CALICO Spring '22 Online

May 21st, 2022
180 Minutes
11+3 Questions
Problem Packet by CALICO

Table of Contents

Problem 1: RGB for You and Me
3 Points

Problem ID: `rgb`
Rank: 1

Introduction

A Bayer filter is a type of color filter array, or CFA. It represents the pattern of red, green, and blue color filters on a grid of photosensors, and is most commonly used in the image sensors of digital cameras. The filter is unique in the way that it uses twice as many green sensors as red or blue sensors.

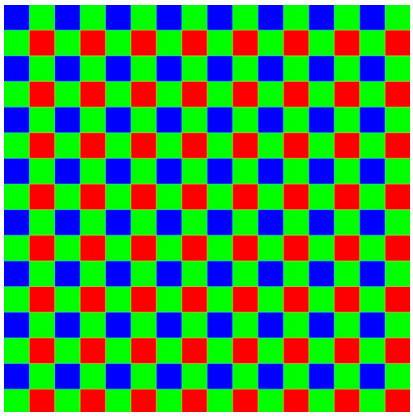
Problem Statement

Your task is to create a program that will output a Bayer filter with $\mathbf{R}$ rows and $\mathbf{C}$ columns.

A Bayer filter is defined as follows:

- Red, green, and blue color filters should be represented by the characters R, G, and B respectively
- The top-left corner of the filter should contain a blue color filter
- Every slot adjacent to a blue or red color filter (excluding diagonals) should contain a green color filter
- Every slot diagonal to a blue color filter should contain a red color filter, and vice versa

Here's an image of a 16x16 Bayer filter for reference:



Input Format

The first line of the input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- A single line consists of two integers $\mathbf{R}$ $\mathbf{C}$ separated by an $\mathtt{x}$ where:
  - The positive integer value $\mathbf{R}$ denotes the number of rows the Bayer filter must have
  - The positive integer value $\mathbf{C}$ denotes the number of columns the Bayer filter must have

Output Format

For each test case, your program should output a Bayer filter with the dimensions $\mathbf{R}$ and $\mathbf{C}$. Each test case output should be separated by a blank line.

Problem Constraints

$1 \leq \mathbf{T} \leq 100$

$1 \leq \mathbf{R}, \mathbf{C} \leq 100$

The sum of $\mathbf{R} \times \mathbf{C}$ across all test cases does not exceed 10^5.

Sample Test Cases

Sample Input

```
3
1x8
2x2
6x5
```

Sample Output

```
BGBGBGBG

BG
GR

BGBGB
GRGRG
BGBGB
GRGRG
BGBGB
GRGRG
```

Problem 2: What's Up?
3 Point(s)

Problem ID: `updog`
Rank: 1

Introduction

Does it smell like updog in here? You never know, but you don't want to be caught off guard either! You're sick of getting tricked by your peers and their schemes, and you never want to hear the cursed phrase "Nothing much, what's up with you" ever again.

Problem Statement

Your task is to respond accordingly to an input statement in the following format:
`it smells like <KEYWORD(S)> in here`

- If the first keyword in `<KEYWORD(S)>` begin with `up`, your program should output `what's that`
- Otherwise, your program should output `what's <KEYWORD(S)>` while maintaining its original capitalization and spacing

Input Format

The first line of the input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. Each test case contains a single line in the following format:

`it smells like <KEYWORD(S)> in here`

Output Format

For each test case, your program should output a single line containing the correct response to the input statement.

Problem Constraints

$1 \leq \mathbf{T} \leq 100$

The length of the input message will not exceed 100.

`<KEYWORD(S)>` is not an empty string.

`<KEYWORD(S)>` may contain numbers, special characters, or spaces.

The input will not contain consecutive spaces.

All inputs will be entirely in lowercase.

Sample Test Cases

Sample Input

```
6
it smells like updog in here
it smells like fish in here
it smells like upholstery in here
it smells like in here in here
it smells like pneumonoultramicroscopicsilicovolcanoconiosis in here
it smells like calico spring '22 online? im so glad you asked. in here
```

Sample Output

```
what's that
what's fish
what's that
what's in here
what's pneumonoultramicroscopicsilicovolcanoconiosis
what's calico spring '22 online? im so glad you asked.
```

Problem 3: Not Quite Fibonacci
3+2 Points

Problem ID: `trib`
Rank: `1+2`

Introduction

While listening to Mr. Recursion talk about Fibonacci numbers for the 11235813213455th time, you decided to invent a number sequence of your own! Beginning with -1, 0, and 1, you determine the next number by summing the previous *three* numbers in the sequence instead of the previous two. These are the *Tribonacci* numbers!

Problem Statement

Find the $\mathbf{N}^{\text{th}}$ Tribonacci number, $T_{\boldsymbol{N}}$.

