Problem A. An Easy-Peasy Problem

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

Travis was recently saddled with writing the easy problem for a competitive programming contest aimed at high school students. Writing an easy problem is a bit of a challenge for Travis; a problem labeled as "Medium Difficulty Level" by Travis typically ends up being the hardest problem of the set!

But this time things were different. When the contest was over, Travis was happy that **everyone** solved his problem. However, some of the other judges informed Travis a problem is easy only if at least half of the contestants solve the problem in the first half of the contest.

Travis has some data on his problem, and now he wants to determine: was his problem truly easy?

In programming competitions, the term "*solves*" is used to refer to the number of correct submissions for a problem, i.e., that many contestants solved the problem.

Given the number of *solves* for Travis's problem at the end of the first half and at the end of the contest, determine whether at least half the people solved the problem in the first half. Note that *solves* at the end of the contest includes *solves* of the first half as well.

Input

There is only one input line; it contains two integers, s_1 and s_2 ($0 \leq s_1 \leq s_2$; $1 \leq s_2 \leq 100$), representing *solves* of Travis's problem at the end of the first half and at the end of the contest, respectively.

Output

Print "`E`" (quotes for clarity) if the problem is considered easy by the other judges' standards. Print "`H`" (quotes for clarity) if the problem is not considered easy by the other judges' standards. Everyone knows that if Travis is not writing an easy problem, the problem is probably hard!

Example

Input	Output
10 20	E
6 13	H
15 22	E

Problem B. Age Expression

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

Dr. O has two granddaughters (named Alyssa and Konari) and they keep Dr. O young!

When people ask Dr. O how old he is, rather than giving one positive integer (his age), Dr. O provides two positive integers a and k; Dr. O's age can then be computed using the expression ($a \times$ Alyssa's age) + ($k \times$ Konari's age).

Given three positive integers (Dr. O's age, Alyssa's age, and Konari's age), determine if the positive integers a and k exist.

Input

There is only one input line; it provides (respectively) Dr. O's age, Alyssa's age, and Konari's age. Assume that 1 $\leq$ Konari's age $<$ Alyssa's age $<$ Dr. O's age $\leq$ 150.

Output

Print 1 (one) if a and k exist, 0 (zero) otherwise. Note that a and k must both be greater than zero.

Example

Input	Output
69 9 1	1
76 11 7	1
50 9 3	0
70 10 5	1
10 7 2	0

Problem C. Increasing Sublist

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

Given a list of numbers, we define a *sublist* as one or more consecutive elements in the list. An *increasing sublist* is when the consecutive elements are in strictly increasing order, i.e., each element is greater than the element to its left in the sublist (except the first element in the sublist which does not have an element to its left).

Given a list, find the length (number of elements) of the longest increasing sublist, i.e., the length of the sublist with the most number of elements.

Input

The first input line contains an integer, n ($1 \leq n \leq 30$), indicating the number of elements in the list. The next input line provides the n elements in the list. Assume each element is between 1 and 100, inclusive.

Output

Print the length (number of elements) of the longest increasing sublist.

Example

Input	Output
6 5 7 2 4 6 3	3
15 10 70 80 5 5 5 15 20 30 40 60 9 8 70 80	6

Problem D. Let's Portmanteau

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

From "yourdictionary.com": Portmanteau, pronounced "port-man-tow," refers to a new word made from two words and their meanings. For example, the portmanteau *brunch* refers to a combined meal of breakfast and lunch, and *spork* is a mix between a spoon and a fork.

You are to combine two words using a simplified approach.

