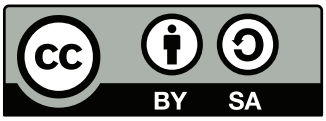
Freshmen Programming Contest 2022

Contest Problem Set

May 11, 2022

Problems



A Avant-garde

Time limit: 1s

Museum of Bad Art: Blue Meanie
CC BY-NC-ND 2.0 By Chris Devers on Flickr

A new style of painting has recently exploded in popularity. The artist will start with some famous artwork as the background, and proceeds to splash big blobs of paint on the canvas. You think this new art form is terrible. To prove a point, you want to estimate the area of the nice background picture that gets ruined, because it gets covered by the blobs.

The artist has splashed n big blobs on the canvas, which are shaped like perfect circles. All the centres of the blobs lie on the x-axis, so each blob is defined by a radius and x-coordinate. The second sample input is visualized in Figure A.1.

Calculate the area of the background that is no longer visible, because of the blobs. Your answer is correct if its relative error is less than 10%. So your answer is correct if:

$$0.9 \times A_{\text{actual}} \leq A_{\text{your output}} \leq 1.1 \times A_{\text{actual}}$$

Input

The input consists of:

- One line containing a single integer n ($1 \leq n \leq 10$), the number of blobs.
- n lines follow, one for each blob, each containing two integers: x ($|x| \leq 10$), the x-coordinate of the centre of the blob and r ($1 \leq r \leq 10$), the radius of the the blob.

Output

Output the area of the background that is covered by the blobs as a decimal number. Your answer should have a relative error of at most 10%.

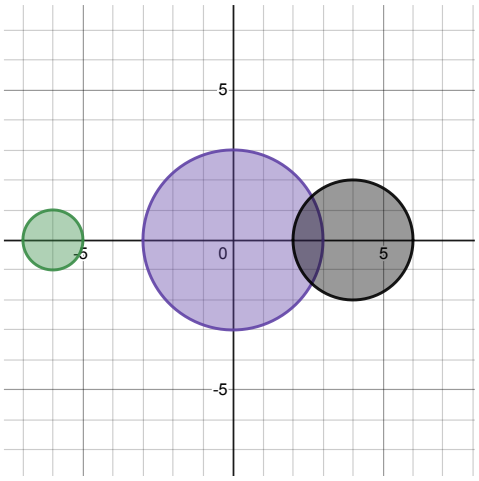


Figure A.1: Visualization of Sample Input 2. The covered area is roughly equal to 42.

Sample Input 1

```
1
1 1
```

Sample Output 1

```
3.1415926583406
```

Sample Input 2

```
3
-6 1
0 3
4 2
```

Sample Output 2

```
41.9925053413308
```

Sample Input 3

```
3
-4 3
-4 1
-4 3
```

Sample Output 3

```
28.2743338897644
```

B Balance by Elimination

Time limit: 3s

You are given a binary tree with n nodes. The nodes are conveniently numbered from 1 to n. Node 1 is the root of the binary tree.

CC-BY 2.0 By Floyd Wilde on Flickr
Unidentified bright green tree

The height of the subtree rooted at node u is:

$$h_u = 1 + \max\left(h_{\text{left child}}, h_{\text{right child}}\right)$$

If a left or right child doesn't exist, its subtree height is defined to be 0. In particular, if a node is a leaf, it has a height of 1.

You want the tree to become height-balanced. A node is height-balanced if:

$$|h_{\text{left child}} - h_{\text{right child}}| < 2$$

A binary tree is height-balanced if all its nodes are height-balanced.

Find a way to remove at most 1 leaf from the tree, such that the binary tree becomes height-balanced, or output that this is impossible. For example, the tree of the second sample input (visualized in Figure B.1) becomes balanced when removing node 5.

Input

The input consists of:

- One line containing a single integer n ($1 \leq n \leq 10^5$), the number of nodes in the binary tree.
- Then n lines follow, numbered from 1 to n. The ith line contains two integers, the labels of the left and right child of node i.

If a left child or right child does not exist, the corresponding integer is equal to 0. It is guaranteed that the input graph is a binary tree.