The -1st, 0th, and 1st Tribonacci numbers are defined to be -1, 0, and 1 respectively. All Tribonacci numbers are equal to the sum of the three Tribonacci numbers before it. In other words:

$T_{-1} = -1$, $T_0 = 0$, $T_1 = 1$
$T_K = T_{K-1} + T_{K-2} + T_{K-3}$ where K can be any integer

The first few Tribonacci numbers are as follows:

T_{-1}	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
-1	0	1	0	1	2	3	6	11	20	37

Input Format

The first line of the input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. Each test case is described in a single line containing an integer $\mathbf{N}$ denoting the Tribonacci number you must find, T_N.

Output Format

For each test case, output a single line containing an integer denoting the $\mathbf{N}^{\text{th}}$ Tribonacci number, T_N.

Problem Constraints

$1 \leq \mathbf{T} \leq 100$
abs(T_i), the absolute value of the ith Tribonacci number, is guaranteed to be less than 10^9 for all values of $-60 \leq i \leq 30$.

Main Test Set

$0 \leq \mathbf{N} \leq 30$

Bonus Test Set

$-60 \leq \mathbf{N} \leq 30$
To find Tribonacci numbers of negative $\mathbf{N}$, algebraically rearrange the formula to solve for T_{K-3}.

Sample Test Cases

Sample Input

```
6
0
1
2
5
10
27
```

Sample Output

```
0
1
0
3
68
2145013
```

Sample Explanations

For test cases #1 and #2, this is because the 0th and 1st Tribonacci numbers are defined to be 0 and 1 respectively.

For test case #3, using the formula with K = 2, we have $T_2 = T_1 + T_0 + T_{-1}$ = 1 + 0 + -1 = 0

For test case #4, using the formula with K = 5, we have $T_5 = T_4 + T_3 + T_2$ = 2 + 1 + 0 = 3

For test case #5, using the formula with K = 10. we have $T_{10} = T_9 + T_8 + T_7$ = 37 + 20 + 11 = 68

Sample Input

```
4
-2
-3
-4
-50
```

Sample Output

```
2
-1
-2
-3792150
```

Sample Explanations

Negative Tribonacci numbers are found by rearranging the Tribonacci formula to solve for T_{K-3}. Note that negative $\mathbf{N}$ values will only appear in the bonus test set.

Problem 4: Wordsearch
3+3 Point(s)

Problem ID: `wordsearch`
Rank: `1+2`

Note

This problem has a challenge version, Problem X: Wordsnake! Both problems are exactly the same except for the additional constraints in the problem statement. However, solutions to one may not necessarily also be valid solutions for the other.

Introduction

To prepare for the upcoming SAT, your English teacher Mrs. Boomer assigned you vocabulary *wordsearches* of all things to do for homework! However, you have way more important things to do with your life than boring wordsearches, so you take a picture of the puzzle and run OCR on it. Then, you decide to write a program to search the words for you instead!

Problem Statement

Given the hidden word as an uppercase string $\mathbf{S}$ and the wordsearch puzzle as an uppercase letter grid with $\mathbf{R}$ rows and $\mathbf{C}$ columns, find the hidden word in the puzzle and output a copy of the puzzle with all letters except the letters of the hidden word replaced with `#`.

The hidden word in the grid is a sequence of adjacent letters with the additional constraints:

- For the main test set, the word will be hidden horizontally from left to right
- For the bonus test set, the word can be hidden horizontally or vertically, and can also be reversed
    - In other words, it can be hidden horizontally rightward, horizontally leftward, vertically downward, or vertically upward

Note that these are not the only constraints. See the constraints section below for more constraints.

Input Format

The first line of input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains a single string $\mathbf{S}$ denoting the hidden word
- The second line contains two space separated positive integers $\mathbf{R}$ and $\mathbf{C}$ denoting the number of rows and columns of the wordsearch puzzle
- The next $\mathbf{R}$ lines contain $\mathbf{C}$ uppercase letters each, denoting the puzzle itself
- The final line is blank to separate individual test cases

Output Format

For each test case, output the following:

- The first $\mathbf{R}$ lines should contain $\mathbf{C}$ symbols in each line denoting the solved wordsearch
  - All letters not part of the hidden word should be replaced with a pound sign `#`
- The final line should be blank to separate individual test cases

Constraints

$1 \le |\mathbf{S}| \le 26$
In other words, the length of $\mathbf{S}$ does not exceed 26.
$1 \le \mathbf{R}, \mathbf{C} \le 300$
The sum of $\mathbf{R} \times \mathbf{C}$ across all test cases does not exceed 10^5.

$\mathbf{S}$ only contains letters from the uppercase alphabet: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`.
$\mathbf{S}$ contains at most one of each letter.
$\mathbf{S}$ may not be a real English word.
There will be exactly one complete instance of the hidden word in the entire puzzle.

