# Problem 1: Fuzzy Socks

## The Problem

Katie was doing laundry one day, and she put all her clothes away. She really likes fuzzy socks, so she has a lot of them. In the morning, she noticed that one pair... wait, two pairs... wait, LOTS of pairs of socks were mismatched! Can you help her determine how many pairs of socks she needs to unfold to re-match them?

For example, say Katie has these pairs: (green, green), (yellow, blue), (blue, red), (yellow, red). She needs to unfold the last 3 pairs to match the pairs back together.

## Input

The first line contains one integer, **N**, the number of pairs.

Each of the next **N** lines contains two integers, **X** and **Y**, the colors of the two socks in the pair.

## Constraints

* 1 ≤ **N** ≤ 105
* 1 ≤ **X, Y** ≤ 109
* All of the socks can be matched

## Output

Output one integer, **M**, the number of pairs Katie needs to unfold to match them all again.

## Sample Input

4

1 1

2 4

3 4

2 3

## Sample Output

3

# Problem 2: Julius’s Cipher

## The Problem

Julius was reading a scroll about cryptography, and he learned that many codes and ciphers can be cracked by examining the number of each letter. In English, the most common letters are e, t, a, i, o, and n. Therefore, by examining the most common letters in a coded message, Julius can often figure out which letters correspond to which.

Julius once decided to send a coded message to his friend Brutus, but he forgot to tell Brutus how the message was encoded, so Brutus has come to you to help him understand the message. He knows that Julius always encodes his messages by shifting each of the letters in the alphabet by a fixed amount, but he doesn't know how much they have been shifted. He also knows that Julius only uses words from the dictionary. Can you help Brutus decode the message?

## Input

The first line of input contains a single integer **N**, the number of words in the dictionary.

The next **N** lines each contain one word in the dictionary.

The final line of input contains the coded message that has been encoded by shifting each letter by a fixed amount.

## Constraints

* 1 ≤ **N** ≤ 104
* The total length of the input will not exceed 105 characters

## Output

Print the decoded message. There will be only one possible solution.

## Sample Input 1

3

tu

Brute

Et

Ds st Aqtsd

## Sample Output 1

Et tu Brute

## Sample Input 2

2

goose

duck

hygo hygo ksswi

## Sample Output 2

duck duck goose

# Problem 3: Rogue’s Gallery

## The Problem

Welcome to the crime lab. You are tasked with reconstructing a blurred, partial fingerprint to match one or more complete prints from our rogue's gallery.

Each rogue's fingerprint has been stored as a square, 2D array of characters from the set **|-o()**.

The partial match is also a square, 2D array of characters from the set **|-o()#**, where the **#** character indicates a blurred measurement and could match any fingerprint character. The partial match array is strictly smaller than the fingerprint arrays.

You need to check each rogue fingerprint array to determine if the partial fingerprint array could match it anywhere.

For example, in the sample problem, the partial fingerprint could match the first rogue's fingerprint, but not the second:

A screenshot of a cell phone

Description automatically generated

Since the only **-** character is in the first rogue's fingerprint, and the partial fingerprint fits, the only valid match is the first rogue's fingerprint.

## Input

The first line contains three integers: **R**, the number of rogues, **S**, the side length of a rogue's fingerprint, and **P**, the side length of the partial fingerprint.

The partial fingerprint match follows: the next **P** lines each contain **P** characters from the set **|-o()#**.

Then comes the Rogue's Gallery: **R** \* **S** lines, with **i**th rogue's fingerprint given by lines **i** \* **S** **through** **(i + 1) \* S - 1**. Each line contains **S** characters from the set **|-o()**.

## Constraints

* 1 ≤ **R** ≤ 10
* 3 ≤ **S** ≤ 100
* 2 ≤ **P** < **S**

## Output

For each suspect, in the order given, output one line: **ROGUEISH** if the rogue could match the fingerprint, or **NOT ROGUE** if the rogue could not match the fingerprint.

## Sample Input

2 3 2

-#

#-

--|

--|

ooo

()|

|o|

||o

## Sample Output

ROGUEISH

NOT ROGUE

# Problem 4: Semantic Versioning

## The Problem

Semantic versioning is a system used by many pieces of software to track the current version. You may have seen examples like **1.0.0** or **3.4.2**.