Output

Output a single integer:

- If the tree is already balanced, output "`balanced`".
- If it's impossible to make the tree height-balanced, output "`impossible`".
- Else, output the number of the leaf you want to remove.

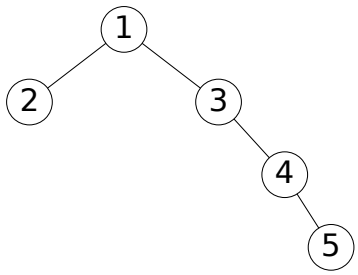


Figure B.1: Visualization of Sample Input 2.

Sample Input 1	**Sample Output 1**
4 4 2 0 0 0 0 0 3	balanced

Sample Input 2	**Sample Output 2**
5 2 3 0 0 0 4 0 5 0 0	5

Sample Input 3	**Sample Output 3**
4 2 0 3 0 4 0 0 0	impossible

C Cake Promise

Time limit: 2s

Today is the big day: professor P. Wright is organizing a programming contest. All of his students are participating, because he has promised to bake a cake for all teams that can beat him in the contest.[1] After the contest has ended, professor Wright receives a raw copy of the scoreboard, in which the teams are out of order. He is short on time: besides baking, he also has some exams to grade. Therefore, he needs a program to calculate how many teams beat him in the contest.

A Fresh Pineapple Cake, the prize for beating professor Wright.
CC-BY-NC 2.0 By Megan Duong on Flickr

The contest consists of a given number of problems, which the teams will need to solve as fast as possible. One team beats another team if the number of problems solved by the first team is strictly greater. If the number of problems solved is equal between two teams, the team that required a lower sum of minutes to solve the problems beats the other. Professor Wright is also willing to bake a cake for teams that solved the same number of problems *and* required the same sum of minutes as himself.

Input

The input consists of:

- One line containing two integers t and p ($2 \leq t, p \leq 1000$), the number of teams participating and the number of programming problems, respectively. Professor Wright counts as one of the teams.
- t lines, each containing p values, the results for each team. The first line represents the results of professor Wright. Each result is either "`X`", indicating that the team did not solve this problem, or an integer r ($1 \leq r \leq 10^6$), representing the number of minutes that this team needed to solve the problem.

Output

Output the number of teams for which professor Wright needs to bake a cake.

Sample Input 1	Sample Output 1
5 5 40 X 10 30 20 X X 20 50 40 50 40 20 30 10 X X 30 X 50 X 30 5 40 20	2

[1] All characters and events in this problem description are fictional. Any resemblance to actual lecturers is purely coincidental.

Sample Input 2	**Sample Output 2**
4 4 4 5 6 X 4 5 X 5 4 X 5 6 X 4 5 7	2

D Dale 'n' Chip

Time limit: 2s

Dale and Chip are preparing the annual Fantastic Party for Chipmunks (FPC). This year, they say, it's much more than a big feast of nuts. In fact, they have prepared a game for their fellow friends. Like any chipmunk, they want their game to be predictable. There is no room for surprises. Last thing you would want is an angry losing chipmunk. Therefore, they will secretly plan the game such that everyone will win and lose exactly one time.

Dale, Chip, and all of their friends, eager for a nice and predictable game of Rock Paper Scissors. CC-NC EmojiPng.com

The game goes like this: a group of chipmunks is arranged in a circle. One of them will start the game by turning to their right neighbour and play a round of Rock Paper Scissors. Afterwards, the right neighbour turns to its right neighbour and plays a round of Rock Paper Scissors, and so on. The game ends when everyone has played with their right neighbour exactly once.

Dale has a list of chipmunks that will attend the party, and Chip knows for each of them what power (Rock, Paper, or Scissors) they will always play—chipmunks are predictable, after all. Not everybody will be at the party at the same time, so they want to prepare the game for several sublists of the list of attendees. For every sublist, calculate the largest possible number of chipmunks picked from that sublist that can be arranged in a circle, such that everyone will be happy, no matter who starts the game.

A happy chipmunk is one that has won and lost exactly once.

