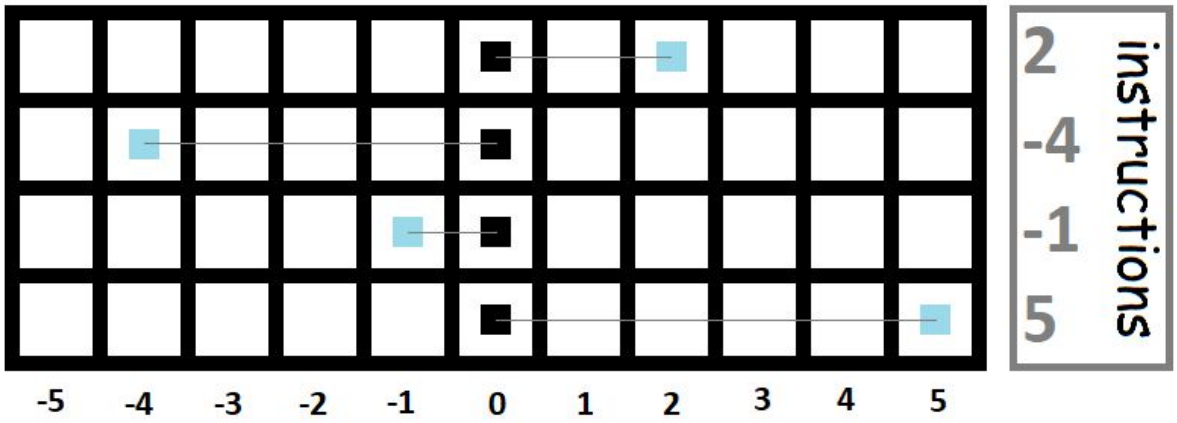
Samples

Input	Output
5 20 1 2 2 1 3 4 4 3 5 5	15
3 20 20 1000 19 999 1 1	1000

Problem G: Ballerina Rage

Filename: `rage`
Time limit: `3 seconds`

The International Ballerina Monarchy has decreed that all of its citizens must stand on a grid and occupy a variety of locations! The Monarchy can instruct its citizens to move left and right using a variety of ***C*** instructions cards the cardinals prepared earlier. The i^{th} number on each card serves as an instruction for the i^{th} citizen in line on the grid to move a certain distance. Look below for an example instruction card and the effect it has if each citizen begins at location



0.

There are ***N*** citizens, and ***N*** infinitely long horizontal rows for them to stand on. Each of the citizens start at location 0 in their respective row. The Monarchy has ***L*** positions they would like the citizens to move into, and they would like to know whether or not they can use the instruction cards to get them there.

Note that all of the cards can be used entirely forward or entirely reverse. For example, if N = 3, and the citizens are currently at (1, 1, 1), an instruction set of (1, -2, 4) could be used to bring the citizens to either (2, -1, 5) or (0, 3, -3). Furthermore, any fraction of the card's values can be used as well. An instruction set of (3, 5, 7) could be scaled down and used like (0.6, 1, 1.4). The cards can also be reused as many times as desired.

Input

Each case will begin with a single line containing three integers ***N*** ($1 \le N \le 50$), ***C*** ($1 \le C \le 50$), and ***L*** ($1 \le L \le 100$), corresponding to the number of citizens, the number of instruction cards, and the number of locations the Monarchy has chosen. ***C*** lines will follow, each will contain ***N*** integers a_i ($-10^4 \le a_i \le 10^4$), indicating that the i^{th} number on the instruction card is a_i. Then ***L*** lines will follow, each containing ***N*** integers p_i ($-10^9 \le p_i \le 10^9$), indicating that the Monarchy would like citizen ***i*** standing at location p_i for all i simultaneously.

Output

For each of the ***L*** sets of positions, output “YES” on a line by itself if the Monarchy can use the cards to get the citizens to that set of locations, or “NO”, otherwise.

Samples

Input	Output
3 3 2 1 1 0 4 -3 -1 0 -9 7 8 7 -3 -2 1 5	YES YES
4 3 4 1 2 3 1 1 2 3 2 1 2 3 3 3 4 1 5 6 12 18 6 1 2 3 1 5 4 9 11	NO YES YES NO

Problem H: Get Spencer Off Campus

Filename: `spencer`
Time limit: `1 second`

Spencer is brilliant when it comes to algorithms but terrible when it comes to directions. He's been having trouble navigating his way around campus. So far, Charles has been helping him find the lecture room, or Arup has been letting him just stay in HEC-101.

But this time, neither Arup nor Charles are around. You must help Spencer get off campus. If he can't leave UCF, Spencer won't be able to earn a Gold Medal at IOI or attend MIT for that matter! This project is a matter of national security. Don't let Spencer down!

