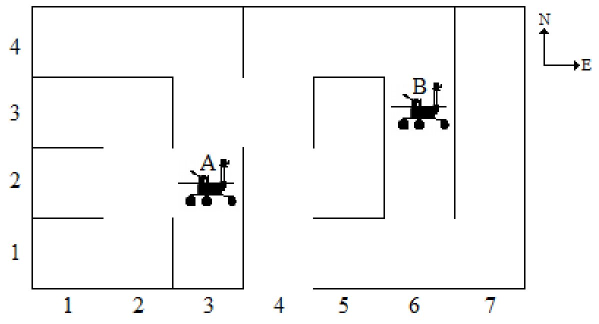
Problem A. A-Mazing Puzzle

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

The Severely Challenging Archaeological Exploration Division (SCArED) specializes in the investigation of archaeological sites that are too difficult, dangerous or downright terrifying for humans to examine firsthand. Instead they send in one or more mobile robots to investigate, each robot outfitted with a complex set of sensors, recorders and manipulators. Each robot is radio-controlled and has a small nuclear reactor to power what sometimes ends up being an extended stay at the sites.

Recently a complex underground maze of the Desreverezam Civilization has been discovered. SCArED sent in two robots to investigate and over the course of several months they have mapped out the entire maze. But then disaster struck — due to some unforeseen anomaly they no longer have radio contact with the two robots (the researchers are torn between the cause of this: either some magnetoferritin crystals in the walls or angry poltergeists). After several weeks of trying to regain contact with the two robots they finally have managed to open up a shared channel with them. There's not much bandwidth in the connection, so they can only send three basic commands to the robots: (Move) Forward, (Turn) Left and (Turn) Right. Since this is a shared channel they cannot send differentiated commands to each robot, so if they send the Forward command each robot will try to move forward simultaneously. If there is an opening in front of them that's fine but if there's a wall in front of one of the robots it will bump into the wall and remain where it is. Any Turn command will likewise cause the robots to turn in lockstep.



Example Input 1.

The researchers would like to find the minimum number of Forward commands to get both robots out of the maze (the Turn commands take so little time and power that they can be ignored). As soon as either is out of the maze they can return to the surface and ignore any future commands. Consider the situation in Figure 1 where both robots are currently facing South. One approach to get the robots out of the maze is to first get robot B out as quickly as possible by moving it to locations $(6, 2)$, $(6, 1)$, $(5, 1)$, $(4, 1)$ and then out of the maze. This takes 5 Forward moves and in the process robot A moves to $(3, 1)$ and stays there, bumping against walls 4 times. After robot B is out robot A can exit the maze in 6 more Forward moves, for a total of 11 moves. However, if robot A is moved out as quickly as possible first and then robot B is moved out, only 8 Forward moves are needed. For other mazes, the minimum number of Forward moves may not involve getting either robot out as quickly as possible (see Sample Input 2).

Given a maze and the initial positions and orientations of the robots, help the researchers out by determining the minimum number of Forward commands to get both robots out of the maze. If there are several sets of minimum Forward commands then the one that leads to the least number of wall bumps is preferred. Oh, and one more thing: the robots should never be on top of each other inside the maze — this may lead to a chain reaction of their nuclear power generators leading to an explosion that could destroy the planet. This should be avoided.

Input

Input starts with a line containing 3 positive integers c r e $(c, r \leq 50, e \leq c)$ indicating the maze has c columns and r rows (all numbered starting at 1) and that the exit to the maze is on the southern side of square $(e, 1)$. Following this is a line of the form c_1 r_1 d_1 c_2 r_2 d_2 $(1 \leq c_i \leq c, 1 \leq r_i \leq r, d_i \in \{\texttt{N,E,S,W}\})$ indicating that the two robots are at locations (c_1, r_1) and (c_2, r_2) with orientations d_1 and d_2, respectively. Locations (c_1, r_1) and (c_2, r_2) will never be the same, and d_1 and d_2 may be different.

After this come two lines which describe the maze. The first line has the form n c_1 r_1 c_2 $r_2 \ldots c_n$ r_n $(n \leq r * c, 1 \leq c_i \leq c, 1 \leq r_i < r)$ where each c_i r_i pair indicates that there is a horizontal wall between grid squares (c_i, r_i) and $(c_i, r_i + 1)$. The next line is similar, where each c_i r_i pair $(n \leq r * c, 1 \leq c_i < c, 1 \leq r_i \leq r)$ indicates that there is a vertical wall between grid squares (c_i, r_i) and $(c_i + 1, r_i)$. All exterior walls except the exit are assumed to exist and are not listed in these two lines.