Input

The input consists of:

- One line containing a single integer n ($1 \leq n \leq 10^4$), the total number of chipmunks on the list of attendees.
- n lines, each line containing a string ("`Rock`", "`Paper`" or "`Scissors`"), which represents for every attendee what power they will always play.
- One line containing a single integer q ($1 \leq q \leq 10^5$), the number of sublists Dale and Chip would like to organise games for.
- q lines, each containing two integers s and e ($1 \leq s \leq e \leq n$), representing a sublist of all chipmunks i where $s \leq i \leq e$, for which Dale and Chip would like to organise a game.

Output

Output, for each of the q sublists, the largest number of chipmunks that can be arranged into a circle such that every chipmunk is happy.

Sample Input 1

```
3
Rock
Paper
Scissors
3
1 3
1 2
2 3
```

Sample Output 1

```
3
0
0
```

Sample Input 2

```
8
Paper
Rock
Rock
Rock
Scissors
Paper
Scissors
Paper
4
3 8
4 6
1 6
2 7
```

Sample Output 2

```
6
3
3
3
```

E Eurovision

Time limit: 1s

It's that time of the year! Time again for Eurovision (and FPC)! Although, there is something rather special about this edition. In an unexpected turn of events, Delft was chosen to host Eurovision in the Aula. As you might expect, all tickets were sold out in a matter of seconds: students from all over Delft's faculties are eager to attend such a special event. Naturally, their enthusiasm is sparked by the same question: "*What's the longest time a contestant has to sing without breathing, given that they breathe optimally?*"

A song is divided into n musical segments, where the ith musical segment has pitch intensity a_i and is b_i seconds in length. Each contestant sings these musical segments consecutively, with no pause in between, but will take some deep breaths at key moments (consider the time it takes to breathe negligible). To maintain a pleasing rhythm of music, performers only breathe immediately after a *local minimum* in pitch intensity (i.e., after a musical segment i where $a_{i-1} > a_i < a_{i+1}, 1 < i < n$) and do not take more than k breaths in total.

Note:

- Eurovision contestants have an inclination towards algorithmic thinking, and therefore, they all breathe optimally: within the constraints, the longest time between two breaths is as short as possible.
- Naturally, a singer will breathe right before starting to sing and immediately after finishing, so these two breaths do **NOT** count towards the total number of breaths.

Input

The input consists of:

- One line containing two integers: n ($1 \leq n \leq 10^4$), the number of musical segments in the song, and k ($0 \leq k < n$), the maximum number of breaths that can be taken while performing the song.
- n lines containing two integers each, a_i and b_i ($1 \leq a_i, b_i \leq 10^9$), the pitch and length (in seconds) of the ith musical segment.

Output

Output the longest time (in seconds) between two breaths, for an optimal performance of the song.

Sample Input 1

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

Sample Output 1

```
3
```

Sample Input 2

```
9 2
2 1
1 1
2 1
1 1
2 1
1 1
2 1
1 1
2 1
```

Sample Output 2

```
4
```

F Fastest Thing Alive

Time limit: 2s

The villainous Doctor Pearman (though some prefer to call him Dr. Mechanic) is up to no good once again. With his newest machine, he has cracked the surface of the earth to tap into the ancient power hidden below and use it to rule the world. After seeing this, the great hero Shonic immediately departs along Dragon Road to reach the Doctor and put a stop to his plan.

A picture of Shonic the Marmot watching over the horizon
CC-BY-NC Pairi Daiza zoo, Belgium

Along the road, Shonic the Marmot comes across a spikefield of n rows, divided along m lanes. Each space in this spikefield contains either a spike trap, or is blank. Shonic is all out of his magic rings, so coming across a single spike would instantly kill him and have Doctor Pearman's plan succeed. Because of that, Shonic needs to go through a blank field in every row of the spikefield. Because he's moving very fast, Shonic can only make one horizontal move when moving to the next row of the spikefield. He can either go left and move down one row, right and move down one row, or stay in the same lane and move down one row. Additionally, he cannot leave the spikefield (he can't move to the left when in the leftmost lane). When initially entering the spikefield, Shonic can choose which field of the first row he starts in (though the field he starts in needs to be blank). A successful solution has Shonic treverse every row, starting with the first, and ending on a free field in the last. Help Shonic find a way to reach the end of the spikefield.

