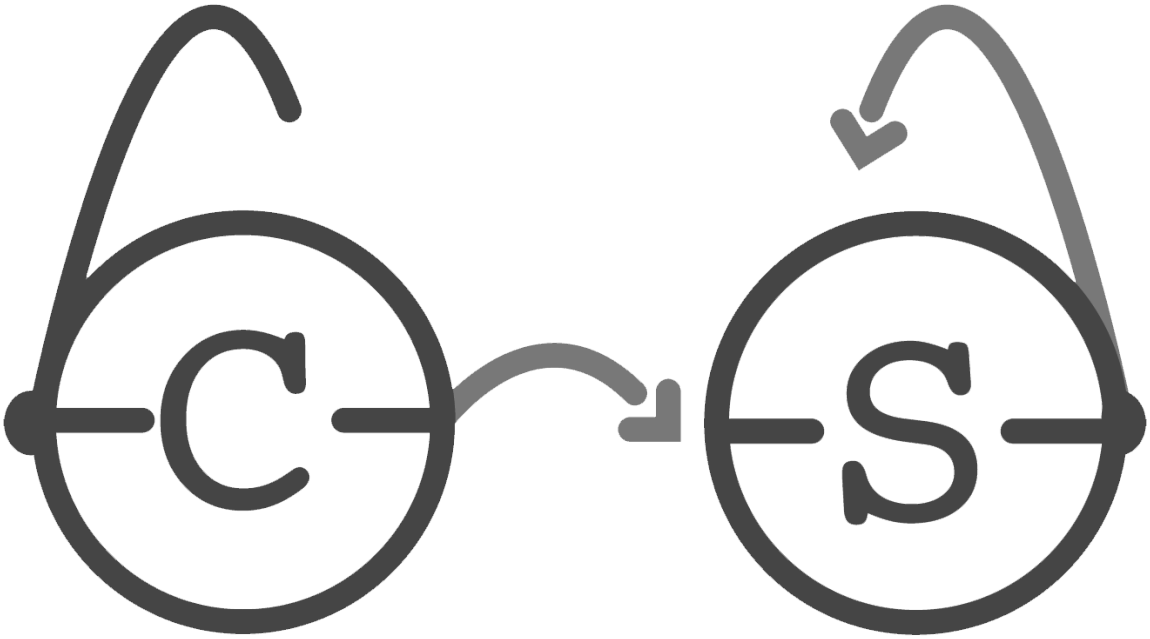
Contest Problems
Philadelphia Classic, Fall 2021
Hosted by the Computer Science Society
University of Pennsylvania



Rules and Information

This document includes 12 problems. Novice teams do problems 1-8; standard teams do problems 5-12.

Any team which submits a correct solution for any of problems 1-4 will be assumed to be a novice team. **If you are not a novice team, please skip problems 1-4.**

Problems 1-4 are easier than problems 5-8, which are easier than problems 9-12. These problems are correspondingly labeled “Novice”, “Intermediate”, and “Advanced.” Order does not otherwise relate to difficulty.

You may use the Internet only for submitting your solutions, reading Javadocs or Python documentation, and referring to any documents we link you to. You **may not** use the Internet for things like StackOverflow, Google, or outside communication.

As you may know, you can choose to solve any given problem in either **Java or Python**. We have provided stub files for all questions in both languages that take care of the input parsing for you. **Do not modify any of the parsing or output code**. Just fill out the stub methods in each file and submit it with exactly the same name.

Do not modify any of the given methods or method headers in our stub files! They are all specified in the desired format. You may add class fields, helper methods, etc as you like, but modifying given parts will cause your code to fail in our testers.

There is no penalty for incorrect submissions. You will receive 1 point per problem solved. A team’s number of incorrect submissions will be used only as a tiebreaker.

Some problems use Java’s “long” type; if you are unfamiliar with them, they’re like an “int”, but with a (much) bigger upper bound, and you have to add “L” to the end of an explicit value assignment:

```
long myLong = 1000000000000L;
```

Otherwise, the “long” type functions just like the “int” type.

1. Porco's Mysterious Call…

`MysteriousCall.py, MysteriousCall.java`

Porco the Porcupine hears something on his walkie-talkie! It sounds like a bunch of letters. At the end of the message, the walkie talkie announces that the last k letters are reversed. Porco is determined to decipher the message from his walkie talkie, whoever it is from. He has all the letters but now needs to reverse the last k letters to figure out the message. Help Porco come up with an algorithm so that he can figure out the coded word.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`reversedMessage(message, k)`

- **`message`**: a string that represents a message that needs to be deciphered.
- **k**: an int that represents the number of last letters that needs to be reversed.

