Problem A
As Easy as CAB

We all know how to alphabetize a list when you know the alphabet: One word may be a prefix of another longer word, in which case the shorter word always comes before the longer word. With any other two words there must be a first place in the words where their letters differ. Then the order of the words is determine by the lexicographical order of these first differing letters.

How about the reverse problem: Can you find the lexicographic order of the alphabet from an ordered list of words? Suppose an alphabet exists where the following list of word strings is given in lexicographical order:

```
cab
cda
ccc
badca
```

It is clear that `c` comes before `b` in the underlying alphabet because `cab` comes before `badca`. Similarly, we know `a` comes before `d`, because `cab` < `cda`, `a` comes before `c` because `cab` < `ccc`, and `d` comes before `c` because `cda` < `ccc`. The only ordering of the 4 alphabet characters that is possible is `adcb`.

However, it may be that a list contains inconsistencies that make it impossible to be ordered under any proposed alphabet. For example, in the following list it must be that `a` comes before `b` in the alphabet since `abc` < `bca`, yet it also must be that `b` comes before `a` in the alphabet since `bca` < `aca`.

```
abc
bca
cab
aca
```

Finally, some lists may not provide enough clues to derive a unique alphabet order, such as the following:

```
dea
cfb
```

In this list, `d` comes before `c` but we don't know about the relative positions of any of the other letters, so we are unable to uniquely discern the order of the alphabet characters.

Input

The first line of input will contain L and N, separated by a space, where L is a lowercase character $\mathtt{b} \le L \le \mathtt{z}$ representing the highest character in the traditional alphabet that appears in the derived alphabet, and N is a integer $1 \le N \le 1\,000$ that is equal to the number of strings in the list. Each of the next N lines will contain a single nonempty string of length at most $1\,000$, consisting only of characters in the derived alphabet. No two strings will be the same.

Output

If the input is consistent with a unique ordering of the alphabet, output a string that designates that ordered alphabet. If the data is inconsistent with any ordering, output `IMPOSSIBLE`. If the data is consistent with multiple orderings, output `AMBIGUOUS`.

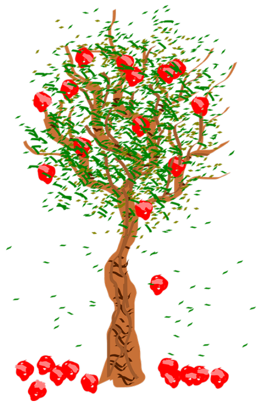
Sample Input 1	Sample Output 1
d 4 cab cda ccc badca	adcb

Sample Input 2	Sample Output 2
c 4 abc bca cab aca	IMPOSSIBLE

Sample Input 3	Sample Output 3
f 2 dea cfb	AMBIGUOUS

Sample Input 4	Sample Output 4
e 5 ebbce dbe adcd bc cd	edabc

Problem B
Falling Apples



You have a 2D rectangular grid. Each grid cell contains either an apple, an obstacle, or is empty. Empty cells are denoted as '`.`', apples as '`a`', and obstacles as '`#`'. You are to implement a simulation of gravity, based on the following rules:

- The obstacles do not move.
- Whenever there is an empty cell immediately below an apple, the apple moves into the empty cell.

Print out the final configuration of the board after all apples reach their final locations. Merely iterating the gravity rule, a step at a time, will likely take too long on large datasets.

Input

The input begins with a line containing integers R and C, designating the number of rows and columns of the grid, such that $1 \le R \le 50\,000$ and $1 \le C \le 10$. The first line is followed by R additional lines, each designating a row of the grid, from top to bottom. Each line has C characters, each of which is either '`.`', '`a`', or '`#`'.

Output

Output R grid lines displaying the final state.

Sample Input 1	Sample Output 1

```
3 3
aaa
#..
..#
```
	```
	a..
	#.a
	.a#
	```

Sample Input 2	Sample Output 2

```
4 5
aaa.a
aa.a.
a.a..
...a.
```
	```
	.....
	a....
	aaaa.
	aaaaa
	```

This page is intentionally left (almost) blank.