Main Test Set

The word in the puzzle will be hidden horizontally.

Bonus Test Set

The word in the puzzle may be hidden horizontally or vertically, and may also be reversed.

Sample Test Cases

Sample Input

```
2
UNCOPYRIGHTABLE
6 30
UNCOPYRIGHTABLYVNWVMHJMYYPUPVI
OVVJOYPOIYNRTLYVPVXZVKVXCCNFTC
BEKDGVCZAFVQSGOLBEDYEYCCGAMBHD
RLQDONTDODRUGSKIDSQXRLQGFFQFEK
NCJRUNCOPYRIGHTABLEZTPHSWWRUGJ
AQZJOYEWTBUCBERKELEYCALICOLCKC

COPYRIGHTABLE
1 13
COPYRIGHTABLE
```

Sample Output

```
##############################
##############################
##############################
##############################
####UNCOPYRIGHTABLE###########
##############################

COPYRIGHTABLE
```

Sample Explanations

For test case 1, we have a puzzle with 6 rows and 30 columns. The word `UNCOPYRIGHTABLE` can be found starting from row 5 column 5 going horizontally rightwards. Note that the incomplete word `UNCOPYRIGHTABL` can be found at row 1 column 1 but since it isn't the full word, we ignore it.

For test case 2, the entire puzzle consists of the word `COPYRIGHTABLE` and no other letters, so no pound signs `#` are added.

Sample Input

```
2
SECRET
6 3
OBS
QNE
LAC
USR
BBE
ILT

AMOGUS
7 10
IEELYLTMDA
OXSUGOMAZJ
TKLRFDNCRO
AQTCLPUFPE
HELPMEIMXX
STUCKINAXX
WORDSEARCH
```

Sample Output

```
##S
##E
##C
##R
##E
##T

##########
##SUGOMA##
##########
##########
##########
##########
##########
```

Sample Explanations

For test case 1, the word `SECRET` can be found starting from row 1 column 3 going vertically downwards. Note that vertical words only show up in the bonus test set, not the main test set.

For test case 2, the word `AMOGUS` can be found starting from row 2 column 8 going horizontally leftwards in reverse. Note that reverse words only show up in the bonus test set, not the main test set.

Problem 5: i trusted you
6 Point(s)

Problem ID: `amogus`
Rank: 2

Introduction

Happy dated reference day! Today we'll be talking about the hit 2020 deception game Among Us, the family-friendly multiplayer game in which you must complete fun tasks with your crewmates before time runs out—but beware, there are imposters among us!!! You and your friend are both imposters, so you must work together in order to deceive your "friends" and win the game. Be careful; if either of you draws too much attention (I'm not saying the word), you might get voted out of the game! Thanks to good old-fashioned election fraud, however, you've gotten a sneak peak at everybody's else's votes.

Problem Statement

Your task is to create a program that will output you and your teammate's votes (imposters) given the votes of $\mathbf{N}$ other players (crewmates). Each player is identified using a number counting upwards from 1; you and your partner are assigned player numbers $\mathbf{N} + 1$ and $\mathbf{N} + 2$, respectively. The ith player's vote $\mathbf{S_i}$ will denote the number of the player they want eliminated.

For each test case, your program should output you and your teammate's votes according to the following criteria:

- If either imposter currently holds the most votes (without tying with any crewmate), you should both vote for the crewmate with the next most votes
    - However, if an imposter is guaranteed to hold the most votes (without tying with any other crewmate), you should both vote `SKIP`
- Otherwise, if any imposter is tied for the most votes with any crewmate, you should vote for the crewmate, and your teammate `SKIP` (regardless of who's tied with the crewmate)
- Otherwise, you should both vote for the crewmate with the least votes
- If multiple players are tied for any criteria, vote for the player with the lowest player number

Input Format

The first line of the input will contain a positive integer $\mathbf{T}$ denoting the number of test cases that follow. Each test case will have the following input:

- A first line containing the single positive integer $\mathbf{N}$ denoting the number of other players (crewmates) present in the game.
- A second line containing the space-separated sequence of $\mathbf{N}$ positive integers $\mathbf{S_{1..N}}$, denoting the player number voted for by each player.
- A blank line separating individual test cases.