Input

The input consists of:

- A line with two integers n ($2 \leq n \leq 10^3$) and m ($2 \leq m \leq 10^3$), the number of rows and number of columns in the spikefield, respectively.
- n lines consisting of a single string of length m - the spikefield. An "$\star$" (asterisk) signifies that a field contains a spike trap and an "_" (underscore) signifies that the field is empty and traversable by Shonic.

Output

The output consists of either:

- A single integer l ($1 \leq l \leq n$), showing which lane Shonic starts in.
- A single string consisting of $n-1$ characters ("`L`", "`R`" and "`F`", standing for left, right and forward respectively) showing the series of directions Shonic can take to reach the end of the spikefield unharmed.

or the string "`impossible`" in case a valid path through the spikefield does not exist.

If there are multiple valid solutions, you may output any one of them.

Sample Input 1

```
5 5
_*_**
__*__
***__
**_**
**_**
```

Sample Output 1

```
3
RFLF
```

Sample Input 2

```
5 5
_*_**
__*__
***__
**_**
****_
```

Sample Output 2

```
impossible
```

G Glass Reflection

Time limit: 1s

Late in the evening, you are sitting at the Funky Punk Café and feeling a bit bored because you have not done any programming all day. You stare out of the window and see that the text on the neon signs gets reflected in a funny way: the letters are partially overlapping, because the window is made of double-paned glass. Would it be possible to automatically read the text of the sign, just by looking at this mangled reflection? With a burst of sudden inspiration, you grab your laptop and OCR[2] camera from your bag, and start programming.

A neon sign reflected in a double-paned window, spelling "bbeeerr". CC BY-SA 2.0 by Brian Ross on Flickr

Your OCR camera appears to correctly recognize the letters in the reflection when it accounts for the mirroring, except that all the letters are duplicated. Moreover, if the text on the sign contains double (or more) letters, their reflection partially overlaps, resulting in only one more letter than there would be in the original word. For example, the word "beer" is reflected as "bbeeerr", and the word "ooo" is reflected as "oooo". All that is left for you to do, is to write a program that converts these reflected words back to their original form.

Input

The input consists of:

- One line containing a string of up to 10^6 English lowercase letters (a–z), the reflected word read by your OCR camera.

Output

Output the word in its original form.

Sample Input 1	Sample Output 1
bbeeerr	beer

Sample Input 2	Sample Output 2
sskkiillleesssnneesss	skilllessness

Sample Input 3	Sample Output 3
bbaallloonnnoookkkeeeppeerr	balloonnookkeeper

[2] OCR = Optical Character Recognition

This page is intentionally left blank.

H Highways of the Future

Time limit: 6s

Mitsubishi Cement Plant - Kyushu, Japan

Midgard is a city of mana energy and center of the world's economy. With its development steered by President Shinda, chair of the Shinda electric power company, the great city has undergone an age of great prosperity. Recently, however, there have been reports that a group of bandits called Snowfall has been attacking and shutting down its mana reactors. In preparation for another attack, President Shinda has assigned you the role of chief engineer in a restructuring plan.

The city consists of n sectors, each of which has a mana reactor: an enormous facility which extracts energy from deep within the earth and transforms it into electricity powering the whole sector. Currently, there are m highways between sectors. Each highway can be used to transport electric power from sector to sector and has only one direction. The president has instructed you to build new highways between sectors, such that no matter which reactors get shut down, the entire city will still have electrical power as long as there is at least 1 reactor functioning.

Each reactor has an unlimited capacity for mana energy and can supply any number of sectors as long as it is directly or indirectly connected to them. Additionally, due to the sheer cost of building a highway, the president has instructed you to build as few highways as possible, while still satisfying his previous condition.