Problem C
Square Deal

Given the dimensions of three rectangles, determine if all three can be glued together, touching just on the edges, to form a square. You may rotate the rectangles. For example, Figure C.1 shows successful constructions for the first two sample inputs.

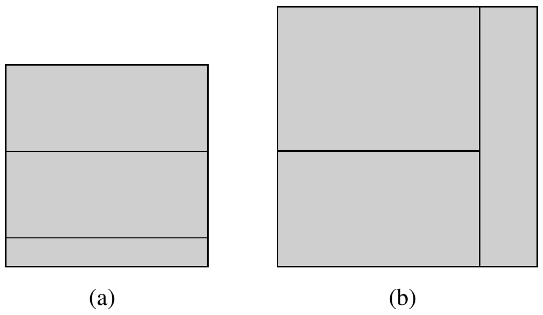


Figure C.1: Constructions for the first two examples

Input

The input consists of three lines, with line j containing integers H_j and W_j, designating the height and width of a rectangle, such that $100 \geq H_j \geq W_j \geq 1$, and such that $H_1 \geq H_2 \geq H_3$.

Output

Output a line saying `YES` if they can be glued together to form a square. Output `NO` otherwise.

Sample Input 1

```
7 3
7 1
7 3
```

Sample Output 1

```
YES
```

Sample Input 2

```
9 2
7 4
7 5
```

Sample Output 2

```
YES
```

Sample Input 3

```
3 1
3 2
3 3
```

Sample Output 3

```
NO
```

This page is intentionally left (almost) blank.

Problem D
Buggy Robot

Your friend just bought a new programmable robot and has asked for your help. The robot operates in a 2D grid that may contain obstacles. The environment has a known start location and a known goal location. The robot is controlled with a string consisting of commands `L`, `R`, `U`, and `D`, which respectively instruct the robot to move one square to the left, right, up or down in the grid. The robot will ignore a command (but continue with future commands) if the command would cause it to leave the grid or to run into an obstacle. If the robot were to reach the goal position, it immediately stops its program with success (even if further commands exist).

Your friend has already programmed the robot with a command string, but the problem is that your friend is not a very good programmer, and so it may be that the commands do not lead the robot successfully to the goal. You would like to fix the string so that the robot will reach the goal, but you do not want your friend to feel bad, so you are hoping to make as few changes to the program as are needed. A single change consists either of deleting an arbitrary character of the command string, or inserting an arbitrary command anywhere within the string.

As an example, if we consider Sample Input 1, we see that your friend's command string of `DRRDD` does not succeed. The initial `D` moves the robot one spot down. From there, the `R` (and the subsequent `R`) are ignored because of the obstacle to the robot's right. The subsequent `D` moves the robot down once again and the final `D` is ignored. However, by deleting the initial `D`, we can rely on the command string `RRDD` which does successfully lead the robot to the goal.

If we consider Sample Input 2, we find that your friend's original command string `LDLDLLDR` is flawed. However, if we insert the single command `U` after the fifth command, the resulting string `LDLDLULDR` successfully guides the robot to the goal. It starts by moving left-down-left; the next down command is ignored because the robot is not allowed to leave the grid. The subsequent left-up-left completes the path to the goal (and the final `DR` commands are ignored as the robot stops immediately upon reaching the goal).

With Sample Input 3, your friend's command string can be corrected with two changes, for example with the new command `ULDLDLLDLR` (although this is not the only way to use two changes to produce a valid sequence).

Input

The first line of the input contains the two integers H and W that respectively define the height and width of the grid such that $1 \leq H, W \leq 50$. The next H lines each has W characters, describing the corresponding row of the grid, such that the character '`S`' designates the starting location, '`G`' designates the goal location, '`#`' designates an obstacle, and '`.`' designates an empty location. There will always be precisely one '`S`' and one '`G`' in the grid, and there will always exist an unobstructed path from the start position to the goal position. The final line of the input contains your friend's original command string consisting of between 1 and 50 characters, inclusive.