Output Format

Your output should be created in the following format:

- A single line containing your vote and your teammate's vote, separated by a space. A vote is represented by the player's number, or the word `SKIP`

Problem Constraints

$1 \le \mathbf{T} \le 100$
$1 \le \mathbf{N} \le 1000$

Sample Test Cases

Sample Input

```
4
5
6 1 6 6 6

3
5 1 2

7
2 9 8 8 9 9 8

4
2 1 2 2
```

Sample Output

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

Sample Explanations

For Test Case #1:
You (Player 6) are guaranteed to lose, so both you and your teammate should vote `SKIP`

For Test Case #2:
Your partner (Player 5), Player 1, and Player 2 are tied for the most votes, so you should vote for Player 1 while your partner votes `SKIP`

For Test Case #3:
You (Player 8) and your partner (Player 9) are tied for the most votes, but you can create a tie by both voting for Player 2.

For Test Case #4:
Player 2 currently has the most votes, so you vote for Player 3 (since both Player 3 and Player 4 have no votes).

Problem 6: Fractals Against Programmability
7 Point(s)

Problem ID: `fractal`
Rank: `2`

Introduction

It turns out that building a real house of cards with real cards bought with real money is too expensive, and building a plain digital house of cards is not very impressive, so you decide to build a digital house of cards that's cool and recursive instead!

Problem Statement

Output a fractal house of cards with $\mathbf{N}$ layers.

The simplest fractal house of cards with 1 layer consists of just two cards leaning on each other in a single line, drawn with a forward slash `/` and a backslash `\`:

```
/\
```

To construct a fractal house of cards with k layers where k is a power of 2, build three fractal houses of cards with k / 2 layers in an equilateral triangle pattern by inserting spaces so that each house is aligned correctly. For example, here is a fractal house of cards with 8 layers:

```
       /\
      /\/\
     /\  /\
    /\/\/\/\
   /\      /\
  /\/\    /\/\
 /\  /\  /\  /\
/\/\/\/\/\/\/\
```

See the sample test cases below for more examples.

Input Format

The first line of input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. Each test case consists of a single line containing a positive integer $\mathbf{N}$ denoting the number of layers in the fractal house of cards you're trying to build.

Output Format

For each test case, output the following:

- The first $\mathbf{N}$ lines should contain the fractal house of cards
  - Each line should contain the forward slashes `/`, backslashes `\`, and spaces for each layer
  - The house is allowed to have trailing spaces on the right side of each layer
- The final line should be blank to separate individual test cases

Problem Constraints

$1 \le \mathbf{T} \le 100$

$1 \le \mathbf{N} \le 256$

$\mathbf{N}$ is a power of 2.

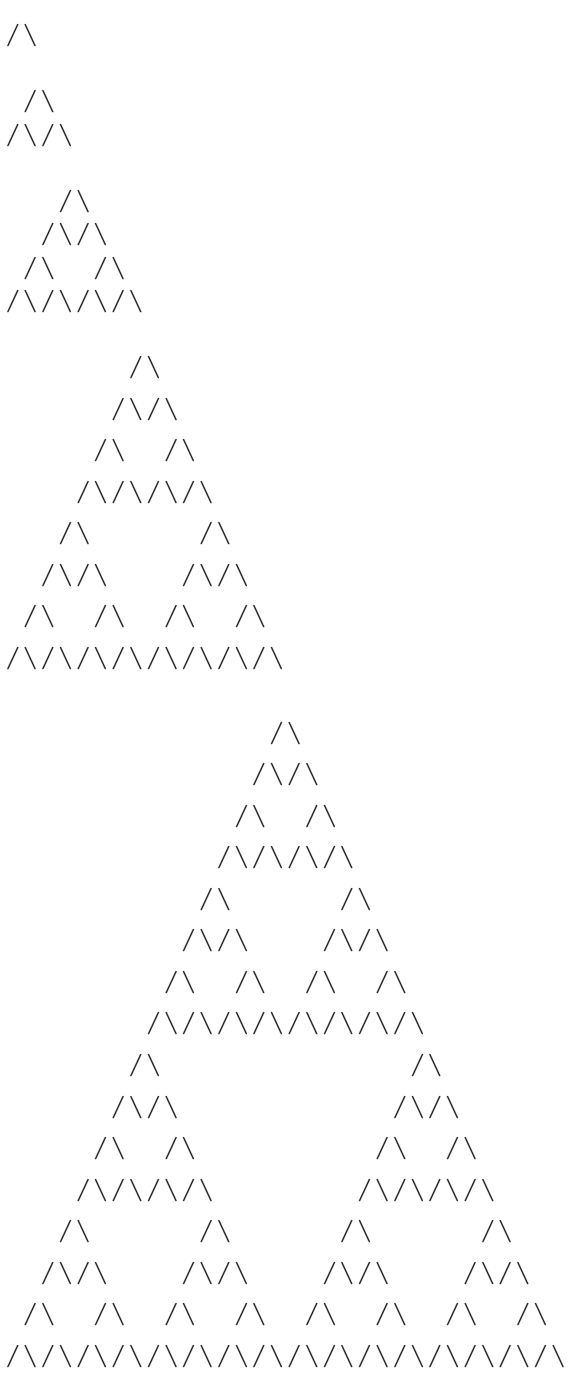
The sum of $\mathbf{N}^2$ across all test cases in an input does not exceed 10^5.