We model campus as a ***R*** x ***C*** grid of squares. Spencer will initially occupy a single one of those squares. At any moment in time, Spencer can be facing one of four directions, up, down, left or right. Some of the campus grid squares are obstructed and Spencer can not travel to those squares. Otherwise, Spencer can move in a straight line in his current direction as many squares as possible until there is an obstructed square, without any trouble. Alternatively, Spencer can turn either left or right, which results in him staying in the same grid square, but changing his direction. (If he wanted to do a u-turn, he would have to either turn left twice or turn right twice.)

The goal of this problem is to get Spencer off campus. Since Spencer has extreme difficulty following directions with many turns, the goal will be to get Spencer to any unobstructed border square (in either the first or last row, or first or last column) ***using as few turns as possible.***

Write a program to calculate the fewest number of turns necessary to get Spencer off campus, so that he can bring glory back to the United States and attend his intended college. If this is not possible, your program must report that the task is impossible.

Input

The first line of input has two integers ***R*** **(**3 ≤ ***R*** ≤ 50) and ***C*** (3 ≤ ***C*** ≤ 50), representing the number of rows and columns, respectively, on the grid. The following ***R*** rows have a string of precisely ***C*** characters. Each of these characters will either be '_' , 'X', 'N', 'S', 'W', or 'E'. The character '_' indicates a passable square. The character 'X' indicates an obstructed square where Spencer can not travel. Of the four characters 'N', 'S', 'W' and 'E', exactly one of them will appear in the grid and that character will appear exactly one time. The location of this character represents Spencer's initial starting square. The letter itself represents which direction Spencer is initially facing, 'N' for North, 'S' for South, 'W' for west, and 'E' for east. It is guaranteed that at least one square on the border of the board will not be an obstructed square.

Output

On a line by itself, output a single integer, indicating the fewest number of ***turns*** necessary for Spencer to get off campus (to a border square). If it's not possible for Spencer to do so, output -1.

Samples

Input	Output
5 5 XXX__ XNX_X X_X_X X___X XXXXX	4
5 7 XXXXXXX __XXX__ __XWX__ __XXX__ _______	-1

Sample Explanation

In the first example, Spencer must turn either left or right twice, so that he's facing South. Then he will travel south two squares before turning left. Then he will travel two more squares east before turning left again. From here, Spencer can go straight and get off campus to row 0, column 3.

Problem I: Byteland Bands Together

Filename: `together`
Time limit: `2 seconds`

Byteland is a very long rectangular land comprising of ***n*** different states arranged as contiguous squares in a line. In order, these states are labeled 1 to ***n***, from left to right. Historically, each state had its own constitution, but the country has banded together to form a single unified constitution!

In order to agree on a single constitution, the process to be used is as follows:

If the contiguous states from state ***i*** to ***j*** (***i*** ≤ ***j***) share a constitution and the contiguous states from ***j***+1 to ***k*** (***j*** < ***k***) share a different constitution, then the delegates from both groups, [***i***, ***j***] and [***j***+1, ***k***] can meet for a convention in either state ***j*** or state ***j***+1 to hammer out the differences and agree on a single unified constitution for the contiguous states in the range [***i***, ***k***]. Unfortunately, hosting such a convention costs money. In particular, in order to entice each of the delegates to attend the convention, the planners must buy each of them a famous Byteland Taco! Due to differences in economies of the different states, the cost of the famous Byteland Taco varies from state to state. In particular, in state ***i***, the cost of a taco is c_i. Also, the different delegations from each state vary in size. The delegation from state ***i*** has m_i members.Thus, the total cost for holding the convention at state ***j*** is $c_j \sum_{p=i}^{k} m_p$.

Over the course of ***n-1*** conventions, Byteland will have its unified Constitution. Unfortunately, depending on the order of these conventions, the total cost of unifications may vary significantly. Although tacos are delicious, Byteland would like to minimize the cost of the tacos over all of the conventions.

Given the cost of tacos in each state and the number of delegates from each state in Byteland, determine the minimum cost of tacos required for the country to agree on a single Constitution.

Input

The first line of input contains a single positive integer ***n*** (1 ≤ ***n*** ≤ 500), representing the number of states in Byteland. The next line of input contains ***n*** integers, the i^{th} of which is m_i ($1 \le m_i \le 10^5$), the number of members in the delegation from state ***i***. The last line of input has ***n*** integers, the i^{th} of which is c_i ($1 \le c_i \le 10^5$), the cost of a taco in state ***i***.

Output

Output the minimum cost of tacos necessary for the conventions for Byteland to agree on a single constitution.

Samples

Input	Output
4 1 9 6 3 3 6 9 1	145
2 1 6 9 3	21