# Problem 1: Campus Tag

## The Problem

Jason decided to play campus tag at Code Fast University. In campus tag, each player is on a team. Before the game begins, each player is given a target to tag before the game ends. The university is having trouble with one problem: They have two teams this year, each with a certain number of players. Can they give everyone a target that is on the opposite team?

For example, let's say that Bob is on team red and Jason is on team green. If Bob gets Jason as a target, and Jason gets Bob, then the order is valid. It is However, if Jason got someone on the green team or Bob got someone on the red team, then the order is invalid.

Given the number of people on two teams, can you determine if it is possible to give everyone a target that is on their opposing team?

## Input

The first line of input contains one integer, **T**, the number of test cases.

Each of the next **T** lines contains two integers **A** and **B**, where **A** is the number of players on the green team, and **B** is the number of players on the red team.

## Output

There should be one line of output for each test case. Print “Valid” if it is possible to create a correct configuration. Otherwise, print “Invalid”.

## Constraints

* 1 ≤ **T** ≤ 105
* 0 ≤ **A**, **B** ≤ 109

## Sample Input

2

1 1

2 3

## Sample Output

Valid

Invalid

# Problem 2: Oh My Word!

## The Problem

Nathan was using Microsoft Word one day, and he noticed an interesting feature in autocorrect. If he typed a word, but he accidentally swapped two adjacent letters, then Word would correct the error. He is wondering if you could write him a program to show him how this feature works.

You are given a dictionary of legal words (which may or may not be English words). Then you are given some words that may have two adjacent characters swapped. Your goal is to determine if it is possible to swap any two *adjacent* characters to create a word in the dictionary. How many possible words in the dictionary can you create from any single character swap?

## Input

The first line contains one integer, **N,** the number of words in the dictionary.

Each of the next **N** lines contains one word each, corresponding to a word in the dictionary.

The next line contains one integer, **Q**, the number of queries. Each query is a word to test against the dictionary.

Each of the next **Q** lines contains one word for each query.

## Output

There should be one line of output for each query. Each line should be printed as follows:

* If the word is already in the dictionary, print “okay”.
* If there is only one way to perform a swap that makes a dictionary word, print that dictionary word.
* If there are many possible dictionary words, print “**x** possibilities”, where **x** is the number of words.
* If the word cannot be found in the dictionary after any swap, print “unknown”.

## Constraints

* 1 ≤ **N**, **Q** ≤ 1000
* All dictionary words are guaranteed to be unique

## Sample Input

4

hello

world

jbu

buj

4

hlelo

world

what

bju

## Sample Output

hello

okay

unknown

2 possibilities

# Problem 3: Bowling Blitz

## The Problem

Arianna is going bowling with some friends. The computer malfunctioned while they were bowling, so she had to keep track of everyone’s moves. Can you help her find out who won the game?

Bowling scoring is calculated as follows:

* A regular play consists of two digits (or ‘-‘). A ‘-‘ means that no pins were knocked down during that move. A digit **p** means that **p** pins were knocked down. The score for a regular play is the sum of the two digits. For example, the score of “62” would be 8 (6 + 2), and “-5” would be 5 (0 + 5).
* A spare play ends with a ‘/’. The score is calculated as 10 points + the points scored on the next move. For example, the consecutive plays “5/ 61” would score 16 points: 10 for a spare in the first play + 6 on the next move.
* A strike is an ‘X’ by itself. A strike is calculated as 10 points + the score of the next two moves. For example, “X X 53” would be 25: 10 points for the first strike + 10 points for the next move (‘X’) + 5 points for the next move (‘5’).
* Any moves past the end of the game are counted as 0 points. For example, ending the game with a strike is only 10 points for that strike.
* **NOTE**: For simplicity, the last play of the game (10th) is scored in the same way as any other play.

## Input

The first line contains one integer, **T**, the number of test cases.

The first line of each test case contains one integer, **N**, the number of players.

Each of the **N** player descriptions consists of two lines. The first line contains the player’s name as a string. The second line contains a description of the player’s game. Each game has 10 plays, described above.