Rules for combining:

1. Get the first letter of the first word regardless of what (vowel or consonant) it is. Then, starting from the second letter of the first word, get letters moving right until you reach a vowel. If no vowels while moving right, all letters will be taken. If there is a vowel moving right, let's call it v_1.
2. Get the last letter of the second word regardless of what (vowel or consonant) it is. Then, starting from the letter next to the last letter of the second word, get letters moving left until you reach a vowel. If no vowels while moving left, all letters will be taken. If there is a vowel moving left, let's call it v_2.
3. Combine the two words by listing the letters taken from the first word (Step 1), followed by a vowel (let's call it the *merging vowel*), followed by the letters taken from the second word (Step 2). The *merging vowel* is as follows:
   - If the second word has a vowel while moving left (i.e., v_2 exists), v_2 is the merging vowel.
   - If the second word does not have a vowel while moving left (i.e., v_2 does not exist) but the first word has a vowel while moving right (i.e., v_1 exists), v_1 is the merging vowel.
   - If v_1 and v_2 don't exist (i.e., neither word has a vowel while moving right/left), use the letter "o" as the merging vowel.

Assume the vowels are "aeiou".

Input

There are two input lines: the first line provides the first word and the second line provides the second word. Assume that each word starts in column 1, is at least 4 letters and at most 20 letters, and contains only lowercase letters.

Output

Print the combined word on one output line.

Example

Input	Output
abcdefun ghijku	abcdijku
abcdefun ghmn	abcdeghmn
abycd fgyhu	abycdofgyhu

Problem E. Weighted Window Sums

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

Given a sequence of numbers, we define a window within the sequence to be a contiguous subsequence of those numbers. For example, in the sequence [3, 6, 2, 3, 5, 4], there are four windows of size 3: [3, 6, 2], [6, 2, 3], [2, 3, 5] and [3, 5, 4]. We call window i of size k the window which starts with the i^{th} value in the list and includes exactly k consecutive values, ending with the $(i + k - 1)^{\text{th}}$ value in the list.

For a window with values $a_1, a_2, \ldots, a_k$, define its weighted window sum to be:

$$1a_1 + 2a_2 + 3a_3 + \ldots + ka_k$$

For the four window sums described above, the corresponding weighted window sums are:

$$\begin{array}{ll} [3, 6, 2] \rightarrow 1 \times 3 + 2 \times 6 + 3 \times 2 = 21 & \text{(Rank 2)} \\ [6, 2, 3] \rightarrow 1 \times 6 + 2 \times 2 + 3 \times 3 = 19 & \text{(Rank 1)} \\ [2, 3, 5] \rightarrow 1 \times 2 + 2 \times 3 + 3 \times 5 = 23 & \text{(Rank 3)} \\ [3, 5, 4] \rightarrow 1 \times 3 + 2 \times 5 + 3 \times 4 = 25 & \text{(Rank 4)} \end{array}$$

For each window of a given size within a sequence of numbers, we sort those windows in increasing order of weighted window sum, breaking ties by the starting index of the window, from smallest index to largest index.

Given a sequence of integers and the size of a window, sort each window of the given size in the sequence by weighted window sum in increasing order, breaking ties by the starting index of the window.

Input

The first input line contains two integers: n ($1 \leq n \leq 2 \cdot 10^5$) and k ($1 \leq k \leq n$), representing the length of the sequence and the size of the window. Each of the next n input lines will contain one number of the sequence, in order. Each of these values will be in between 1 and 10^8, inclusive.

Output

Print a sorted list of the windows, with one window per line. For each window, output its starting index, followed by the weighted window sum of that window.

Example

Input	Output
6 3 3 6 2 3 5 4	2 19 1 21 3 23 4 25
7 3 2 3 2 3 2 4 1	5 13 1 14 3 14 2 16 4 19

Problem F. Close Triangles

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

When dealing with children, you want to divide things as evenly as possible. In particular, the child who receives the least will compare themselves to the child who receives the most, so you'd like these two values to be as close to each other as possible.

Given $3n$ points, form n triangles (using each point exactly once) such that the difference between the largest triangle (in area) and the smallest triangle (in area) is as small as possible. That is, we want these two triangles to be as close to each other (in area) as possible.

Input

The first input line contains an integer, n ($2 \leq n \leq 5$), indicating the number of triangles to be formed. This is followed by $3n$ input lines. Each of these lines provides the x and y coordinates of a point. Assume that all of these input values are integers between 1 and 10^3, inclusive. Also assume that all the points are distinct and we can form n triangles. It is also guaranteed that no triplet of the given points will be colinear.