Output

Output a single integer indicating the minimum number of changes that are needed to fix the program.

Sample Input 1

```
3 3
S..
.#.
..G
DRRDD
```

Sample Output 1

```
1
```

Sample Input 2

```
3 7
.......
.G.#.S.
.......
LDLDLLDR
```

Sample Output 2

```
1
```

Sample Input 3

```
3 7
.#.....
.G.##S.
.......
LDLDLLDR
```

Sample Output 3

```
2
```

Sample Input 4

```
2 4
S.#.
#..G
RRUUDDRRUUUU
```

Sample Output 4

```
0
```

Problem E
Construction Toy

Alaa fondly remembers playing with a construction toy when she was a child. It consisted of segments that could be fastened at each end. A game she liked to play was to start with one segment as a base, placed flat against a straight wall. Then she repeatedly added on triangles, with one edge of the next triangle being a single segment already in place on her structure, and the other two sides of the triangle being newly added segments. She only added real triangles: never with the sum of the lengths of two sides equaling the third. Of course no segment could go through the wall, but she did allow newly added segments to cross over already placed ones. Her aim was to see how far out from the wall she could make her structure go. She would experiment, building different ways with different combinations of some or all of her pieces. It was an easy, boring task if all the segments that she used were the same length! It got more interesting if she went to the opposite extreme and started from a group of segments that were all of distinct lengths.

For instance, the figures below illustrate some of the structures she could have built with segments of length 42, 40, 32, 30, 25, 18 and 15, including one that reaches a maximum distance of 66.9495 from the wall.

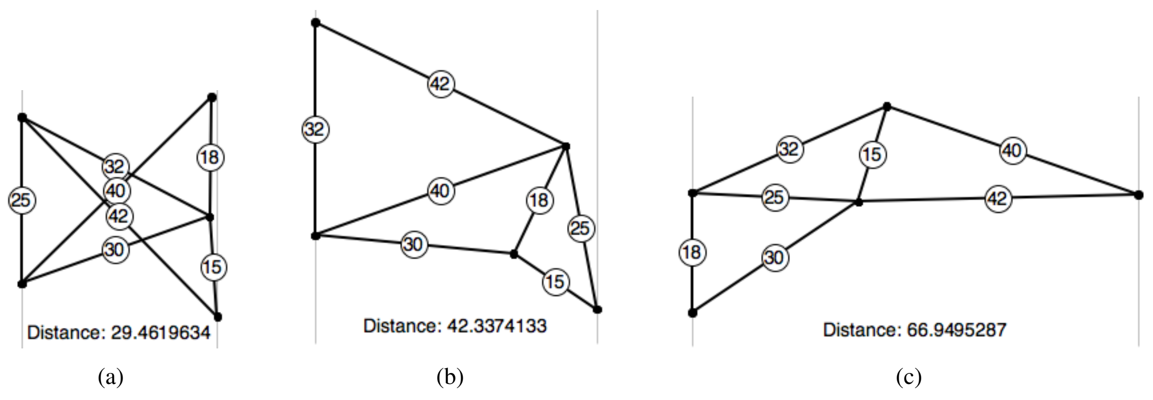


Figure E.1: Candidate constructions for example lengths, with the wall at left in each

Now, looking back as a Computer Science student, Alaa wondered how well she did, so she has decided to write a program to compute the maximum distance given a set of segment lengths.

Input

The input is a single line of positive integers. The first integer n designates the number of segments, with $3 \leq n \leq 9$. The following n integers, $\ell_1 > \ell_2 > \cdots > \ell_n$ designate the lengths of the segments, such that $1 \leq \ell_j \leq 99$ for all j. The lengths will permit at least one triangle to be constructed.

Output

Output is the maximum distance that one of Alaa's structures can reach away from the wall, stated with a relative or absolute error of at most 10^{-2}. The input data is chosen so that any structure acheiving the maximum distance has all vertices except the base vertices at least 0.0001 from the wall.