Your function should return a string, deciphered message.

Constraints

$1 \leq |message| \leq 10000$
$0 \leq k \leq |message|$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **message**.
The second line contains **k**.

Output

For each test case, print a string representing the deciphered message.

(cont. on next page)

1. Porco’s Mysterious Call...

Sample Input	Parsed Input	Sample Output	Explanation
`1` `heoll 3`	`reversedMessage(“heoll”, 3)`	`hello`	Reverse the last 3 letters from “oll” to “llo.”
`1` `buzz 0`	`reversedMessage(“buzz”, 0)`	`buzz`	The last 0 letters are reversed, so the word is fine the way it is.
`1` `cissalcp 8`	`reversedMessage(“cissalcp”, 8)`	`pclassic`	The length of the word is 8, and k is 8, so the whole word is reversed.

2. Secret Code

`SecretCode.py, SecretCode.java`

Polly the Penguin is writing Panda Panda secret messages. These are top secret messages so she writes them with a secret code. In Polly's code each letter corresponds with a number: a-1, b-2, …, z-26. Polly chooses some integer k to add to the number corresponding to each letter in her message. The coded message contains the letters whose numbers correspond to the letters in the original message plus k.

For example, if k=2 then a → c because 1 + 2 = 3 and 3 corresponds with c. Also note that there is wraparound when the number is greater than 26: if k=2, then z → b, because 26 + 2 = 28 and 28 % 26 = 2 which corresponds to b. There is also wraparound for numbers less than 1. For example, k = -2 then a → y. Also, Polly's message can contain punctuation and spaces. These characters are unaffected by Polly's code. Polly tells Phil what k is. Help Phil decode Polly's message.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`decode(s, k)`

- **s**: a string containing symbols, spaces and lowercase characters that represents the coded message.
- **k**: an int that represents the number added to each alphabetical character in **s**.

Note: only shift the alphabetical characters.

Your function should return the decoded string.

Constraints

$0 \leq |s| \leq 100$
$-1000 \leq k \leq 1000$

(cont. on next page)

2. Secret Code

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **s**
The second line contains **k**

Output

For each test case, print a string representing the deciphered message.

Sample Input	Parsed Input	Sample Output	Explanation
1 rovvy 10	decode(“rovvy”, 10)	hello	r = 18 o = 15 v = 22 y = 25 r - 10 = 8 = h o - 10 = 5 = e v - 10 = 12 = l y - 10 = 15 = o
1 ibgg 7	decode(“ibgg”, 7)	buzz	i = 9 b = 2 g = 7 i - 7 = 2 = b b - 7 = -5 % 26 = 21 = u g - 7 = 0 = 26 = z *Since a = 1, we must wrap around to the end of the alphabet, so 0 can also be considered as z

1 zab -1	decode(“zab”, -1)	abc	z = 26 a = 1 b = 2 z - (-1) = 26 + 1 = 27 % 26 = 1 = a a - (-1) = 1 + 1 = 2 = b b - (-1) = 2 + 1 = 3 = c

3. Dangerous Dinosaur Dodgeball

`DinoDodgeball.py, DinoDodgeball.java`

Ptolemy the Pterodactyl is playing a game of dinosaur dodgeball with his pterodactyl pals. In this version of dodgeball, there are two types of balls: fire balls and ice balls. Additionally, the game is very quick in that every pterodactyl throws some amount of balls (which may be zero) at other pterodactyls all at the same time. Unfortunately, these pterodactyls aren't very good at dodging, so each pterodactyl is hit by all balls thrown at them. Each ball is given a number which corresponds to its temperature. Ice balls are represented by negative numbers and fire balls are represented by positive numbers. Help determine the number of pterodactyls such that the sum of the temperatures of balls that hit them is zero.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`netTemperature(n, temperatures)`

- **n**: an int representing the number of pterodactyls.
- **temperatures**: A list of dodgeballs thrown where each element is of the form [x, y, t]
  x represents the number of the pterodactyl who is throwing the ball
  y represents the number of the pterodactyl who gets hit by the ball
  t represents the temperature of the ball that is thrown.

Your function should return a single integer which represents the number of pterodactyls who were hit by balls whose temperatures sum to zero.