Output

Print the difference (in area) between the largest and smallest triangles, rounded to one decimal point, e.g., 0.74 should be printed as 0.7 and 0.75 should be printed as 0.8.

Example

Input	Output
2 1 1 2 1 2 2 4 2 4 3 5 3	0.0
3 3 3 2 5 9 3 3 2 2 1 7 2 10 8 10 6 1 10	1.0

Problem G. Rightsizing

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

Many tech companies have realized that they have "over-hired." Now, looking at their books, they realize they might have to let go of some employees. In order to prioritize their bottom line, a company will always fire the employee who is making the greatest annual salary at the time. If multiple employees are making the same maximal annual salary, then the person whose name comes first alphabetically will be fired.

Of course, to improve morale, employees can still get raises, which makes figuring out who to fire a bit more tricky.

Given a set of employees and their initial annual salaries, followed by a sequence of actions (either the firing of an employee or giving a raise to a current employee), determine for each firing event, which employee got fired.

Input

The first input line contains two integers: n ($1 \leq n \leq 10^5$) representing the number of employees in the company, and a ($1 \leq a \leq 2 \cdot 10^5$) representing the number of actions taken.

The next n input lines will each contain a string, e, representing an employee's name, followed by an integer, s ($1 \leq s \leq 10^9$), representing employee e's annual salary in dollars. These n lines represent the initial status of the company's employees and their salaries. Each name will be a unique string and will contain 1-20 lowercase letters (starting in column 1).

The next a input lines will each contain information about an action.

If the first integer on one of these input lines is 1, that means that an employee is getting a raise. This will be followed by a string e, representing the employee getting the raise and an integer, r ($1 \leq r \leq 10^9$), representing the raise amount for employee e. It is guaranteed that each employee getting a raise was listed in the original input and has not been fired yet.

If the first integer on one of these lines is 2, that means the company is firing the employee making the maximum salary. If there is more than one such person, then the person getting fired is the one whose name comes first alphabetically. Since there are n employees, assume that the maximum number of times action 2 will appear in the input is n.

Output

For each firing event (action 2 in the input), print a single line with the name of the employee who got fired and how much they were making annually at the time they were fired, separated by a space.

Example

Input	Output
5 6 eduardo 6000000 mark 7000000 andrew 5000000 dustin 1000000 chris 500000 2 1 andrew 2000000 1 dustin 5000000 2 2 2	mark 7000000 andrew 7000000 dustin 6000000 eduardo 6000000
10 15 a 6 b 5 c 4 d 2 e 1 f 3 g 8 h 10 i 7 j 10 1 c 4 1 d 9 2 1 e 3 1 h 1 1 b 6 2 2 1 f 6 1 a 3 2 1 i 2 2 1 e 5 2	d 11 b 11 h 11 j 10 a 9 e 9

Problem H. Brightline - Back to the Future

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

Many of the UCF programming team coaches are getting old and are looking for any way possible to regain their youth. There's a new train in town, the Brightline. Rumor has it that if a person takes some of the trains, he or she can actually get younger!

Brightline runs two types of trains: the *blue* line and the *red* line. On each blue line, train passengers age (get older) by some amount of time. One each red line, train passengers go back in time (get younger) by some amount of time.

Each train line is directed, connecting two cities, a *source* to a *destination*, and is labeled as either red or blue, and has an associated number with it, representing the number of minutes a passenger will age (if a blue line) or move back in time (if a red line) if they take that line. Assume that any time you arrive in a city, you can immediately take any of the trains leaving that city.

The train system doesn't allow for going back in time indefinitely; there will be no possible way to start at any city and end at that same city younger than you were before. This would create a time paradox too difficult for even the UCF coaches to solve.

All of the coaches currently reside in Orlando (which will be location 1 for the purposes of this problem).

Given a list of all Brightline train lines (each line will have a source city, destination city, a red or blue label, and a positive integer representing how much one ages on the line), determine all of the destination cities that someone, starting in city 1 (Orlando) can travel to, such that when they arrive at their destination, they will be younger than when they started.