Sample Input 1

```
3 50 40 30
```

Sample Output 1

```
40
```

Sample Input 2

```
4 50 40 30 29
```

Sample Output 2

```
40
```

Sample Input 3

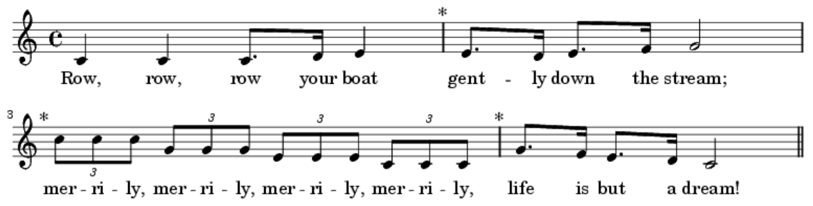
```
7 42 40 32 30 25 18 15
```

Sample Output 3

```
66.9495287
```

Problem F
Around and Around We Go

A *round* is a musical arrangement in which two or more voices repeat the same melodic line, at times offset from one another. One of the more famous rounds is the nursery rhyme "Row, Row, Your Boat", shown here.


Row, row, row your boat gent - ly down the stream;
mer - ri - ly, mer - ri - ly, mer - ri - ly, mer - ri - ly, life is but a dream!

In western music, an integer count of time units can be assigned to each syllable to indicate how long that syllable is sung. A rest of one or more time units can be treated as a syllable that simply isn't sung. If you know the time allocated to each syllable, and the time offset at which a second voice begins singing the song, you can determine which words will overlap in the round.

You are to write a program to display a two-voice round so that syllables that are sung simultaneously in the two voices appear vertically aligned. For each line of the original input, there are to be two lines of output: one for the original line sung by the first voice and one to display syllables (if any) that are started by the second voice during the time period where the first voice is singing the indicated line. Syllables for each voice in the output must be separated by one or more underscores ('_'), with each syllable displayed as far to the left as possible, subject to the following constraints:

- Consecutive syllables on a line are separated by at least one '_' character.
- Two syllables that begin at the same time in their respective voices are displayed with their leftmost characters in the same column.
- Consider syllables S1 and S2, either sung by the same or different voices, that are displayed within the same pair of output lines. If S2 is sung beginning k time units after S1 begins, for $k \geq 1$, then the first character of S2 must be displayed at least k columns to the right of the first character of S1.

In some cases there will be a first-voice line when no second-voice syllables are started. Print '/' for the second voice instead of an empty line.

It is possible (in fact, likely), that not all syllables of the second voice will be printed, as only those syllables that start while the first voice is active are to be displayed.

Input

The first line contains two integers, L and D, such that $1 \leq L \leq 10$ indicates the number of lines in the song and $0 \leq D \leq 128$ indicates the delay, in time units, between the time when the first voice signs the first syllable and the time when the second voice begins singing the first syllable.

The remainder of the input consists of L *pairs* of lines. The first line in each pair contains the syllables of that line of the song. Adjacent syllables in the input will be separated by a single space The syllables are strings of any non-whitespace characters other than underscores or '/'. This line contains at most 80 characters.

The second line in each pair will consist of positive integers, one per syllable from the first line of the pair, indicating the time allocated to the corresponding syllables. Each such integer t will satisfy $1 \leq t \leq 128$.

Output

For each dataset, display $2L$ lines corresponding to the two voices in the round, as described above.

Sample Input 1

```
2 16
Hot cross buns! = Hot cross buns! =
4 4 4 4 4 4 4 4
One a pen- ny, Two a pen- ny, Hot cross buns! =
2 2 2 2 2 2 2 2 4 4 4 4
```

Sample Output 1

```
Hot_cross_buns!_=___Hot_cross_buns!_=
____________________Hot_cross_buns!_=
One_a_pen-_ny,_Two_a_pen-_ny,_Hot___cross____buns!_=
Hot___cross____buns!_=________One_a_pen-_ny,_Two_a_pen-_ny,
```