Constraints

$1 \leq n \leq 10^6$

$1 \leq length(temperatures) \leq 10^6$

$0 \leq x, y \leq n - 1$

$-1000 \leq t \leq 1000$

(cont. on next page)

3. Dangerous Dinosaur Dodgeball

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **n** and **k**.
For the following **k** lines:
Each line contains **x, y,** and **t**

Integer **n** representing the number of pterodactyls
Integer **k** representing the amount of balls thrown
x represents the number of the pterodactyl who is throwing the ball
y represents the number of the pterodactyl who gets hit by the ball
t represents the temperature of the ball that is thrown.

Output

For each test case, print the number of pterodactyls that have a sum of 0 temperature.

Sample Input	Parsed Input	Sample Output	Explanation

`1`
`3 4`
`0 1 5`
`0 1 -5`
`1 0 3`
`1 0 -3`
	`netTemperature(3, 4, [[0, 1, 5], [0, 1, -5], [1, 0, 3], [1, 0, -3]])`	`3`	Pterodactyl 0 is hit with balls of temperatures 5 and -5, so the sum of temperatures is 5 + (-5) = 0. Pterodactyl 1 is hit with balls of temperatures 3 and -3, so the sum of temperatures is 3 + (-3) = 0. Pterodactyl 2 is not hit by any balls, so the sum of temperatures of balls that hit it is also 0.

`1`
`4 8`
`3 0 2`
`1 2 4`
`0 1 6`
`2 3 2`
	`netTemperature(4, 8, [[3, 0, 2], [1, 2, 4], [0, 1, 6], [2, 3, 2], [1, 3, -4], [0, 3, 2], [3, 2, -5], [0, 2, 1]])`	`2`	Pterodactyl 0 → 2
			Pterodactyl 1 → 6
			Pterodactyl 2 → 4 + (-5) + 1 = 0
			Pterodactyl 3 → 2 + (-4) + 2 = 0
			Pterodactyls 2 and 3 have temperatures that sum to zero.

1 3 -4 0 3 2 3 2 -5 0 2 1			
1 4 4 3 1 4 1 2 1 2 0 -100 0 3 0	netTemperature(4, 4, [[3, 1, 4], [1, 2, 1], [2, 0, -100], [0, 3, 0]])	1	Pterodactyl 0 → -100 Pterodactyl 1 → 4 Pterodactyl 2 → 1 Pterodactyl 3 → 0 Only pterodactyl 3 has temperatures that sum to zero.
1 2 2 0 1 17 1 0 -12	netTemperature(2, 2, [[0, 1, 17], [1, 0, -12]])	0	Pterodactyl 0 → -12 Pterodactyl 1 → 17 None of the pterodactyls have temperatures that sum to zero.

4. Love Letter

`LoveLetter.py, LoveLetter.java`

Paul the Polar Bear wants to write a lovely note to his friend, Piper the Parrot. His only way to get a message from his house in the Arctic to Piper's abode in a faraway tropical forest is to send strings of loose Scrabble tiles that he picked up from local litterers.

Since Paul graduated at the top of his class at the University of Polar Bears, he is quite eloquent, and can easily spell out his message using the tiles. However, due to his intelligence and tendency to overachieve, he wants his message to have double meanings, so that he can fully express all his heart to Piper.

He wants to write a letter on the back of each tile, so that they can spell out new words and a new message. This would be a trivial task for Paul on its own, but he is very picky and does not want the same Scrabble tile letter to have different letters written on their backs. That is, if there was a Scrabble tile A with the letter B on the back, every Scrabble tile A must also have B on their backs. The opposite must also be true: for every Scrabble tile with the letter B written on its back, its original Scrabble letter must be A. Help Paul determine if there exists a way for him to write both of his messages!

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`checkMessages(s, t)`

- **s**: a string containing some number of lowercase letters, representing Paul's original message
- **t**: a string of lowercase letters, of the same length as **s**, representing Paul's desired message

Your function should return true if the valid mapping exists and false otherwise.

Constraints

$0 \leq |s|, |t| \leq 10^6$

(cont. on next page)

4. Love Letter

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **s**
The second line contains **t**

Output

For each test case, output “yes” or “no” depending on your answer.