This versioning system has three parts: *major*, *minor*, and *patch*. A *major* version is when a breaking change is introduced. A *minor* version is when a non-breaking change is introduced. A *patch* version is when a bug is fixed.

Given a sequence of major, minor, and patch updates, can you tell what the current version of the software is if it starts at **0.0.0**?

## Input

The first line contains one integer, **N**, the number of updates. Each of the next lines contains a command from the following:

* **major**: Increment the major number, set minor and patch to 0
* **minor**: Increment the minor number, set patch to 0
* **patch**: Increment the patch number
* **release**: Print the current version number

## Constraints

1 ≤ **N** ≤ 104

## Output

There should be one line of output per **release** command: the version at that point in time.

## Sample Input

10

major

minor

minor

release

major

release

minor

patch

patch

release

## Sample Output

1.2.0

2.0.0

2.1.2

# Problem 5: Lost Undo

## The Problem

Ryan was using his calculator to solve some math problems. Sometimes, he would input the wrong command, and he would have to undo it. He finished all of the commands, but somehow, the calculator didn't register one of the undo commands! Can you help Ryan figure out all of the possible answers if he places an undo anywhere in his list of commands?

For example, say Ryan enters the sequence **+5, undo, +10, -2, undo, -3**. The answer would be 7 because the **+5** and **-2** commands are undone. However, one undo is missing. It could be after any one of the commands. Because of that, the possible answers are -3 and 10. See the example below.

* +5, **undo**, undo, +10, -2, undo, -3 = *invalid* (underflow)
* +5, undo, **undo**, +10, -2, undo, -3 = *invalid* (underflow)
* +5, undo, +10, **undo**, -2, undo, -3 = -3
* +5, undo, +10, -2, **undo**, undo, -3 = -3
* +5, undo, +10, -2, undo, -3, **undo** = 10

## Input

The first line contains one integer, **N**, the number of commands.

The next **N** lines contain a command with one of the following:

* **undo**: Takes back the last command on the stack.
* **+[number]**: Adds a number to the result (e.g. +5)
* **-[number]**: Subtracts a number from the result (e.g. -10)

## Constraints

* 1 ≤ **N** ≤ 105
* It is guaranteed that the sequence of commands is valid (the stack will not underflow)
* Adding an undo that makes the stack underflow is invalid
* There will be at least one valid answer after adding an undo anywhere

## Output

The first line consists of one integer, **P**, the number of possible answers.

The next line contains a space-separated list of possible answers **in unique, sorted order**.

## Sample Input

6

+5

undo

+10

-2

undo

-3

## Sample Output

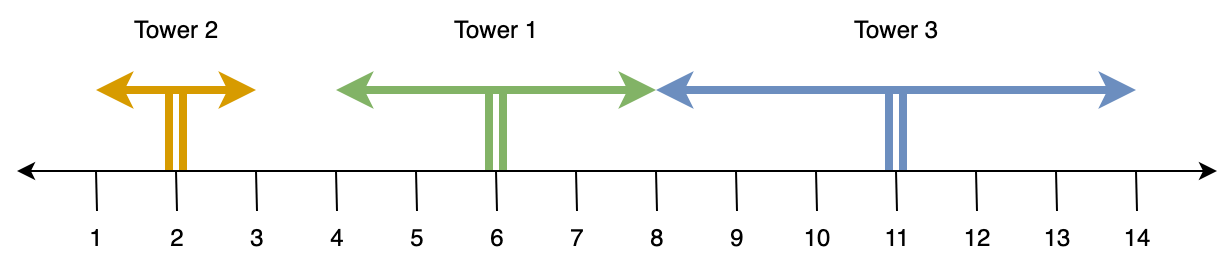
2

-3 10

# Problem 6: Teleportation Network

## The Problem

Zac lives in a one-dimensional world. He is building a teleportation network consisting of teleporter towers. Each tower has a position and a radius. Two towers are connected if their range overlaps or touches. Each day, he builds one tower and adds it to the network. Can you help him find out how many clusters of connected towers are in the network?



See the example scenario above:

* Day 1: Tower 1 at position 6, radius 2. **1 cluster ([T1])**
* Day 2: Tower 2 at position 2, radius 1. **2 clusters ([T1], [T2])**
* Day 3: Tower 3 at position 11, radius 3. **2 clusters ([T1, T3], [T2])**

The completed network consists of 2 clusters.

## Input

The first line contains one integer, **N**, the number of days.