Sample Input 2

```
5 32
And ev- ry-
1 1 1
one neath a vine and fig tree, = shall live in
2 1 1 2 2 2 2 1 1 1 1
peace and un- a- fraid. =
2 2 2 2 6 2
And in- to plow shares beat their swords.
2 1 1 2 2 2 2 4
Na- tions shall learn war no more =
2 2 2 2 2 2 1 3
```

Sample Output 2

```
And_ev-_ry-
/
one_neath_a_vine_and_fig_tree,_=_shall_live_in
/
peace_and_un-_a-_fraid.___=
______________________And_ev-_ry-
And_in-___to_plow_shares_beat_their_swords.
one_neath_a__vine_and____fig__tree,_=_shall_live_in
Na-___tions_shall_learn_war_no_more_=
peace_and___un-___a-____fraid._______=
```

Problem G
The Calculus of Ada

While mostly known for the programs she wrote for Charles Babbage's Analytic Engine, Augusta Ada King-Noel, Countess of Lovelace, described how the method of finite differences could be used to solve all types of problems involving number sequences and series. These techniques were implemented in Babbage's Difference Engine.

The algorithm: If we compute the difference between consecutive values in a numeric sequence, we will obtain a new sequence which is related to the derivative of the function implied by the original sequence. For sequences generated from first-order polynomials (linear functions) the successive differences will be a list of identical values, (i.e., a constant difference). For second-order polynomial functions the lists of differences will be a new sequence whose values change linearly. In turn, the list of differences of the values in this generated list (i.e., the finite differences of the list of differences) will be constant, and so on for higher-order polynomials. In general the n^{th} row of differences will be constant for an n^{th} degree polynomial.

For example, the first-order polynomial $3x + 3$ produces the sequence below at $x = 0, 1, 2, 3, 4$, and the first differences are shown on the following line.

```
3       6       9      12      15
    3       3       3       3
```

As another example, the polynomial x^2, if evaluated at inputs $x = 3, 5, 7, 9$, produces the sequence below.

```
9      25      49      81
   16      24      32
        8       8
```

Furthermore, if we consider a minimum-order polynomial that produces the original sequence, its value at the next regularly spaced input can be predicted by extending the difference table.

Input

The input consists of a value n, designating the number of polynomial evaluations given with $2 \leq n \leq 10$, follwed by n values $v_1, v_2, \ldots v_n$ which represent the value of a polynomial when evaluated at n regularly spaced input values. Each v_j will satisfy $-2\,000\,000 \leq v_j \leq 2\,000\,000$ and at least two of those values will differ from each other.

Output

Output two integer values d and v_{n+1}, separated by a space. The value d must be the degree of a minimal-degree polynomial producing the original sequence, and v_{n+1} must be the value of the polynomial if evaluated at the next regularly spaced input value.

Sample Input 1

```
5 3 6 9 12 15
```

Sample Output 1

```
1 18
```

Sample Input 2

```
4 9 25 49 81
```

Sample Output 2

```
2 121
```

Sample Input 3

```
6 39 6 -3 0 3 -6
```

Sample Output 3

```
3 -39
```

Problem H
Ghostbusters 2

In the 1984 Ghostbusters^{TM} movie, the protagonists use proton pack weapons that fire laser streams. This leads to the following memorable dialog between scientists Peter Venkman and Egon Spengler:

Spengler: There's something very important I forgot to tell you.
Venkman: What?
Spengler: Don't cross the streams.
Venkman: Why?
Spengler: It would be bad.
Venkman: I'm fuzzy on the whole good/bad thing. What do you mean, "bad"?
Spengler: Try to imagine all life as you know it stopping instantaneously and every molecule in your body exploding at the speed of light.
Venkman: Right. That's bad. Okay. All right. Important safety tip.

In the 30+ years since that time, there have been several technical advances in their weapons systems:

- The laser streams have been polarized, firing either horizontally or vertically. There is no longer any danger if streams having opposite polarity cross each other. However, there will still be catastrophic results if two streams having the same orientation in any way touch each other.
- A weapon now simultaenously fires its streams in opposite directions. More specifically, a weapon has an integer power P and when fired will reach locations P units to the left and P units to the right of the ghostbuster, if fired horizontally, or P units above and below the ghostbuster if fired vertically.