Input

The first input line contains two integers: n ($1 \leq n \leq 5 \cdot 10^3$) representing the number of cities, and m ($1 \leq m \leq 10^4$) representing the number of Brightline train lines. The cities are labeled 1 to n, with 1 representing the starting city, Orlando.

The next m input lines will each contain information about a single train line. Each of these lines will have four pieces of information about a single train line:

- s ($1 \leq s \leq n$), starting city of the train line
- e ($1 \leq e \leq n,\ s \neq e$), ending city of the train line
- t ($t =$ 'b' or $t =$ 'r'), representing whether the line is a blue line or a red line
- a ($1 \leq a \leq 10000$), representing the corresponding number of minutes one ages (forward if blue, backwards if red) if they take that line.

Output

Print each destination city, in increasing numerical order, that one could arrive at younger than when they started in Orlando (city 1).

Example

Input	Output
5 8 1 2 b 7 1 3 b 3 3 2 b 4 2 4 r 2 3 5 r 5 5 2 b 1 4 3 b 6 5 4 b 4	2 4 5
5 9 1 2 b 3 1 5 r 4 1 3 b 8 2 4 b 1 2 5 b 7 3 2 b 4 4 1 b 2 4 3 r 5 5 4 b 6	3 5

Problem I. Know your ABC's

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

UCF has started running a pre-school, where it is teaching all the students their ABC's! Naturally, all of the blocks the kids have to play with have one of the three letters on them. The kids like arranging the blocks, but don't like any arrangements where two of the same letters are in a row (i.e., right next to each other).

For example, if a kid had 2 `A` blocks, 3 `B` blocks and 1 `C` block, a valid arrangement of the blocks would be `BACBAB`, but `CBABBA` would not be valid because the letter `B` appears twice in a row.

The preschool teacher is wondering: how many different valid orders can the blocks be arranged? Two arrangements are considered different if there is a different *letter* in the same corresponding positions in the two arrangements.

Given the number of blocks showing the letter `A`, the number of blocks showing the letter `B`, and the number of blocks showing the letter `C`, determine the number of arrangements of all of the blocks in a row, with no two consecutive blocks showing the same letter. Since the answer might be quite large, determine the result modulo $10^9 + 7$.

Input

There is only one input line; it contains three integers: a ($1 \leq a \leq 100$), b ($1 \leq b \leq 100$), and c ($1 \leq c \leq 100$), representing the number of blocks with the letter `A`, the number of blocks with the letter `B` and the number of blocks with the letter `C`, respectively.

Output

Print the number of valid arrangement of all of the blocks in a row without two of the same letter appearing consecutively modulo $10^9 + 7$.

Example

Input	Output
2 3 1	10
15 2 7	0

Problem J. Cut the Cake

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

Geometry Refresher:

- A *simple polygon* is a polygon that does not intersect itself and has no holes. That is, it is a flat shape consisting of straight, non-intersecting line segments or "sides" that are joined pairwise to form a single closed path.
- A polygon is *convex* if every line that does not contain any edge intersects the polygon in at most two points.
-

Given a cake in the shape of simple convex polygon, we would like to cut the cake into two pieces; the single cut (straight line) must go thru two of the polygon vertices. Find the two vertices that will make the two pieces as close (in area) as possible, i.e., the difference in the area for the two pieces is the minimum.

Input

The first input line contains an integer, n ($4 \leq n \leq 100$), indicating the number of vertices in the polygon. This is followed by n input lines. Each of these lines provides the x and y coordinates of a vertex in the polygon (the polygon vertices will be provided in a clockwise order). Assume that all of these input values are integers between 1 and 10^3, inclusive. Also assume that all the vertices are distinct.

Output

Print the difference (in area) between the largest and smallest pieces, rounded to one decimal point, e.g., 0.74 should be printed as 0.7 and 0.75 should be printed as 0.8.

Example

Input	Output
4 6 4 3 4 4 5 5 5	1.0
5 10 10 8 15 13 16 18 15 16 10	15.0