Sample Input	Parsed Input	Sample Output	Explanation
1 abc def	checkMessages (abc, def)	yes	a ↔ d, b ↔ e, c ↔ f
1 abc dde	checkMessages (abc, dde)	no	Characters ‘a’ and ‘b’ cannot both be matched to character ‘d’.
1 abc abc	checkMessages (abc, abc)	yes	Characters can match to themselves.
1 aabbcc hkccbb	checkMessages (aabbcc, hkccbb)	no	Character ‘a’ cannot match both characters ‘h’ and ‘k’.

5. Pompous Peacocks

`PompousPeacocks.py, PompousPeacocks.java`

Percy the Peacock is going to a festival this spring. He knows there will be a beautiful peacock, Pearl, who will be attending. Pearl has always attracted many other peacocks to the festival, all who want to show off their feathers to Pearl.

Percy is a businessman, and knows this is a great opportunity to run a betting pool on which of these N peacocks will win Pearl's heart. He has been doing some statistical research and knows that Pearl always chooses peacocks with the most vibrant colors and most confident strut. Thus, he has quantified the vibrancy and confidence of every peacock who will be vying for Pearl's attention and put this data into arrays for analysis. For any two peacocks A and B, Peacock A beats Peacock B if A's vibrance is at least as high as B's vibrance, A's confidence is at least as high as B's confidence, and A and B both don't have the same confidence and vibrance. A peacock is unbeatable if there is no other peacock that can beat them. Help Percy find the number of the unbeatable peacocks so that he can run his betting pool for maximum profit.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`unbeatable(peacocks)`

- **peacocks**: 2D array where each row i has two integers v and c representing i-th peacock's vibrance and confidence values

Your function should return an integer representing the number of unbeatable peacocks.

Constraints

$1 \leq |peacocks| \leq 10^4$

$0 \leq v, c \leq 10^4$

(cont. on next page)

5. Pompous Peacocks

Input

The first line represents the number of test cases **T**.
For each of the **T** cases:
There are a variable number of lines, representing each peacock, followed by an empty line. Before the empty line, each line contains **v** and **c** representing the vibrance and confidence of the peacock.

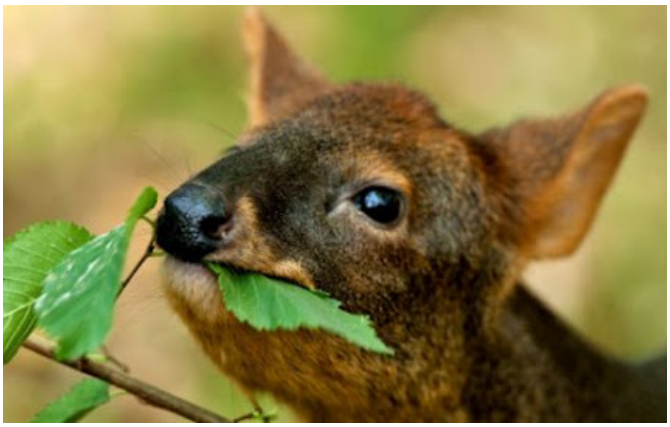
Output

For each test case, output the number of unbeatable peacocks.

Sample Input	Parsed Input	Sample Output	Explanation
1 0 1 1 2	unbeatable([[0, 1], [1,2]])	1	The second peacock beats the first because 1 > 0 and 2 > 1.
1 1 2 1 4	unbeatable([[1,2], [1,4]])	1	The second peacock beats the first because 4 > 2 and 1>=1.
1 1 3 0 5	unbeatable([[1,3], [0,5]])	2	The first peacock cannot beat the second because the second has a greater confidence. The second peacock cannot beat the first because the first has a greater vibrancy.
1 3 4 3 4	unbeatable([[3,4], [3,4]])	2	Both peacocks cannot beat each other because they have equal stats.

6. Picky Puupuu's Preferences

PickyPuupuu.py, PickyPuupuu.java



Puupuu the Pudu is a very hungry pudu. Today, she is in the mood to eat some berries. While roaming around searching for food, luck strikes her, and she finds a rectangular field full of berries! However, there are rivers that run vertically and horizontally which partition the field into m rows and n columns of land patches. Each of these m*n land patches contain an integer amount of berries between 1 and 9 inclusive. Puupuu only wishes to eat berries from a subset of these land patches. In fact, the subset of land patches must form a 3x3 square in the field, and the amount of berries within each patch of the 3x3 square must form a magic square. A magic square is a 3x3 square of numbers such that the sum of the numbers in each row, column, and diagonal is identical. Note that the values within the square do not need to be distinct. Help Puupuu discover whether or not there exists a set of land patches where she can eat the berries from.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`magicSquare(grid)`

- **`grid:`** a 2D array of arrays of integers between 0 and 9

Your function should return true if there exists a magic square and false otherwise

Constraints

$0 < m, n \leq 1000$

Input

The first line represents the number of test cases **T**.
For each of the **T** cases:
There are a variable number of lines, representing each row of the grid, followed by an empty line. It is guaranteed that each row has the same number of elements, where each element represents the number of berries.