Output

Output two values: the minimum number of Forward commands to get the two robots out of the maze and the number of bumps that occur using these commands. If there are several ways to get the robots out with the same minimum number of Forward command, use the one that minimizes the number of bumps. It will be possible for both robots to escape the maze in each input instance.

Example

Input
7 4 4 3 2 S 6 3 S 6 1 1 1 2 1 3 2 3 5 1 5 3 11 2 1 3 1 3 2 5 2 6 2 2 3 4 3 5 3 6 3 3 4 6 4
Output
8 1

Input
3 4 2 1 3 S 3 2 S 1 3 3 5 1 1 2 1 1 2 1 3 2 3
Output
7 2

Problem B. A Musical Question

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

Bob Roberts likes to listen to music while he drives, but the car he owns is a little antiquated. No Bluetooth or USB connections here, but at least he has a CD player, so he's been transferring a lot of his music to CDs. At the moment he has only two CDs left and would like to get as much of his remaining music as possible on them. Given the capacity of the CDs and collection of songs, can you help him find the maximum number of minutes of music he can put on the two CDs?

Input

Input starts with a line containing two integers c n, where c $(1 \leq c \leq 1\,000)$ is the number of minutes of music each CD can hold, and n $(1 \leq n \leq 1\,000)$ is the number of songs to select from. Following this is a single line containing n positive integers indicating the length (in minutes) of each of the songs. No song will be longer than $1\,000$ minutes.

Output

Output the amount of music on each CD, in minutes, that maximizes the total amount of music that Bob can transfer to the two CDs. Display the time of the larger-filled CD first. If there is a tie, use the solution which minimizes the time difference between the two CDs.

Example

Input	Output

`100 5`
`10 20 40 60 85`
	`100 95`

`100 5`
`10 20 30 40 50`
	`80 70`

Problem C. Cribbage On Steroids

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

Cribbage is a two-person card game where players score points for various combinations of cards. A standard 52-card deck is used where cards have one of 4 suits (not important in this problem) and one of 13 ranks. The card ranks, from lowest to highest, are Ace (A), 2, 3, 4, 5, 6, 7, 8, 9, 10 (T), Jack (J), Queen (Q), King (K). In normal cribbage, players make combinations from a hand consisting of five cards (four in their hand and one common card). The possible combinations and their point values are the following:

15's : any combination of card ranks that total exactly 15 scores 2 points. The values of Kings, Queens and Jacks are 10, the value of Aces is 1 and all other card values are the same as their rank.

pairs : any two cards with the same rank scores 2 points. Note that three cards with the same rank will score 6 points since it contains three separate pairs; four cards with the same rank scores 12 points; five cards with the same rank... but we're getting ahead of ourselves.

runs : for each disjoint run of length three or more, each set of cards that generates the longest run scores 1 point per card. So for example, the cards $2, 2, 3, 4, 5, 8, 9$, T, J, Q will score 8 points for two four-card sequences (there are 2 ways to make $2, 3, 4, 5$) and 5 points for a five-card sequence (there is one way to make $8, 9$, T, J, Q).

The total score across each of these three categories is the hand's score.

For example, a five-card hand consisting of $4, 5, 5, 5, 6$ will score 23 points: 8 points for 15's (three 4-5-6 combinations and one 5-5-5), 6 points for pairs (three pairs of 5's) and 9 points for runs (three 4-5-6 runs). The five-card hand T,T,J,Q,Q scores 16: 4 points for pairs (T's and Q's) and 12 points for runs (four different runs of T-J-Q). There are other ways to score as well but the three rules above will suffice for this problem.

Now as we said, in normal cribbage you look for combinations in five-card hands. But we're far from normal. Your task is to determine the value of an n-card hand when n can be significantly larger than 5.

Input

Input starts with a integer n ($5 \leq n \leq 100$) indicating the number of cards in the hand. Following this are n characters taken from the set `{A,2,3,4,5,6,7,8,9,T,J,Q,K}` indicating the ranks of the n cards. These characters will be on one or more lines with one space between any two characters on the same line. The card ranks will not necessarily be in sorted order. Note that unlike a standard deck of cards, there may be more than four cards of any rank.