Input

The input consists of:

- A line containing two integers n ($1 \leq n \leq 10^5$) and m ($0 \leq m \leq 2 \cdot 10^5$), representing the number of sectors in Midgard and the number of existing highways, respectively.
- Then follow m lines containing two integers each, x and y ($1 \leq x, y \leq n$), which indicate the presence of a one-directional highway from sector x to sector y.

Output

Output the minimum number of highways that have to be added to have any reactor be able to power every sector.

Sample Input 1	Sample Output 1
3 2 1 2 1 3	2

Sample Input 2

```
8 9
1 3
5 1
2 5
2 6
3 4
3 5
5 6
8 5
3 7
```

Sample Output 2

```
3
```

Sample Input 3

```
9 9
1 3
5 1
2 5
2 6
3 4
3 5
8 5
7 3
3 7
```

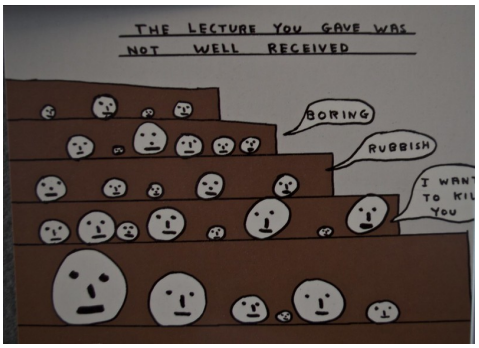
Sample Output 3

```
3
```

I Inspiring Professors

Time limit: 3s

At TU Delft, more and more courses are going back to on-campus lectures. So, naturally, it becomes more difficult to effectively schedule which lecturer can have which lecture hall. They asked you, an algorithm expert, for help on this sub-problem:



There are n lectures that happen at the same time, numbered 1 to n. In the ith course, x_i students will attend the lecture on-campus. The lectures are ordered by friendliness of the professor who gives the lecture, with the friendliest lecturer (we all know who that is) giving lecture 1.

There are m lecture halls. The lecture halls are numbered from 1 to m and the jth lecture hall has capacity c_j. The list of m lecture halls is ranked on "niceness", with the nicest lecture hall on top.

Write a program that reads in the lectures and lecture halls and makes a valid assignment of the halls to lectures. In a valid assignment, the capacity of the hall assigned to a lecture should be greater or equal than the number of students attending.

If there exist multiple valid assignments, compute the assignment which maximizes the niceness of the lecture hall of the friendliest professor. If there are still multiple assignments, maximize the niceness of the lecture hall of lecturer 2, and so on.

Input

The input consists of:

- One line containing two integers n ($1 \leq n \leq 5000$) and m ($1 \leq m \leq 5000$), the number of lectures and halls, respectively.
- The next line contains n integers x_i, with x_i ($1 \leq x_i \leq 10^9$) the number of students attending the ith lecture.
- The last line contains m integers c_j, with c_j ($1 \leq c_j \leq 10^9$) the capacity of lecture hall j.

Output

If there is a valid assignment, output a line with n numbers, with the ith being equal to the number of the lecture hall that gets assigned to lecture i.

If there is no valid assignment, output "`impossible`".

Sample Input 1

```
3 3
4 5 6
6 5 5
```

Sample Output 1

```
2 3 1
```

Sample Input 2

```
5 5
3 2 1 4 8
6 8 1 7 5
```

Sample Output 2

```
1 4 3 5 2
```

J Journey to Mastery

Time limit: 1s

Your best friend Ken has really gotten into the new Street fighter game (Ultra Ultimate Street Fighter 2 Turbo Championship Edition DX+ & Knuckles). Unfortunately, he is not very good yet, especially on defence, and really wants to attend the Fighting Pro Cup (FPC) in a few months. To get better, he has set up a training mode room where he can set a dummy computer player to fight him one on one using a specific set of actions. Reyn's goal is to consistently hit the training mode dummy without getting hit. To give him an example for how to always play optimally under these particular circumstances, you have decided to write a program that, provided with any computer player actions, always hits the dummy with an attack first.