Output

For each test case, output “true” or “false” depending on your answer.

(cont. on next page)

6. Picky Puupuu's Preferences

Sample Input	Parsed Input	Sample Output	Explanation
1 2 1 2 7 6 1 1 9 5 1 1 1 4 3 8 2 3 1 4 1	magicSquare({{2, 1, 2, 7, 6}, {1, 1, 9, 5, 1}, {1, 1, 4, 3, 8}, {2, 3, 1, 4, 1}})	true	The magic square is composed of the values grid[i][j] where 0 <= i <= 2 and 2 <= j <= 4. Numerically, the square is: 2 7 6 9 5 1 4 3 8
1 2 1 7 6 1 9 5 1 1 4 3 8	magicSquare({{2, 1, 7, 6}, {1, 9, 5, 1}, {1, 4, 3, 8}})	false	There are no magic squares in the grid.
1 8 1 6 3 5 7 4 9 2	magicSquare({8, 1, 6}, {3, 5, 7}, {4, 9, 2}}	true	The magic square is composed of the values grid[i][j] where 0 <= i <= 2 and 0 <= j <= 2. Numerically, the square is: 8 1 6 3 5 7 4 9 2
1 7 7 7 7 7 7 7 7 7	magicSquare({7, 7, 7}, {7, 7, 7}, {7, 7, 7}}	true	The magic square is composed of the values grid[i][j] where 0 <= i <= 2 and 0 <= j <= 2. Numerically, the square is: 7 7 7 7 7 7 7 7 7

7. Cactus Farm

`CactusFarm.py, CactusFarm.java`

Porco the Porcupine has a farm and is trying to grow a line of cactuses to the right of his farm. The line has plots of soil in a line and a cactus seed in each. He also has a machine that holds a line of n hoses such that hoses can hover above the i-th to (i + n - 1)-th (inclusive) plot for some i. If he activates the hoses, special water drops below which instantly grow the cactus seed. Unfortunately, some of the hoses are defective and so their water does not instantly grow the cactus. It is guaranteed that the last hose is not defective.

He can also shift the location of the machine, in other words, he can change i. Giving grown cacti extra water does nothing. Adding extra normal water to cactus seeds does nothing.

He first activates the machine on plots 0 to n - 1, which results in an initialPattern, represented as a string of 0s and 1s, where 0 is a seed and 1 is a grown cactus. Porco wants his cactus farm to match finalPattern, which is also represented as a string of 0s and 1s, and the last plot is guaranteed to be a full grown cactus. Given that he can shift and activate the machine any number of times, can you help Porco find if it is possible for his current cactus farm to match finalPattern?

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`matchable(initialPattern, finalPattern)`

- **`initialPattern`**: a string of 0s and 1s that represents the initial cactus farm after the first activation of the machine. This string ends in 1.
- **`finalPattern`**: a string of 0s and 1s that represents the final cactus farm to match. This string ends in a 1.

Your function should return true or false if it is possible to transform the initial cactus farm to the final cactus farm.

Constraints

$1 \leq |finalPattern| \leq 2000$

$1 \leq |initialPattern| \leq |finalPattern|$

Input

The first line represents the number of test cases **T**.
For the every **T** lines:
The first line contains **initialPattern**.
The second line contains **finalPattern**.

Output

For each test case, output "1" if it is possible to transform the cactus farm or "0" if it is not.

Sample Input	Parsed Input	Sample Output	Explanation
1 1 11	matchable("1", "11")	1	The machine is first activated at plot 0, so the initial pattern is 1. He shifts the machine to plot1 and then activates again so the resulting pattern is 11. Activate: 1 Move machine to plot 1 and activate: 11
1 1 101	matchable("1", "101")	1	The machine is first activated at plot 0, so the initial pattern is 1. He shifts the machine to plot 2 and then activates again so the resulting pattern is 101. Activate: 1 Move machine to plot 2 and activate: 101
1 101 111101	matchable("101", "111101")	1	The machine is first activated at plots 0 through 2, so the initial pattern is 101. He shifts to plot 3 and then activates again so the pattern is then 101101. He then shifts plot 1 and then activates to get the

			pattern 111101. Activate: 101 Move machine to plot 3 and activate: 101101 Move machine to plot 1 and activate: 111101
1 101 111	matchable(“101”, “111”)	0	This is not possible because it is not possible to turn the second plot into a 1.