Output

Output the total score achieved by the n-card hand.

Example

Input	Output
5 4 5 6 5 5	23
13 A 2 3 4 5 6 7 8 9 T J Q K	71
10 2 2 3 4 5 8 9 T J Q	45

Problem D. Determining Nucleotide Assortments

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

Genes 'R Us specializes in analyzing strands of DNA to look for anomalies, matches, patterns, or whatever specific items their customers are interested in. You may recall from high school biology that the DNA molecule consists of two chains wrapped around each other to form the well-known double helix. The chains are made up of four different types of nucleotides, each distinguished by a specific nitrogen base: adenine (`A`), thymine (`T`), guanine (`G`) and cytosine (`C`).

A recent customer of Genes 'R Us wants to be able to quickly determine which of the four types of nucleotides is most prevalent in a given section of DNA, as well as which are second, third and fourth most prevalent. For a given strand of DNA (which may contain tens of thousands of nucleotides) there may be many such sections where this information is desired. You've been asked to write an efficient program to answer such questions.

Input

Input starts with a string describing a DNA strand. This string consists of the characters `A`, `T`, `G` and `C` and have positive length $n \leq 50\,000$. The next line contains a positive integer m ($m \leq 25\,000$) indicating the number of sections of the DNA strand to investigate. Following this are m lines each containing a pair of integers s_i e_i, each indicating the start and ending positions of a section of the strand, where $1 \leq s_i \leq e_i \leq n$. Integer pairs are separated by a single space.

Output

For each section output a line containing a permutation of `ATGC`, where the first character indicates the most prevalent nucleotide in the section, the second character indicates the second most prevalent, and so on. Break any ties using the ordering `ATGC` (i.e., print an `A` before an equally occurring `T`, `G` or `C`, print a `T` before an equally occurring `G` or `C`, and so on).

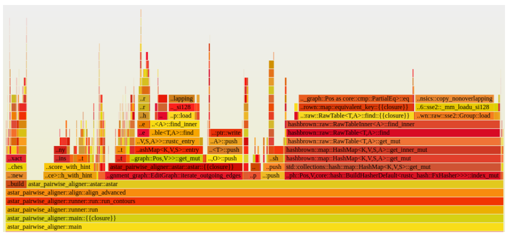
Example

Input	Output

`TATATGCTCT`
`3`
`1 10`
`6 10`
`6 6`
	`TACG`
	`TCGA`
	`GATC`

Problem E. Fastestest Function

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


A flamegraph.

You are working as a software developer for the Bug Acquisition Programming Company. They developed a specific piece of software called Program C that they sell to their clients. For the past weeks, you have been working on optimising a specific function `foo` in the main code path in Program C. You have made it a lot faster and would like to show off to your boss about it.

Your IDE has a nice tool that allows you to profile your code and tell you what percentage of the total running time `foo` takes. You can run this on the version before your change and after your change. However, you think it looks a lot cooler if you can just tell your boss how much faster you have made `foo` itself.

Input

The input consists of:

- One line with two integers x and y ($0 < x, y < 100$), where x is the percentage of the total running time that `foo` took before optimising and y the percentage of the total running time it took after optimising.

Output

Output the factor of how much faster `foo` got after your optimization.

Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
75 50	3.0
50 75	0.3333333333333333
50 50	1.0

Problem F. Pea Pattern

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

Do you see the pattern in the following sequence of numbers?

$$1, 11, 21, 1112, 3112, 211213, 312213, \ldots$$

Each term describes the makeup of the previous term in the list. For example, the term 3112 indicates that the previous term consisted of three 1's (that's the 31 in 3112) and one 2 (that's the 12 in 3112). The next term after 3112 indicates that it contains two 1's, one 2 and one 3. This is an example of a *pea pattern.*

A pea pattern can start with any number. For example, if we start with the number 20902 the sequence would proceed 202219, 10113219, 1041121319, and so on. Note that digits with no occurrences in the previous number are skipped in the next element of the sequence.

We know what you're thinking. You're wondering if 101011213141516171829 appears in the sequence starting with 20902. Well, this is your lucky day because you're about to find out.

Input

Input consists of a single line containing two positive integers n and m, where n is the starting value for the sequence and m is a target value. Both values will lie between 0 and $10^{100} - 1$.