Sample Test Cases

Sample Input

```
5
1
2
4
8
16
```

Sample Output



Problem 7: Put a Knife In It
8+6 Point(s)

Problem ID: `cipher`
Rank: 2+3

Introduction

Take a stab—or 23—at this problem! Caesar may be remembered in the modern era for his salad dressing, but there is something else named after him: the Caesar cipher! In this problem, you will be working with basic cryptography to test your programming skills.

Problem Statement

Your task is to recover the original message from a ciphertext given a series of $\mathbf{N}$ ciphers used to encode it.

The ciphers encrypt the original message in series, with the output from one cipher becoming the input to the next. Each cipher can be any one of the following:

- A Caesar cipher with an offset x, where each alphabetic character in a string is shifted forward x letters in the alphabet (if $x = 2$, then "a" would become "c"). Non-alphabetic characters are ignored
  - A negative x symbolizes shifting each letter backwards instead of forwards. If the end of the alphabet is reached, you loop to the beginning of the alphabet, and vice versa.
- An Atbash cipher, where each alphabetic character in a string is flipped to its counterpart on the opposite end of the alphabet ("a" becomes "z" and "y" becomes "b"). Non-alphabetic characters are ignored
- A Reverse cipher, where the order of characters in a string is reversed ("atlanta" becomes "atnalta" and "racecar" becomes "racecar")

For your convenience, here are the lowercase letters of the English alphabet in order:
`abcdefghijklmnopqrstuvwxyz`

Input Format

The first line of the input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains the ciphertext to be decrypted
- The second line contains a positive integer $\mathbf{N}$ denoting the number of ciphers used to encrypt the message
- The third line contains a space-separated sequence of $\mathbf{N}$ ciphers. The ciphers represent the order in which the original message was encoded, from left to right. The first cipher is on the very left, and the last cipher is on the very right. Each cipher can be any one of the following:
  - Caesar ciphers, represented by a `C` immediately followed by an integer x, representing a Caesar cipher with an offset of x
  - Atbash ciphers, represented by an `A`
  - Reverse ciphers, represented by an `R`
- The final line is blank to separate individual test cases

Output Format

For each test case, output a single line containing the original message by decoding the given ciphertext.

Problem Constraints

$1 \le \mathbf{N} \le 10^3$
$-26 \le x \le 26$ for all Caesar ciphers.
The ciphertext will be non-empty. All letters in the ciphertexts will be entirely in lowercase.
The ciphertext may contain numbers, special characters, or spaces.

Main Test Set

$1 \le \mathbf{T} \le 100$
The length of the ciphertext will not exceed 100.

Bonus Test Set

$1 \le \mathbf{T} \le 10$

The length of the ciphertext will not exceed 10^5.

Sample Test Cases

Sample Input

```
4
lipps asvph
1
C4

iatcqsa lq cetwcq mpnwry 22 qrtwra!
2
C-5 A

hw'x p ebuler mpr wbmpr.
2
A C-10

!wvhwqrf hkw ir wvhu hkw qr nfxo grrj
3
C14 R C-11
```

Sample Output

```
hello world
welcome to calico spring 22 online!
it's a lovely day today.
good luck on the rest of the contest!
```

Sample Explanations

For test case #2:
The message `iatcqsa lq cetwcq mpnwry 22 qrtwra!` was first encrypted using a Caesar cipher shifting each character backwards 5 positions, followed by an Atbash cipher. Undoing these ciphers will decode the message `welcome to calico spring 22 online!`

Problem 8: C1000001
10 Points

Problem ID: `piano`
Rank: `3`

Introduction

Billy the billion-tentacled octopus wants to perform some profound pieces at his upcoming piano recital! He'll be showcasing his custom built piano with a few "engineering" quirks.

Billy's piano has 1000000 octaves with 12000004 keys numbered from 1 to 12000004! In scientific pitch notation, key 1 plays A0, 2 plays A#0, 3 plays B0, 4 plays C1, …, 40 plays C4 (middle C), …, 49 plays A4 (A440), …, and 12000004 plays C1000001 (which has such a high frequency it deatomizes air particles!).

With a billion tentacles, Billy can press as many notes as he needs simultaneously. Although the piano is long, he can instantly move to different parts of the piano when he isn't pressing down any keys. However, at any given point in time, he has limited reach! Help Billy determine what piece to play by finding the reach needed to play each piece!