8. Peaceful Poles

`PeacefulPoles.py, PeacefulPoles.java`

Polly the Penguin has formally declared the South Pole as the superior pole. Paul the Polar Bear hears this and declares a war with the penguins of the South Pole. After five minutes of intense battle, the polar bears and penguins have grown weary of the fighting. After signing a peace treaty, Polly and Paul decide to play a cooperative game to celebrate, using the letters in the peace treaty.

Polly receives x cards and Paul receives y cards, where each card contains a random lowercase letter from the treaty, and the cards are ordered in the order they appear in the treaty. In each round, each animal can choose to put down the card at the top of their deck or not. If both animals put down the same letter, they gain a point. If they put down different letters, they lose a point. If only one animal puts down a letter, they lose two points. If neither animal puts down a letter, no points are gained or lost. The game ends when both animals finish putting down all of their cards. Given the letters Polly and Paul receive, what is the maximum number of points they can achieve?

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`maxPoints(s, t)`

- **`s:`** A string of the letters on Polly's **x** cards in order from top to bottom
- **`t:`** A string of the letters on Paul's **y** cards in order from top to bottom

Your function should return the maximum number of points Polly and Paul can achieve.

Constraints

$0 \leq x, y \leq 10^4$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **s**
The second line contains **t**

Output

Output an integer representing the maximum number of points Polly and Paul can achieve.

(cont. on next page)

8. Peaceful Poles

Sample Input	Parsed Input	Sample Output	Explanation
1 us us	maxPoints(“us”, “us”)	2	us us Both Polly and Paul can put down “u” in the first round and “s” in the second round, gaining 1 point in each. 2(1) - 0(1) - 0(2) = 2
1 ba bc	maxPoints(“ba”, “bc”)	0	ba bc In the best possible game, Polly and Paul both put down “b” in the first round, gaining 1 point, but they put down different letters in the second round, losing 1 point. 1(1) - 1(1) - 0(2) = 0
1 abcdef abecdf	maxPoints(“abcdef”, “abecdf”)	1	ab_cdef abecd_f Polly and Paul put down the same letter five times, gaining 5 points, but in two rounds, only one of them put down a card, losing a total of 4 points. 5(1) - 0(1) - 2(2) = 1

9. Polar Bear Testing

`PolarTesting.py, PolarTesting.java`

The local zoo is currently experiencing problems with its polar bears: out of a certain number of bears, one has a disease that will eventually make it shrink to the size of an atom. We wouldn't want that!

There are some number of polar bears, and exactly one of them is sick. To test whether a polar bear is sick, a device can be used to take samples from a subset of polar bears. If one of the bears selected was sick, the device will turn bright red. However, the results are not immediate and take several minutes to load.

Your testing process is as follows: you take a fixed amount of devices, and each device touches a subset of bears. You wait until the devices are done, then see which devices (if any) turn red, then take the remaining devices to repeat the processes. However, you only have limited time to complete this process before the sick polar bear shrinks! Given these constraints, determine the minimum number of devices you need to guarantee you have found the sick polar bear.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`testing(bears, testtime, totaltime)`

- **`bears`**: an integer that represents the number of polar bears.
- **`testtime`**: an integer that represents how long a device takes to react.
- **`totaltime`**: an integer that represents how much time you have to find the sick polar bear before it shrinks.

Your function should return an integer that is the smallest amount of devices needed to guarantee that the sick polar bear can be found.

Constraints

$1 \leq bears, testtime, totaltime \leq 10^6$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **bears, testtime, totaltime.**

Output

Output an integer representing the least number of devices needed to guarantee that the sick polar bear can be found.

Sample Input	Parsed Input	Sample Output	Explanation
1 4 4 4	`testing(4, 4, 4)`	2	The first device can test bears 1,2, while the second device can test bears 1,3. If the first and second device both turn red, then we know bear 1 is sick. If only one of the devices turns red, either bear 2 or bear 3 are sick (corresponding to first and second device respectively). If none of the devices turn red, bear 1, 2, and 3 are all not sick so it must be bear 4 that is sick.
1 4 15 45	`testing(4, 15, 45)`	1	You can use 1 device 3 times in a row, each time on a different bear (bears 1, 2, 3), to figure out if any of those bears specifically are sick. If none of the devices turn red, bear 4 must be the one that is sick.