Screen capture of the arcade version of Street Fighter III: Third Strike. Street Fighter and the image depicted are © Capcom

The fight starts with the player and the dummy n spaces away from each other. They then proceed to take turns taking actions at a regular interval of 1 unit of time (the dummy takes an action, then the player takes an action 1 unit of time later, followed by the dummy taking an action 1 unit of time after that). The completion of every action takes two units of time, *but some moves have an additional cooldown of 2 or 4 units of time after the move is finished*, in which no action can be taken, effectively "skipping" the user's next move(s). The player and the dummy both have a predetermined set of actions to choose from, detailed below:

Training Mode Dummy:

- **Walk Forward** ("`W`")
  The user slowly moves a single space toward the opponent. If this action is performed when already 1 space away from the opponent, it does nothing.
- **Jump Forward and Kick** ("`J`")
  The user makes a swift leap, moving 2 spaces towards the opponent, then performs a kick 1 unit of time after the action has been initiated. Since the user is in the air, the jump is able to avoid any "Hadouken" attacks for the duration of the action. If the opponent is 1 space away, after the leap, the kick attack will connect and end the game. If before the start of the action, players are 1 unit away, the dummy will jump over the player and be 1 space away on the other side (functionally, the side does not matter). If the players is 2 spaces away, the jump will only travel 1 space and end up 1 space from the player.
- **Grounded Kick** ("`K`")
  The user performs a kick attack that hits the opponent if they are up to 2 spaces away, ending the game if it connects. The kick comes out 1 unit of time after the attack has been selected and lasts for the next 1 unit of time. Has a cooldown of 2 units of time after being used.

- **Hadouken** ("`H`")
  A blue fireball that fires immediately, after which it begins travelling horizontally across the screen at a speed of 1 space per unit of time (by the user's next action it will have traveled 2 spaces, and by their action after that it will have traveled 4 spaces). Hits grounded opponents when it reaches the space they are currently on, ending the game on contact. If a Hadouken comes into contact with an opposing Hadouken, the fireballs cancel each other out. Has a cooldown of 2 units of time after being used.

Player:

- **Nothing** ("`N`")
  The user stands completely stationary for the duration of this action.
- **Grounded Kick** ("`K`")
  Identical to dummy Grounded Kick.
- **Hadouken** ("`H`")
  Identical to dummy Hadouken.
- **Shoryuken** ("`S`")
  The user performs an inhumanly high leap upward, and performs an uppercut that hits an opponent 1 space away and ends the game on contact. The uppercut comes out immediately as the user selects the attack. *For the first unit of time after this action is selected, the user is completely invulnerable to any attacks* (so for instance, this move can be used in reaction to a "jump forward and kick" action, and if players are 1 space away by the time the Shoryuken action is taken, the human player wins, as they land the uppercut and are invulnerable to the kick). Has a cooldown of 4 units of time after being used.

Interaction

This is an interactive problem. Your submission will be run against an *interactor*, which reads the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

The interactor first sends one line containing one integer n ($1 \leq n \leq 100$), the initial number of spaces between the players. After that begins the following process:

- The interactor either sends one letter, corresponding to one of the 4 dummy actions, or a "`-`" to signify a cooldown.
- Then, your program should either output a letter, corresponding to one of the 4 player actions, or a "`-`" to signify a cooldown.

This process continues until either player manages to land a successful attack within its range on the other player. A test is considered passed if, at the end of the process, the only successful attack to end the game is made by the player. A correct solution is indicated by the interactor by inputting the symbol "`V`" on the last line. Tests with no correct solution possible will not be used.

To prevent infinite loops, the dummy makes a move forward ("`W`" or "`J`") at least once every five moves. Additionally, a limit of 1000 moves has been placed on the number of actions the player can make, after which the solution will be considered incorrect.

Make sure you flush the buffer after each write.

A testing tool is provided to help you develop your solution. Instructions on how to use it can be found in the comments at the top of that file.

Read	**Sample Interaction 1**	**Write**
3 W		
	K	
V		

Read	**Sample Interaction 2**	**Write**
3 K		
	H	
–		
	–	
V		

Read	**Sample Interaction 3**	**Write**
6 H		
	H	
–		
	–	
W		
	H	
H		
	–	
–		
	H	
J		
	–	
V		