Problem Statement

Given a piece of music with $\mathbf{N}$ timesteps involving a sequence of actions $\mathbf{A_1}, \mathbf{A_2}, \ldots, \mathbf{A_N}$ and keys $\mathbf{K_1}, \mathbf{K_2}, \ldots, \mathbf{K_N}$, where at timestep i, action $\mathbf{A_i}$ is taken with key $\mathbf{K_i}$, find the reach needed to play the entire piece.

An action can involve either pressing or releasing a key.

The reach needed at a given timestep is equal to the maximum difference between any two currently pressed key numbers. The reach needed to play the entire piece is equal to the maximum reach needed at any single timestep.

All keys begin in the released state, and are pressed/released in the order of their respective timestep.

Input Format

The first line of the input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains an integer $\mathbf{N}$ denoting the number of timesteps in the piece
- The next $\mathbf{N}$ lines contain 2 space-separated values each $\mathbf{A_i}$ $\mathbf{K_i}$, denoting actions and keys in the order they are played
  - The single character $\mathbf{A_i}$ denotes the action to perform on key $\mathbf{K_i}$, and is one of the following:
    - `P`, denoting $\mathbf{K_i}$ is pressed at timestep i
    - `R`, denoting $\mathbf{K_i}$ is released at timestep i
  - The integer $\mathbf{K_i}$ denotes the key to take action $\mathbf{A_i}$ with
- The final line is blank to separate individual test cases

Output Format

For each test case, output a single line containing a positive integer denoting the reach needed to play the entire piece.

Problem Constraints

$1 \le \mathbf{T} \le 100$

$2 \le \mathbf{N} \le 10^5$

$\mathbf{N}$ is even.

The sum of $\mathbf{N}$ across all test cases in an input does not exceed 10^5.

$1 \le \mathbf{K_i} \le 12 \times 10^6 + 4 = 12000004$

$\mathbf{A_i} \in \{$`P`, `R`$\}$

The piece as a whole is additionally subject to the following constraints:

- All keys begin in the released state
- Every pressed key will be released by the end of the piece
- The same key may be pressed and released multiple times
- If a key is pressed, it will not be pressed again until it is released and vice versa

Sample Test Cases

Sample Input

```
2
4
P 40
R 40
P 40
R 40

10
P 47
P 32
P 20
R 47
P 44
R 44
P 51
R 51
R 32
R 20
```

Sample Output

```
0
31
```

Sample Explanations

For test case 1, only one note is ever played at any given time. The maximum and minimum key number currently pressed is the same, so their difference is zero.

For test case 2, we have the following keys pressed and reach needed at each timestep:

Timestep	Keys	Reach
1	47	0 (47 - 47)
2	47, 32	15 (47 - 32)
3	47, 32, 20	27 (47 - 20)
4	32, 20	12 (32 - 20)
5	32, 20, 44	24 (44 - 20)

Timestep	Keys	Reach
6	32, 20	12 (32 - 20)
7	32, 20, 51	31 (51 - 20)
8	32, 20	12 (32 - 20)
9	20	0 (20 - 20)
10	(None)	0 (No keys)

The max reach at any timestep is 31, which occurs at step 7 while pressing keys 51, 32, and 20. The reach needed here is the largest distance between any two of the keys, 51 - 20 = 31.

Problem 9: Wordsnake
12 Point(s)

Problem ID: `wordsnake`
Rank: `3`

Note

This problem has a challenge version, Problem X: Wordsnake! Both problems are exactly the same except for the additional constraints in the problem statement. However, solutions to one may not necessarily also be valid solutions for the other.

Introduction

You turn in your assignment only to find Mrs. Boomer was just getting started! For tonight's homework, she invented a harder version of Wordsearch, where the words themselves can *bend*! She calls these puzzles *Wordsnakes*.

You run OCR on the puzzle again. But this time, you'll need to write a smarter program to search for words with bends!

Problem Statement

Given the hidden word as an uppercase string $\mathbf{S}$ and the wordsnake puzzle as an uppercase letter grid of $\mathbf{R}$ rows and $\mathbf{C}$ columns, find the hidden word in the puzzle and output a copy of the puzzle with all letters except the letters of the hidden word replaced with `#`.

The hidden word in the grid is a sequence of adjacent letters with the additional constraints:

- The hidden word can start at any location, and subsequent letters can follow in any of the 4 adjacent letters (up, down, left, right) regardless of where previous letters in the word are
    - This means the word can contain bends, but not self-intersections!
- The word may contain any number of bends

Note that these are not the only constraints. See the constraints section below for more constraints.