10. Panda Land

`PandaLand.py, PandaLand.java`

Panda Panda is traversing a n x m bamboo forest where he starts at the top left corner (1, 1) and wants to walk to the bottom right corner (n, m). He can only move down and right through the forest.

As the zookeeper, you get to decide how much bamboo is in each cell in the grid. However, each cell has a distinct number of bamboo in the range 0 through n * m - 1 inclusive.

Recently, Panda Panda has invented a device that allows him to collect all of the bamboo in every cell in row r and every cell in column c if he is standing in the cell (r, c). Furthermore, the bamboo in a cell regenerates as soon as Panda Panda finishes collecting from it, so the bamboo in a cell can be collected multiple times. Because you want to prevent Panda Panda from having a monopoly on the bamboo, you want to minimize the amount of bamboo Panda Panda collects during his walk.

Given Panda's path through the forest as a sequence of moves where "D" represents a move down and "R" represents a move right, find the minimum amount of bamboo Panda can collect mod 10^9 + 7.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`pandaLand(n, m, path):`

- **`n`**: the number of rows in the grid
- **`m`**: the number of columns in the grid
- **`path`**: A string of length n + m - 2 describing the path from (1, 1) to (n, m) where “D” represents a move down and “R” represents a move right

Your function should return an integer mod 10^9 + 7 denoting the minimum amount of bamboo Panda Panda can collect.

Constraints

$2 \leq n, m \leq 10^5$
$|path| = n + m - 2$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **n, m** which represent the number of rows and columns of the grid respectively.
The second line contains the string **path** which represents Panda's path from the top left corner to the bottom right corner.

Output

Output an integer representing the minimum amount of bamboo Panda Panda can collect.

Sample Input	Parsed Input	Sample Output	Explanation
1 2 2 RD	`pandaLand(2, 2, "RD")`	12	10 23 You start at the cell with '1', which gives you 3 bamboo. Then you move right to the cell with '0', which gives you 4 bamboo. Then you move down to the cell with '3', which gives you 5. 3 + 4 + 5 = 12
1 2 3 RDR	`pandaLand(2, 3, "RDR")`	36	502 314 10 + 8 + 8 + 10 = 36

11. Pretentious Peacocks

`PretentiousPeacocks.py, PretentiousPeacocks.java`

Percy the Peacock found the unbeatable peacocks that will be at the peacock festival this spring, and is now feeling accomplished. However, while basking in his (projected) increased profits, he realized that he can further optimize his betting business by analyzing the relationships between the peacock competitors. Specifically, every peacock ***p*** who will be competing for Pearl the Peacock's heart at the festival has identified some singular other peacock, ***q***, to be their main rival-- their archnemesis-- this season.

Each peacock ***p*** will have exactly one archnemesis ***q*** and ***q***'s archnemesis need not be ***p***. Peacock ***p1*** and Peacock ***pk*** are rivals if there is a sequence of archnemeses ***p1, p2, ..., pk*** (i.e. ***p2*** is the archnemesis of ***p1, p3*** is the archnemesis of ***p2*** and so on). Unlike in Question 5, no two peacocks have both the same vibrancy and confidence as another.

Percy wants to use this information to find out who will be the finalists in this tense competition and exploit the data to run more betting pools. He defines a peacock as "strong" if none of their rivals can beat them, where "beating" another peacock is still defined as being more vibrant and confident than them (see Question 5 for full definition). Help Percy find the number of "strong" peacocks that will be competing, so that he may become the richest peacock in town from his betting ring.

(cont. on next page)

11. Pretentious Peacocks

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`unbeatable(n, m)`

- **n**: A 2D array where the i^{th} row has 2 elements **v**,**c** representing the vibrancy and and confidence of the i^{th} peacock i
- **m**: An array where the i^{th} element is an int representing the archnemesis of the i^{th} peacock
  Note: i is one indexed so the first element in n is the first peacock

Your function should return the number of unbeatable peacocks.