The next **N** lines contain two integers, **P** and **R**: the position and radius of the tower for that day.

## Constraints

* 1 ≤ **N** ≤ 105
* 0 ≤ **P, R** ≤ 109

## Output

Output one integer, **C**, the number of clusters in the completed teleportation network.

## Sample Input

3

6 2

2 1

11 3

## Sample Output

2

# Problem 7: Rosetta Repo

## The Problem

Indiana Johnson was a history teacher at Harvard, and he spent his weekends fighting off snakes and searching for ancient artifacts. For many years he came across ancient writings in a language called Schaubese, but he was never able to understand the language. Then one day while he was in Egypt, he found a GitHub repo with the same code written in Schaubese and Python. How wonderful this was! Now the whole world could understand the ancient language of Schaubese. But there's just one problem: Indiana Johnson isn't very good at programming, so he's asked you to write a program to translate all of the ancient Schaubese scripts into modern, readable Python.

## Input

The input for your program is grouped in three chunks, each separated by two dashes. The first chunk is a script written in Python, the second chunk is an equivalent script written in Schaubese, and the third chunk is an ancient Schaubese script for you to translate.

The scripts may have spaces for indentation, but they will not have trailing spaces or tab characters anywhere. The Schaubese script will be a letter-for-letter translation of the Python script, and each letter in Python will correspond to exactly one letter in Schaubese.

## Constraints

The length of the input will not exceed 105 characters

## Output

Print the Python translation of the Schaubese script. If a character in Schaubese is not determined by the input, print a question mark instead. Note that the input and output are case sensitive.

## Sample Input

print("Hello World")

--

ocwHk\*~xNiiz LzciW~^

--

ocwHk\*~mzzWqeN LzciW~^

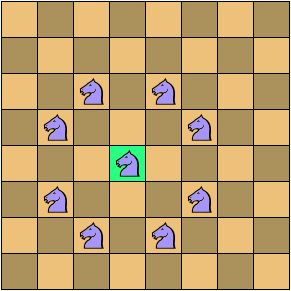
## Sample Output

print("?ood??e World")

# Problem 8: Knight Racing

## The Problem

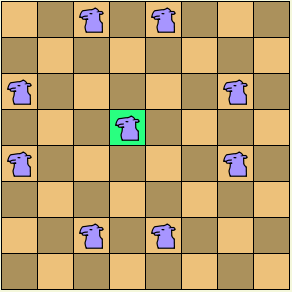
In the game of chess, a knight moves in a *L*-shape. More formally, it is called a “(1, 2) leaper”, because, if its current position is given by (*x, y*), it can move to any unoccupied square of the form (x±1, y±2), or (x±2, y±1).



This image shows the possible moves for the central, highlighted knight.

There is a sequence of moves for a knight that will take it from any square to any other square (the so-called “Knight’s Tour”).

But what about the Knight’s less popular friend, the Camel? The Camel is a (1, 3) leaper. It can move to any square of the form (x±1, y±3), or (x±3, y±1):



This image shows the valid moves for the central, highlighted camel (is there a “Camel’s Tour?” This is left as an exercise for the reader).

In a race from one square to another, who would win? The camel can move faster, but the knight is more agile. Your task is to decide, given a starting (row, column) position and ending (row, column) position on the chessboard, whether the camel or knight could make the journey in the fewest moves.

## Input

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

The next **T** lines contain four integers of the form **R1 C1 R2 C2**, where (**R1, C1**) gives the starting row and column and (**R2, C2**) gives the destination row and column.

## Constraints

* 1 ≤ **T** ≤ 100
* 1 ≤ **R1, C1, R2, C2** ≤ 8

## Output

For each test case, output a single line of the form "KNIGHT" if the knight wins, "CAMEL" if the camel wins, or "TIE" if the race is a tie.