Output

If m appears in the pea pattern that starts with n, display its position in the list, where the initial value is in position 1. If m does not appear in the sequence, display `Does not appear`. We believe that all of these patterns converge on a repeating sequence within 100 numbers, but if you find a sequence with more than 100 numbers in it, display `I'm bored`.

Example

Input	Output
`1 3112`	`5`
`1 3113`	`Does not appear`
`20902 101011213141516171829`	`10`

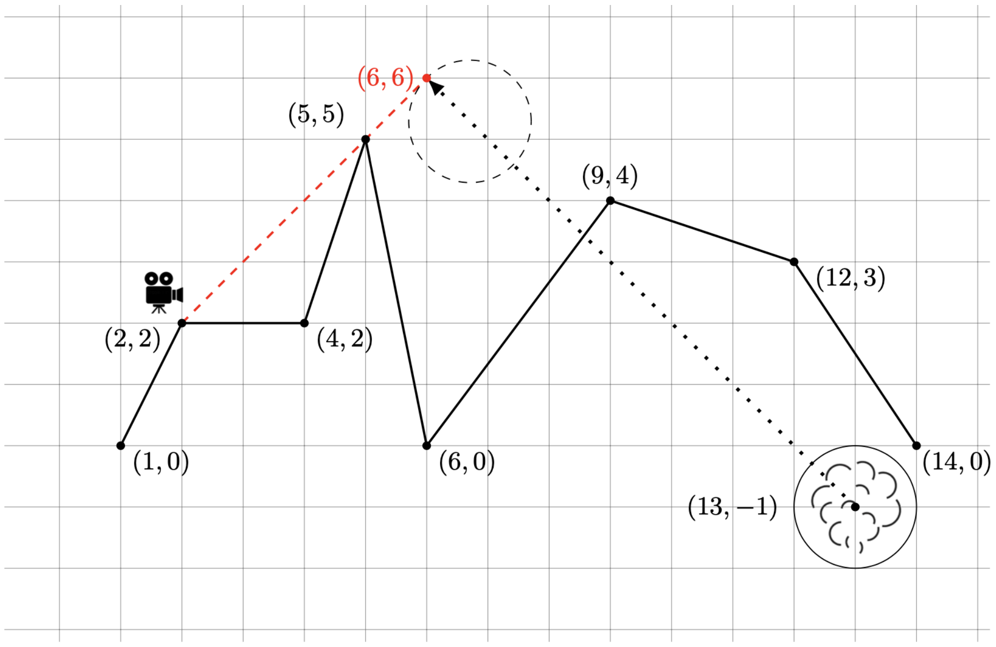
Problem G. Picking Up Steam

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

Meteorologist Wendy Wynne Blose and geologist Maddy Morfik are in the Porous Mountains of Central Asia, studying mild eruptions of superheated smoke and steam from below ground. Unlike more dramatic eruptions such as the one that blew the top off Mount St. Helens in 1980, these milder events involve the expelling of steam through many tiny fissures in the (extremely porous) ground, preserving the ground surface while producing a visible cloud that rises through the air. Wendy and Maddy would like to capture the path of such a cloud on video.

Like all good scientists, they have made some simplifying assumptions. They assume the cloud will be spherical, of known radius, and will travel at a fixed known speed along a straight line. They also assume that the cloud will form underground and will be expelled in a certain direction, always able to pass through the porous mountain rock with no change in speed, direction, or shape. The scientists have delicate instruments that are quite good at predicting when such an underground steam cloud will be expelled. Finally, they have a model of the surrounding mountain range in the form of a continuous piecewise-linear approximation.

For safety reasons they will set up a video camera and a timer that will turn on the camera at the instant they predict the steam cloud will first become visible from the camera's vantage point; then they will go to a safe location far away and wait until the recording is complete. The figure below shows a typical scenario. The camera at $(2, 2)$ needs to be aimed in the direction of the peak at $(5, 5)$ in order to pick up the cloud at the moment it first becomes visible at $(6, 6)$. If the cloud has a radius of 1 unit and is travelling in the direction of the vector $[-1, 1]$ (i.e., at an angle of 135 degrees) from its underground starting point at $(13, -1)$ at a speed of one unit per second, the camera should be turned on after 8.899 seconds.



First example input

In order to get maximum recording time from the camera's battery, they need to know the precise time to turn on the camera. That's where you come in.