Constraints

$1 \leq m.length, n.length \leq 10000$
$0 \leq v, c \leq 10000$

Input

The first line represents the number of test cases **T**.
For every test case:
The first line contains a number **i** representing the number of peacocks (aka the length of **m** and **n)**
The next **i** lines contain **v**, **c** representing the vibrancy and confidence of the peacocks
The next **i** lines contain **a** representing the archnemesis of the peacocks

Output

Output an integer representing the number of unbeatable peacocks.

(cont. on next page)

11. Pretentious Peacocks

Sample Input	Parsed Input	Sample Output	Explanation
1 3 1 2 3 4 5 6 2 3 1	unbeatable(3, [[1, 2], [3, 4], [5, 6]], [2, 3, 1])	1	Peacocks 1, 2, and 3 all consider each other rivals (via the rule that rivals of their ultimate rival are also rivals). Only peacock 3 is strong because none of its other rivals have a vibrancy greater than or equal to their own vibrancy, and none of its other rivals have a confidence greater than or equal to their own confidence.
1 4 1 1 3 3 4 2 2 2 2 3 1 2	unbeatable(4, [[1, 1], [3, 3], [4, 2], [2, 2]], [2, 3, 1, 2])	2	Peacocks 1, 2, and 3 all consider each other rivals. Peacock 4 considers peacocks 1, 2, and 3 its rivals. Only peacocks 2 and 3 are strong because no other peacocks they consider rivals are greater or equal to them in terms of both vibrancy and confidence.
1 7 4 5 2 9 2 0 8 4 1 7 2 3 6 9 4 1 1 2 2	unbeatable(7, [[4, 5], [2, 9], [2, 0], [8, 4], [1, 7], [2, 3], [6, 9]], [4, 1, 1, 2, 2, 3])	4	Left as an exercise to the reader.

3 3			

12. New Park on Google Maps

NewPark.py, NewPark.java

Pedro Parrot is designing an amusement park with many thrilling rides! Because Pedro is a bird, he somehow designed all these rides to float in the air. Google wants to mark this park on Google Maps, so Pedro designed a 2D layout of the park, where each ride is enclosed in a **rectangular** area. Because Google Maps can only look at the park from top down, Google Maps may actually underestimate the total area of the park because some rides overlap. Can you figure out how many units of area the park looks like, based on what Google Maps sees?

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`area(rectangles)`

- **`rectangles`**: a list of rectangles representing the rides of the form [x1, y1, x2, y2], which are the coordinates of the corners of the rectangle.

Your function should return the units of area mod 10^9 + 7.

Constraints

$1 \leq x1, y1, x2, y2 \leq 10000$
$x1 \leq x2$
$y1 \leq y2$
$0 \leq |rectangles| \leq 100000$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **the number of rectangles, k**.
For the next **k** lines:
Each line contains four integers **x1, y1, x2, y2** -- the coordinates of the corners of each rectangle.

Output

For each test case, print an integer representing the total area covered by the amusement park rides mod 10^9 + 7.

Sample Input	Parsed Input	Sample Output	Explanation
1 1 0 0 4 4	rectangleArea([[0, 0, 4, 4]])	16	There is only one ride, and its corner coordinates are (0,0) and (4,4), so the total area is 4*4 = 16
1 2 0 0 4 4 5 5 6 6	rectangleArea([[0, 0, 4, 4], [5, 5, 6, 6]])	17	There are two rides. The first ride has corner coordinates at (0,0) and (4,4), so its area is 4*4 = 16. The second rectangle hs corner coordinates at (5,5) and (6,6), so its area is (6-5)*(6-5) = 1. Since the rectangles do not overlap, the total area is 16 + 1 = 17
1 2 0 0 4 4 3 3 6 6	rectangleArea([[0, 0, 4, 4], [3, 3, 6, 6]])	24	There are two rides. The first ride has corner coordinates at (0,0) and (4,4), so its area is 4*4 = 16. The second rectangle hs corner coordinates at (3,3) and (6,6), so its area is (6-3)*(6-3) = 9. The rectangles overlap in the square with corner coordinates (3,3) and (4,4), so this overlap area is 1. Thus, the total area is 16 + 9 - 1 = 24
1 3 0 0 4 4 1 3 5 5 1 1 3 8	rectangleArea([0, 0, 4, 4], [1, 3, 5, 5], [1, 1, 3, 8]])	27	Three rectangles with much overlap. The areas of the individual rectangles are 16, 8, and 14. The total amount of area that overlaps (including double counting and triple

			counting) is 11. So the total overall area is 16 + 8 + 14 - 11 = 27