## Output

For each test case output a line as follows:

* If the game has one winner, output “**N** wins with **P** points!”, where **N** is the player’s name, and **P** is how many points they scored.
* If the game has a tie, output “**N** tie with **P** points!”, where **N** is a sorted, comma-separated list of the names of the players who won, and **P** is how many points they scored.

## Constraints

* 1 ≤ **T** ≤ 10000
* 1 ≤ **N** ≤ 10
* 1 ≤ **len(name)** ≤ 20
* All names in a given game are unique

## Sample Input

2

3

Ethan

25 7- 5/ 2/ X X -7 81 X 7/

Jim

71 8/ 9/ X 1/ X 3- 17 23 X

Stephen

9/ 5/ X X X 1- -- X 3/ 52

3

Bob

X X X 52 8- -- 13 81 3/ 23

Bill

-- -- 1- 5- -- 7/ -- 1- 2- X

Brad

-8 X X X 34 3/ 23 -2 5- 35

## Sample Output

Stephen wins with 140 points!

Bob, Brad tie with 117 points!

# Problem 4: Inverted Inverted Index

## The Problem

Josiah heard about a tool called an inverted index one day. An inverted index is a data structure that maps a word to its corresponding document and location. For example, imagine that someone searches the word “hello” in a search engine. The engine uses the inverted index to look up which documents contain that search along with where (what index) those results appear.

Josiah got access to a complete inverted index, but he wants to reconstruct the original documents. Given a list of words along with which document and index each word is found, can you reconstruct the original documents?

## Input

The first line of input contains an integer, **T**, the number of test cases.

The first line of every test case contains an integer, **N**, the number of queries to follow. Each query contains a string **S** and two integers **D** and **I**. The query means that the string **S** is found inside the document with ID **D** at index **I**.

## Output

The output should contain one line per unique document ID in the input for each test case. Each line should contain the data in the original document, space-separated. The documents should be printed in order of their ID. No extra newlines are required between test cases.

## Constraints

* 1 ≤ **T** ≤ 100
* 1 ≤ **N** ≤ 5000
* 1 ≤ total documents ≤ 50
* 0 ≤ document ID ≤ 10000
* 1 ≤ **len(S)** ≤ 10

## Sample Input

1

8

an 3 2

is 3 1

Hello 1 0

:D 3 5

This 3 0

inverted 3 3

world! 1 1

index 3 4

## Sample Output

Hello world!

This is an inverted index :D

# Problem 5: Typing Statistics

## The Problem

Jacob has gotten very fast at typing. Since he is interested in probability, he would like you to assist him in writing a program that performs some statistics on his typing. He notices that the probability of typing a one-character word is the probability of pressing any character key followed by the space bar, or **26/27 \* 1/27**, which is **26/729**. We may assume that each key, including the space bar, has the same chance of being pressed.

Given a keyboard with **K** characters, what is the probability of typing a word with **N** letters? What about the probability of typing a word with the same length where the letters are unique? The probability should be expressed as a ***reduced*** fraction.

## Input

The first line of input contains an integer, **T**, the number of test cases.

Each test case contains two integers, **K** and **N**. **K** is the number of character keys on the keyboard (not including the space bar), and **N** is the length of the word to type.

## Output

There should be one line of output per test case. Each line will have two reduced fractions: one for the probability of typing a word with **N** characters, and one for the probability of typing a word with **N** characters where the characters are all unique.

If the probability is 0 (or impossible), print “0” instead of a fraction.

## Constraints

* 1 ≤ **T** ≤ 500
* 1 ≤ **K** ≤ 1018
* 0 ≤ **N** ≤ 64
* The numerator and denominator, before reducing, can be stored in a 64-bit integer.

## Sample Input

4

26 0

26 1

5 1

2 3

## Sample Output

1/27, 1/27

26/729, 26/729

5/36, 5/36

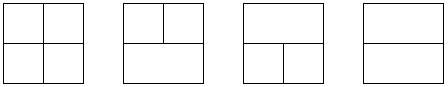
8/81, 0

# Problem 6: Wall Building

## The Problem

James has been put in charge of creating a wall of height **N** and width **M** out of bricks. However, these bricks are not your average bricks. The bricks can be any width, up to **M**. James has an infinite number of these bricks. He wants to know how many ways he can create this wall using any type of brick.