Input

The first line contains an integer n ($2 \le n \le 1\,000$, the number of line segments defining the mountain range, followed by $n+1$ integer pairs (x_i, y_i) ($-5\,000 \le x_0 < \cdots < x_n \le 5\,000$, $0 \le y_i \le 5\,000$), where (x_{i-1}, y_{i-1}) and (x_i, y_i) define the endpoints of the i-th line segment. No three consecutive points will be collinear. The second line contains seven integers c, sx, sy, r, dx, dy, and v, where c ($x_0 \le c \le x_n$) specifies the x-coordinate of the camera's location along the mountain range, (sx, sy) ($x_0 \le sx \le x_n$, $-5\,000 \le sy \le 5\,000$) specifies the initial underground location of the steam cloud, r ($1 \le r \le 5\,000$) is the radius of the cloud, dx and dy ($-5\,000 \le dx \le 5\,000$, $1 \le dy \le 5\,000$) specify the direction of travel of the steam in vector form (dy units of vertical distance for each dx units of horizontal distance), and v ($1 \le v \le 1\,000$) is the velocity of the cloud in units per second.

Output

Output a single real number t, the time, in seconds, when the camera should be turned on, i.e., when the first point of a non-zero volume of visible cloud can be seen at a location between x_0 and x_n. Assume the cloud begins moving from its starting point at time 0 and maintains a constant speed and direction from that point on. The cloud will first emerge from underground at a point whose x-coordinate is between x_0 and x_n. If it will never be possible for the camera to view the cloud at a point whose x-coordinate lies between x_0 and x_n, output the value -1. Answers should be accurate with an absolute error of 10^{-3}.

Example

Input	Output

`7 1 0 2 2 4 2 5 5 6 0 9 4 12 3 14 0`
`2 13 -1 1 -1 1 1`
	`8.899`

Problem H. Road To Savings

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

Pat Wholes is in charge of road maintenance in Capitol City, and boy do those roads need maintenance. The road conditions are so poor that accidents have become almost a daily event, and the surviving public is in an uproar. Now while Pat would love to pave every road in the city, he also wants to keep his job. The cost of all that paving would certainly upset the mayor, who would just as certainly replace Pat if he spends too much. So now the decision is: which roads get paved and which don't? After thinking about this problem – and his job security – Pat came up with a bright idea: since the ultimate goal is to keep the mayor happy, he'll pave only those roads that are on a shortest path from the mayor's house to the mayor's office. There actually might be several ways for the mayor to drive to work that are equally short, but that should still leave plenty of roads that aren't on any of these paths and hence plenty of roads that don't need to be paved (hopefully). Pat's come down to your cubicle in the basement to ask you to determine the length of roads that don't need to be paved.

Input

Input starts with four positive integers n m a b $(n, a, b \leq 100, a \neq b)$ where n indicates the number of intersections in the town (numbered 1 to n), m is the number of roads connecting intersections, and a and b are the intersections where the mayor's house and office are located, respectively. Following this are m lines, each containing a triplet of numbers i_1 i_2 ℓ $(1 \leq i_1, i_2 \leq n,\ i_1 \neq i_2,\ 1 \leq \ell \leq 100)$ indicating a two-way road exists between intersection i_1 and i_2 with length ℓ. At most one road exists between any two intersections and at least one path exists between a and b.

Output

Output the total length of all roads that don't need to be paved.

Example

Input	Output

`4 5 1 4`
`1 2 1`
`1 3 2`
`1 4 2`
`4 2 1`
`3 4 1`
	`3`

`4 5 1 4`
`1 2 1`
`1 3 2`
`1 4 1`
`4 2 1`
`3 4 1`
	`5`

Problem I. Simple Solitaire

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

There are many different varieties of solitaire (card games for one), but most require an area where you can lay out cards. Here's one you can do by simply holding the deck in your hand.

The game uses a standard 52 card deck where each card has one of 4 suits – spades (S), hearts (H), diamonds (D) or clubs (C) – and one of 13 ranks – Ace (A), 2, 3, 4, 5, 6, 7, 8, 9, 10 (T), Jack (J), Queen (Q), King (K). To play the game you hold the deck face down in your hand and starting turning over cards from the bottom of the deck to the top, face up. Each time you turn over a card you check if it matches the rank or suit of the card three positions before it in the previously turned-over cards. If they match in rank, you remove those two cards and the two cards between them; if they match in suit you remove just those two cards. These actions might cause a cascade effect as now one or more of the remaining turned-over cards might match the ranks or suits of the cards three positions before them. (More detailed rules for processing cascades are described below.) Once all actions in a cascade have been performed (some of which may have forced the processing of even more cascades), you turn over the next card. If all the cards are removed in a single pass through the deck, you win the game!