Input Format

The first line of input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line contains a single string $\mathbf{S}$ denoting the hidden word
- The second line contains two space separated positive integers $\mathbf{R}$ and $\mathbf{C}$ denoting the number of rows and columns of the wordsnake puzzle
- The next $\mathbf{R}$ lines contain $\mathbf{C}$ uppercase letters each denoting the letters of the wordsnake puzzle itself
- The final line is blank to separate individual test cases

Output Format

For each test case, output the following:

- The first $\mathbf{R}$ lines should contain $\mathbf{C}$ symbols in each line denoting the solved wordsnake
  - All letters not part of the hidden word should be replaced with a pound sign `#`
- The final line should be blank to separate individual test cases

Constraints

$1 \le |\mathbf{S}| \le 26$
In other words, the length of $\mathbf{S}$ does not exceed 26.
$1 \le \mathbf{R}, \mathbf{C} \le 300$
The sum of $\mathbf{R} \times \mathbf{C}$ across all test cases does not exceed 10^5.

$\mathbf{S}$ contains only letters from the uppercase alphabet: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`.
$\mathbf{S}$ contains at most one of each letter.
$\mathbf{S}$ may not be a real English word.
There will be exactly one complete instance of the hidden word in the entire puzzle.

The hidden word can start at any location, and subsequent letters can follow in any of the 4 adjacent letters (up, down, left, right) regardless of where previous letters are. This means the word can contain bends, but not self-intersections!

The word may contain any number of bends.

Sample Test Cases

Sample Input

```
3
ATMOSPHERIC
7 9
RWZMQVRLI
EOYOMMCMC
OAJHJHFPV
XTQROVWBW
MMSIALHSZ
ZOSPHERIC
VOTPTSDHC

AMOGUS
7 10
IEELYLTMDA
OXSUGOMAZJ
TKLRFDNCRO
AQTCLPUFPE
HELPMEIMXX
STUCKINAXX
WORDSNAKEX

ABCDEFGHIJKLMNOPQRSTUVWXYZ
4 13
MTGHIJPCJRSTU
YAFMLKTAYQPXV
QBEGZLMNOPSLW
CCDZVYOSNUZYX
```

Sample Output

```
#########
#########
#A#######
#T#######
#M#######
#OSPHERIC
#########

##########
##SUGOMA##
##########
##########
##########
##########
##########
##########

##GHIJ###RSTU
#AF##K###Q##V
#BE##LMNOP##W
#CD#######ZYX
```

Sample Explanations

For test case 1, we are given a puzzle with 7 rows and 9 columns. The word `ATMOSPHERIC` can be found starting from row 3 column 2 going vertically downward until row 6 column 2. Then, it bends to the right, continuing rightward until row 6 column 8.

For test case 2, the word `AMOGUS` can be found starting from row 2 column 8 going horizontal in reverse from right to left. This is an example of a valid test case whose solution contains no bends.

For test case 3, the word `ABCDEFGHIJKLMNOPQRSTUVWXYZ` can be found starting from row 2 column 2. Then it goes crazy all over the place, making a total of 9 bends.

Problem 10: Taxi Time
14 Point(s)

Problem ID: `taxi`
Rank: 3

Introduction

At your new startup Linear.ly, you have decided to disrupt the transportation industry with your revolutionary app idea—it gives users a rundown of the all cheapest ride options in their city! Lucky for you, most ride options follow a linear model: they first charge you a flat drop rate, and the fare increases at a constant rate from there. Your team has already given you the ability to access all transportation options in any city—it's up to you to finish the rest!

Problem Statement

Your task is to create a program that will output the name of the cheapest ride option in a given city for all possible integer distances given $\mathbf{N}$ taxis in the area with drop rates $\mathbf{B_1}$, $\mathbf{B_2}$, $\mathbf{\ldots}$, $\mathbf{B_N}$ and mileage rates $\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{\ldots}$, $\mathbf{M_N}$. If multiple taxis share the same cheapest cost at a given distance, output the one with the lowest mileage rate.

Input Format

The first line of the input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. For each test case:

- The first line consists of a city name and a positive integer $\mathbf{N}$ denoting the number of taxis that follow.
- The next $\mathbf{N}$ lines each consist of three space-separated values $\mathbf{s_i}$ $\mathbf{B_i}$ $\mathbf{M_i}$:
  - The string $\mathbf{s_i}$ denotes the name of taxi i
  - The non-negative integer $\mathbf{B_i}$ denotes the drop rate (flat starting fee) of taxi i in dollars.
  - The non-negative integer $\mathbf{M_i}$ denotes the mileage rate of taxi i in dollars per mile.
- The final line is blank to separate individual test cases.

Output Format

For each test case, your program should output the cheapest ride option in a given city for all possible distances in the following format:

```
<CITY NAME>:
<DISTANCE RANGE>: <NAME>
<DISTANCE RANGE>: <NAME>
<...>
<DISTANCE>+: <NAME>
```

- Each distance range should consist of non-negative integer mile distances for which a given ride option is the cheapest. The ranges may consist of the following:
  - A distance range consisting of two mile distances separated by a dash `-`, representing the minimum and maximum distances $\mathbf{X_1}$ $\mathbf{X_2}$ for which a given listing is the cheapest option.
  - A single distance value $\mathbf{X}$, for which a given listing is the cheapest option.
  - An open-ended distance range consisting of a single distance value $\mathbf{X}$ followed by a plus symbol `+`, representing all values above a minimum distance for which a given listing is the cheapest option.