When stationed at their positions, the ghostbusters can communicate to decide who will fire horizontally and who will fire vertically. They will all use the same power value P and would like to use as much power as possible without causing catastrophe. As an example, Figure H.1(a) shows a configuration of ghostbusters in which an arbitrarily large power value can be used, so long as the ghostbusters coordinate their orientations. Figure H.1(b) shows a configuration in which there is an orientation to allow power level 3, but for which no orientation allows power level 4. (Notice that the streams of the bottom-left and bottom-right ghostbusters would touch if using power level 4 with that same orientation of streams.)

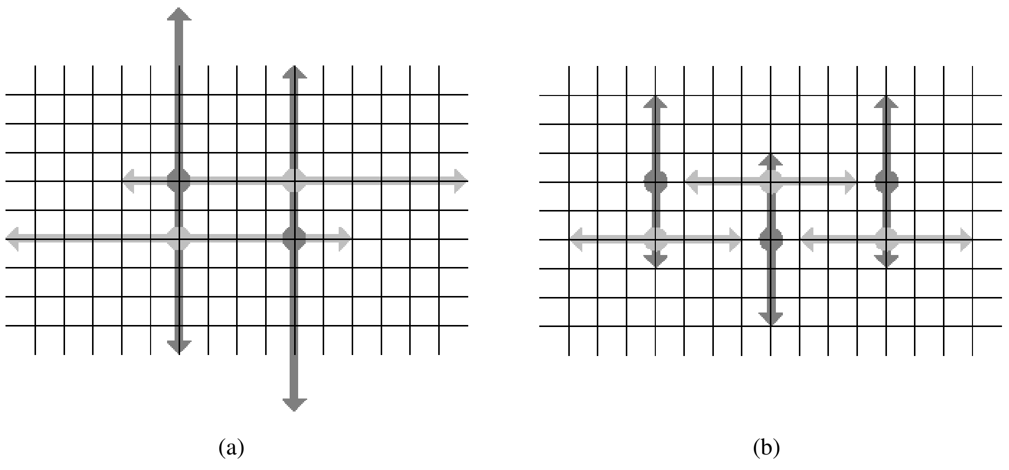


Figure H.1: Example configurations for the first two sample inputs

Input

The first line contains an integer N, such that $1 \leq N \leq 4\,000$, indicating the number of ghostbusters. Following that are N lines, each containing integers x and y which describe the location of one ghostbuster, such that $0 \leq x, y \leq 1\,000\,000$. No two ghostbusters are at the same location.

Output

If the ghostbusters can use arbitrarily large power without catastrophe, output `UNLIMITED`. Otherwise output the largest integral power value that may be safely used with appropriately chosen orientations.

Sample Input 1

```
4
5 4
9 4
5 6
9 6
```

Sample Output 1

```
UNLIMITED
```

Sample Input 2

```
6
3 3
7 3
11 3
3 5
7 5
11 5
```

Sample Output 2

```
3
```

Sample Input 3

```
6
0 0
1 0
2 0
0 1
1 1
2 1
```

Sample Output 3

```
0
```

Problem I
Postal Delivery

The postal service is interested in cutting costs as an alternative to raising the postage rates. One way to do this is by minimizing the distance traveled when delivering mail from the post office to all the required locations and returning to the post office. It may be that all the mail to be delivered does not fit on the mail truck at once, in which case the distance traveled by the truck must include travel back to the post office to reload. For simplicity, we assume a one dimensional world with the post office at the origin, and delivery locations each identified by a single coordinate. As an example, suppose a postal truck can carry up to 100 letters and that 50 letters need to be delivered to location -10, that 175 need to be delivered to location 10, and 20 delivered to location 25. A maximally efficient plan would be:

Deliver the 50 letters to location -10 (travel $2 \cdot 10$), the first 100 letters to location 10 (travel $2 \cdot 10$), the remaining 75 letters to location 10 while on the way to delivering the 20 to location 25 (travel $2 \cdot 25$). The total round-trip distance traveled is 90.