## Sample Input

2

1 1 2 3

1 1 2 4

## Sample Output

KNIGHT

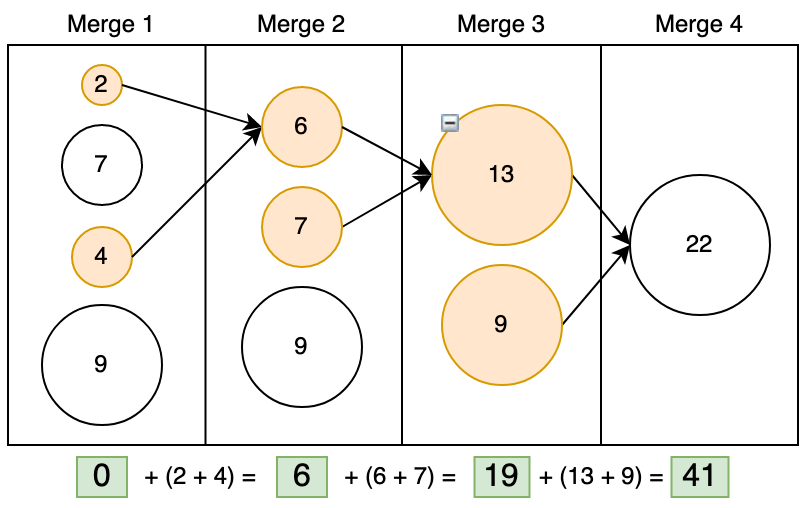
CAMEL

# Problem 9: Gold Merge

## The Problem

Jason has discovered hundreds of gold deposits around the world! Each deposit has a given weight. He wants to merge the deposits into one large deposit. He can merge two deposits at a time. The cost of a merge is the summed weights of the two deposits. What is the minimum cost of merging all of the deposits in any order?

For example, say Jason has deposits with weights 2, 7, 4, and 9. The optimal merge costs 41:



## Input

The first line contains one integer, **N**, the number of deposits.

The next **N** lines contain one integer each, **W**, the weight of each deposit.

## Constraints

* 1 ≤ **N** ≤ 105
* The minimum cost will not exceed a signed integer

## Output

Output one integer, **C**, the minimum cost of merging all the deposits into one.

## Sample Input

4

2

7

4

9

## Sample Output

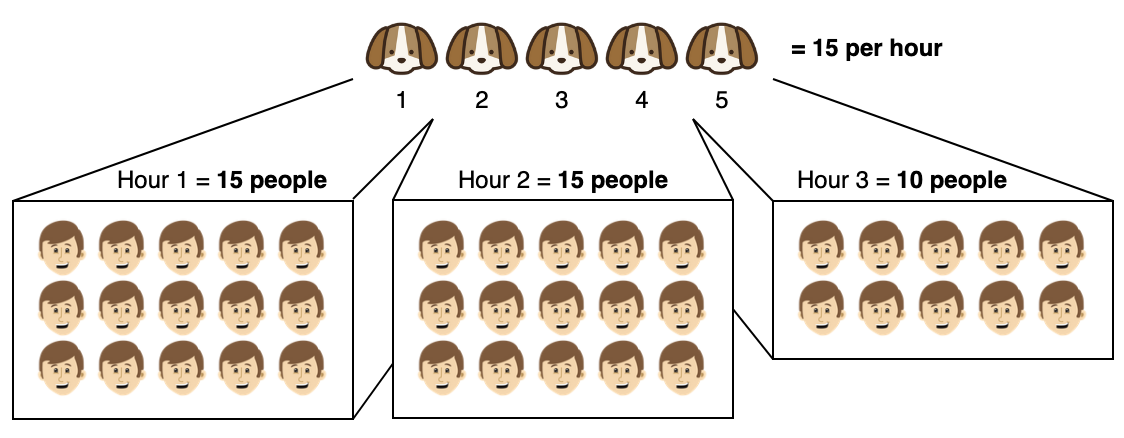
41

# Problem 10: Therapy Dogs

## The Problem

Everyone in the class was sad about their test coming up the next week. However, they heard that there were therapy dogs on campus! They decided to go see the dogs. One therapy dog can cheer up a given amount of people in an hour. You are given **N** therapy dogs and **P** people, along with one **X** for each dog: the number of people that dog can cheer up in one hour. How many hours will it take to cheer everyone up?

For example, there are 5 dogs and 40 people. The dogs can cheer up 1, 2, 3, 4, and 5 people respectively. As shown in the image below, it will take them three hours to cheer all 40 people up.



## Input

The first line contains two integers, **N** and **P**, the number of therapy dogs and people.

The next **N** lines contain one integer each, **X**, the number of people that dog can cheer up in an hour.

## Constraints

* 1 ≤ **N** ≤ 105
* 1 ≤ **P, X** ≤ 109

## Output

Output one integer, **H**, the number of hours it takes to cheer everyone up.

## Sample Input

5 40

1

2

3

4

5

## Sample Output

3