Problem Constraints

$1 \le \mathbf{T} \le 10^3$

$1 \le \mathbf{N} \le 100$

$1 \le \mathbf{B_{1..N}}, \mathbf{M_{1..N}} \le 10^4$

$0 \le \mathbf{X_i} \le 10^6$ for all i

All names will be non-empty.

The length of all names will not exceed 100.

All names will only consist of lowercase letters, numbers, and underscores.

All ride options will have different names.

No two taxis will share the same drop rate and mileage rate.

Sample Test Cases

Sample Input:

```
4
rio_de_janeiro 3
yellow_cab 12 237
blue_transit 1626 84
smart_car 799 100

palo_alto 1
uber 510 137

berkeley 4
red_bus 0 1611
green_bus 0 1610
blue_bus 123 456
yellow_bus 2034 455

hangzhou 3
fly_taxicab 1134 211
premium_cab 753 211
blue_line 2649 0
```

Sample Output:

```
rio_de_janeiro:
0-5: yellow_cab
6-51: smart_car
52+: blue_transit

palo_alto:
0+: uber

berkeley:
0: green_bus
1-1910: blue_bus
1911+: yellow_bus

hangzhou:
0-8: premium_cab
9+: blue_line
```

Problem 11: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!0
20 Point(s)

Problem ID: `truths`
Rank: `4`

Introduction

Ben Bitdiddle has given up on learning the rules of boolean algebra! Instead, he decides to just straight up memorize ALL the true boolean expressions. You try to convince him that this idea is ridiculous, as the number of equations to memorize will grow exponentially with size, but he doesn't believe you! To convince him, you must find some evidence to show how absurdly large a task he has set for himself.

Problem Statement

Count the number of true boolean expressions that use exactly $\mathbf{N}$ symbols modulo $10^9 + 7$ (1000000007).

Boolean expressions are strings composed of only the following characters: `01!&|()`, where `0` and `1` are the boolean values for false and true, `!` is the unary prefix NOT operator, `&` is the binary infix AND operator, `|` is the binary infix OR operator, and the parentheses `()` are used to further specify evaluating order.

A boolean expression is true if and only if it is valid and evaluates to `1`. When evaluating, the standard order of operations should be respected: `()`, then `!`, then `&` and finally `|`.

Since the number of boolean expressions can be large, output your answer modulo $10^9 + 7$.

Input Format

The first line of the input contains a positive integer $\mathbf{T}$ denoting the number of test cases that follow. Each test case is described in a single line containing a single integer $\mathbf{N}$ denoting the number of symbols for the number of expressions you want to count.

Output Format

For each test case, output a single line containing the number of true boolean expressions that use exactly $\mathbf{N}$ symbols modulo $10^9 + 7$ (1000000007).

Constraints

$1 \leq \mathbf{T} \leq 100$
$1 \leq \mathbf{N} \leq 250$

Sample Test Cases

Sample Input

```
5
1
2
3
4
5
```

Sample Output

```
1
1
6
11
47
```

Sample Explanations

For test case #1, there is exactly 1 true boolean expression with length 1: `1`

For test case #2, there is exactly 1 true boolean expression with length 2: `!0`

For test case #3, there are 12 valid boolean expressions with length 3, of which 7 are also true. They are:

```
0|1         1&1         1|0         1|1         !!1         (1)
```

For test case #4, there are 22 valid boolean expressions with length 4, of which 11 are also true. They are:

```
0|!0        1&!0        1|!0        1|!1        !0&1        !0|0
!0|1        !1|1        !!!0        !(0)        (!0)
```

For test case #5, there are 90 valid boolean expressions with length 5, of which 47 are also true. They are:

```
0&0|1       0&1|1       0|0|1       0|1&1       0|1|0       0|1|1
0|!!1       0|(1)       1&0|1       1&1&1       1&1|0       1&1|1
1&!!1       1&(1)       1|0&0       1|0&1       1|0|0       1|0|1
1|1&0       1|1&1       1|1|0       1|1|1       1|!!0       1|!!1
1|(0)       1|(1)       !0&!0       !0|!0       !0|!1       !1|!0
!!0|1       !!1&1       !!1|0       !!1|1       !!!!1       !!(1)
!(!1)       (0|1)       (0)|1       (1&1)       (1|0)       (1|1)
(1)&1       (1)|0       (1)|1       (!!1)       ((1))
```