The start of a sample game is shown below: suppose the first seven cards turned over are the TH, 4C, KS, AD, 2S and 8H. The table below show the next few actions taken in the game – suits and ranks are underlined when they match. Note that turning over the 8S results in a cascade of two actions.

Card Dealt	Hand	Action
—	TH 4C KS AD 2S 8H	none
5D	TH 4C KS A$\underline{\text{D}}$ 2S 8H 5$\underline{\text{D}}$	remove AD, 5D
6D	TH 4C KS 2S 8H 6D	none
8S	TH 4C KS 2$\underline{\text{S}}$ 8H 6D 8$\underline{\text{S}}$	remove 2S, 8S
—	T$\underline{\text{H}}$ 4C KS 8$\underline{\text{H}}$ 6D	remove TH, 8H
9H	4C KS 6D 9H	none
KC	4C $\underline{\text{K}}$S 6D 9H $\underline{\text{K}}$C	remove KS, 6D, 9H, KC
2C	4C 2C	none

If there is ever a choice of actions (which can happen during a cascade), you should pick an action which removes four adjacent cards over an action that removes two non-adjacent cards. If several possible actions might result in removing the same number of cards, use the one that removes the most recently turned-over card. For example, if the 6D had been 6C in the example above, then in the fifth row of the table the 4C and 6C would have been removed instead of the TH and 8H.

You have one solitary task: given the order in which cards are turned over, determine the cards remaining at the end of the game.

Input

Input consists of four lines, each containing 13 cards. Cards appear in the order in which they are turned over in the game. Each card is represented by two consecutive characters, the first indicating the rank of the card (from `A`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `T`, `J`, `Q`, `K`) and the second indicating the suit of the card (from `S`, `H`, `D`, `C`). A single space will separate adjacent cards.

Output

Output the number of cards remaining at the end of the game followed by the remaining cards, using the format described in the Input section.

Example

Input	Output
TC 2C 6C TS KC QS QC 3C KD 8D JH JS KH 5D JD 2S 8S AS 9S 3D 5H 9C AH 4D 4C KS JC 4S 7S 6D 2H 7C 8C 7D AD 7H TH 2D QH 8H 9H 5C TD 3S 6H 3H QD 5S 9D 4H 6S AC	2 3S 9D

Problem J. Two Charts Become One

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

When businesses feel that they are a little bloated and in need of restructuring, they call in the well-known reorganization consultant Don Sizing. One of the first things Don asks for is a chart showing the hierarchy of departments in the business, i.e., which departments are in charge of other departments. Often there will be some confusion about this hierarchy and Don will end up with two or more different charts displaying the same set of departments. In situations like this Don must determine if the charts are similar, i.e., if they show the exact set of hierarchies while not necessarily being drawn the same way. For example, consider the two charts shown on the left in Figure 3. While they do not look the same, they do represent the same hierarchies – department 11 is in charge of departments 10 and 12 and department 12 is in charge of departments 13, 17 and 28. The three charts on the right in Figure 3 are all truly different, each representing different hierarchies.

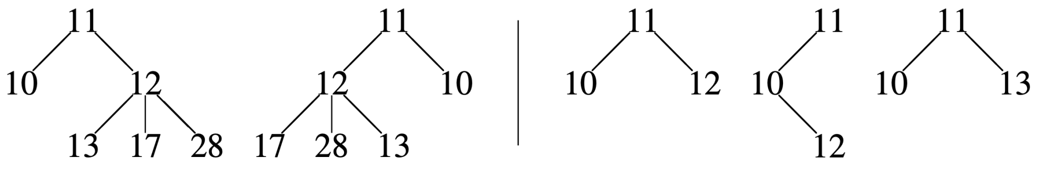


Figure 3. Two sets of hierarchical charts.

For companies with small numbers of departments, determining whether two charts are the same is relatively simple, but for larger companies it can be rather difficult. Don has sized up this problem and decided he needs some software to solve this problem for him.