For example, suppose he wants to make a wall of height **2** and width **2**. Here are the possible formations:



As shown in the image above, there are **4** ways to create a wall with bricks of up to width **2**. Can you help James solve this problem? When computing the result, you must output it modulo **109 + 7**.

## Input

The first line contains an integer **T**, the number of test cases.

Each test case contains a line with two integers, **N** and **M**, the height and width of the wall to build.

## Output

There should be **T** lines of output. For each test case, print the number of ways James can create a wall of size **N** by **M** modulo **109 + 7**.

## Constraints

* 1 ≤ **T** ≤ 2 \* 105
* 1 ≤ **N**, **M** ≤ 106

## Sample Input

1

2 2

## Sample Output

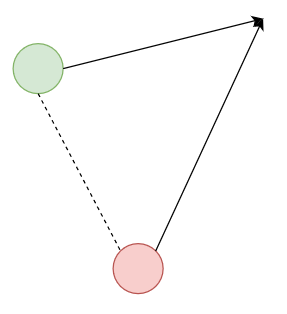
4

# Problem 7: Target Acquired

## The Problem

Melchi is working on creating a top-down shooter game. He ran into an interesting problem and wants to see how to solve it. In the game, there is a player and an enemy. The player is moving at a certain speed in a certain direction (angle). The enemy will shoot at the player, but he wants to be smart about where he fires. Instead of firing directly at the player, he wants to fire in the correct direction so that the bullet hits the player if the player were to keep moving in the same speed and direction. Can you help Melchi code a solution?

In the image below, the green represents the player, and the red represents the enemy. The dotted line is where the enemy *would* shoot if he were not smart. The solid line from the enemy is where the smart enemy shoots to make the bullet hit the player.



What angle should the enemy shoot to intercept the player?

## Input

The first line contains an integer, **T**, the number of test cases.

The following **T** lines contain the following floating-point numbers: **px**, **py**, **pdir**, **pspeed**, **ex**, **ey**, **bspeed**.

* **px**, **py**: The (x, y) location of the player
* **pdir**: The angle that the player is moving, in radians
* **pspeed**: The speed that the layer is moving, in pixels per second
* **ex**, **ey**: The (x, y) location of the enemy
* **bspeed**: The speed of the bullet to be shot, in pixels per second

## Output

There should be **T** lines of output. Each line contains the optimal angle, in radians, for the enemy to shoot the bullet. The output should have two decimal places after the point, and it should be **between 0 and 2\*π**.

## Constraints

* 1 ≤ **T** ≤ 1000
* -10000 ≤ **px**, **py**, **ex**, **ey** ≤ 10000
* 0 ≤ **pdir** ≤ 2\*π
* 0 ≤ **pspeed**, **bspeed** ≤ 1000
* There will be no more than 2 decimal places per input
* It will never take more than 106 seconds for the optimal bullet to hit the player

## Sample Input

1

0.00 0.00 0.78 200.00 -10.00 10.00 400.00

## Sample Output

6.02

# Problem 8: Museum Security

## The Problem

Ethan is working on a security system for a museum. The museum building is set up as a grid. Certain grid cells contain an artifact. Each of these artifacts must be secured with at least one camera. Ethan bought special cameras that only see one cell horizontally and vertically. In other words, a camera can see one cell above, below, right, and left of its current location.

Being frugal, Ethan does not want to spend a lot of money. He wants to minimize the number of cameras he needs to buy. Can you help him find the minimum number of cameras that can cover every artifact?

## Input

The first line will contain two integers, **N** and **M**, which correspond to the height and width of the grid.

The next **N** lines will contain **M** characters. A ‘.’ means a camera can be placed at that location, while a ‘#’ means there is an artifact at that location.

## Output

Output the minimum number of cameras Ethan needs to buy. If it is impossible to cover every artifact, output “impossible”.

## Constraints

* 1 ≤ **N**, **M** ≤ 15
* A camera cannot be on top of an artifact

## Sample Input

3 3

.#.

#.#

.#.

## Sample Output

1