Input

The first line contains two integers, N and K, where $3 \le N \le 1000$ is the number of delivery addresses on the route, and $1 \le K \le 10\,000$ is the carrying capacity of the postal truck. Each of the following N lines will contain two integers x_j and t_j, the location of a delivery and the number of letters to deliver there, where $-1500 \le x_1 < x_2 < \cdots < x_N \le 1500$ and $1 \le t_j \le 800$ for all j. All delivery locations are nonzero (that is, none are at the post office).

Output

Output the minimum total travel distance needed to deliver all the letters and return to the post office.

Sample Input 1

```
3 100
-10 50
10 175
25 20
```

Sample Output 1

```
90
```

Sample Input 2

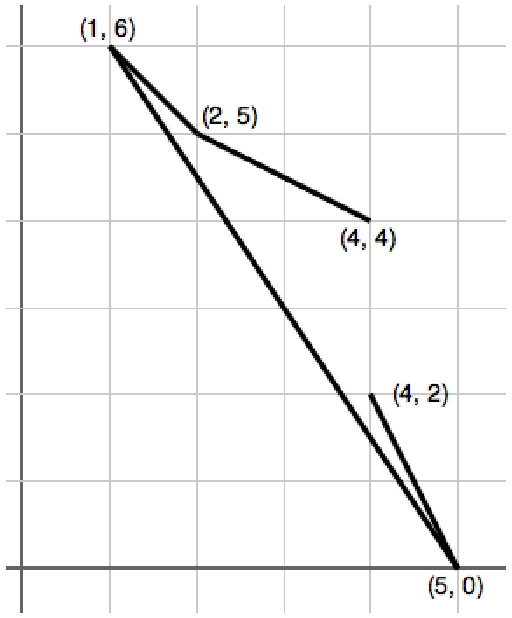
```
5 3
-1002 800
-1001 800
-1000 800
-999 800
-998 800
```

Sample Output 2

```
2668000
```

This page is intentionally left (almost) blank.

Problem J
Windy Path



Consider following along the path in the figure above, starting from $(4, 4)$ and moving to $(2, 5)$. Then the path turns rightward toward $(1, 6)$, then sharp left to $(5, 0)$ and finally sharp left again to $(4, 2)$. If we use ‘`L`’ for left and ‘`R`’ for right, we see that the sequence of turn directions is given by ‘`RLL`’. Notice that the path does not cross itself: the only intersections of segments are the connection points along the path.

Consider the reverse problem: Given points in an arbitrary order, say $(2, 5), (1, 6), (4, 4), (5, 0), (4, 2)$, could you find an ordering of the points so the turn directions along the path are given by ‘`RLL`’? Of course to follow the path in the figure, you would start with the third point in the list $(4, 4)$, then the first $(2, 5)$, second $(1, 6)$, fourth $(5, 0)$, and fifth $(4, 2)$, so the permutation of the points relative to the given initial order would be: 3 1 2 4 5.

Input

The first line of the input contains an integer N, specifying the number of points such that $3 \leq N \leq 50$. The following N lines each describe a point using two integers x_i and y_i such that $0 \leq x_i, y_i \leq 1000$. The points are distinct and no three are collinear (i.e., on the same line). The last line contains a string of $N - 2$ characters, each of which is either ‘`L`’ or ‘`R`’.

Output

A permutation of $\{1, \ldots, N\}$ that corresponds to a nonintersecting path satisfying the turn conditions. The numbers are to be displayed with separating spaces. (There are always one or more possible solutions, and any one may be used.)

Sample Input 1

```
5
2 5
1 6
4 4
5 0
4 2
RLL
```

Sample Output 1

```
3 1 2 4 5
```

Sample Input 2

```
4
1 1
2 1
1 2
2 2
LR
```

Sample Output 2

```
3 1 4 2
```