Input

Input consists of two lines, each representing one hierarchical chart. The syntax for a hierarchical chart is the following:

1. a department number `d` by itself is a hierarchical chart
2. the string `d (c1) (c2) ... (cn)` is a hierarchical chart, where `d` is a department number and `c1, c2, ..., cn` are hierarchical charts.

Case 2 represents the case where department `d` is in charge of the departments at the top of the hierarchies `c1, c2, ..., cn`. Each input line will contain at most 100 000 departments and no department will be in charge of more than 100 other departments. All department numbers are integers between 1 and 1 000 000. We say that a hierarchical chart described by case 1 has leadership depth 1, and a hierarchical chart described by case 2 has leadership depth equal to one plus the maximum leadership depth of the hierarchical charts `c1`, `c2`, ..., `cn`. The input will have a leadership depth at most 1 000. There may be any number of spaces on either side of an opening or closing parenthesis.

Output

Output `Yes` if the two hierachical charts are similar and `No` otherwise.

Example

Input	Output
`11 (10) (12 (13) (17) (28))`	`Yes`
`11 (12 (17) (28) (13)) (10)`	
`11 ( 10 ) ( 12 )`	`No`
`11(10(12))`	
`11 (10) (12)`	`No`
`11 (10) (13)`	

Problem K. Which Warehouse?

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

Anaconda Inc. has a problem in several of the cities where its warehouses are located. The generic problem is the following: each city has n warehouses storing $m \leq n$ different types of products distributed randomly among the warehouses (the CEO of Anaconda say the storage is not random but part of a larger master plan, but who's kidding who here). What they want to do is consolidate the warehouses by selecting m of them to each store one of the m products. They could randomly select the m warehouses, but even the CEO knows that's probably not the smartest approach to this problem, since there is a cost in transferring the products to their designated new warehouses. What they want to do is to select the m warehouses and assign them each a product so as to minimize the total of all the distances that the products must travel.

For example, consider this situation shown in Figure 4 (which corresponds to Example Input 1). The figure shows three warehouses W1, W2 and W3, two products A and B, the amount of each product in each warehouse, and the distances between the warehouses. If we assign A to the W1 warehouse and B to the W2 warehouse, the total distance to move all the A's to W1 is $0(3) + 7(5) = 35$ and the total distance to move all the B's to W2 is $10(3) + 3(3 + 5) = 54$ for a total cost of 89 (note that the shortest path to move all the B's from W3 to W2 goes through W1). However, the best solution is to assign A to W3 and B to W1 which results in a total cost of only 58.

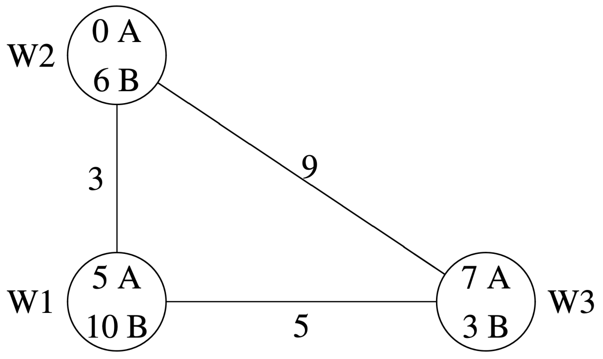


Figure 4. Example warehouse and product layout

Input

Input begins with two positive integers n m $(n \leq 1000, m \leq n)$ indicating the number of warehouses and products, respectively. Following this are n lines each with m non-negative integers. The i^{th} value on the j^{th} of these lines indicates the amount of product i stored in warehouse j. Finally there follow n lines each with n integers. The i^{th} value on the j^{th} of these lines is either a non-negative value that specifies the length of the road between warehouse j to warehouse i, or is -1 that indicates that there is no road directly going from warehouse j to warehouse i. It is possible that the distance to travel on the road from one warehouse r to another warehouse s may not be the same as the distance to travel on the road from s to r. The distance from any warehouse to itself is always 0 and there is always at least one path between any two warehouses.

Output

Output the minimum distance to move all the products using the optimal assignment of products to warehouses.

Example

Input	Output

3 2
5 10
0 6
7 3
0 3 5
3 0 9
5 9 0
	58

3 2
5 10
0 6
7 3
0 -1 5
-1 0 9
5 9 0
	124