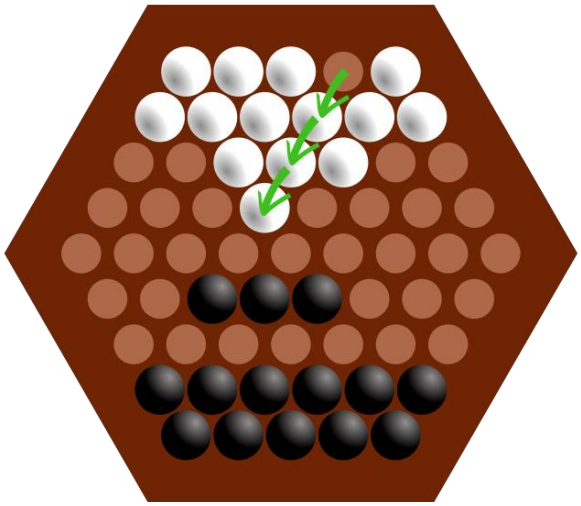
Abalone

Filename: `abalone`

Sharon enjoys playing this board game called Abalone. It is a two-player game played on a hexagonal board with 61 circular spaces. A space can be occupied by a black piece or a white piece. A piece can only be moved by the player who plays its color. In this problem, Sharon is playing the black color.



An illustration of the board, and a possible move by the white player.

For each move, a player moves a straight line of one, two, or three pieces of one color one space in any of six directions. A player can push their opponent's pieces that are in a line to their own if the pushing line has more pieces than the pushed line (three can push one or two, two can push one). Marbles must be pushed into an empty space or off the board. Currently Sharon is in the middle of a game and he began wondering, how many ways can he make a move that pushes a white piece off the board?

The Problem:

Given the state of the board during an Abalone game, determine how many moves can be made such that a white piece gets pushed off the board.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of games. For each game, 9 lines follow representing the board, the first having 5 characters, the second having 6, and so on until the 5th line which has 9 characters. Then the 6th line will have 8 characters, the 7th line will have 7, and so on until the 9th line which will have 5 characters.

Each character is either "`B`", "`W`", or "`.`" depending on whether the space is occupied by a black piece, white piece, or neither respectively. See sample input for an example.

The Output:

For each game, output a single line formatted as "`Game #i: c`" where *c* is an integer representing the number of moves that can be made that will push a white piece off the board.

(Sample Input and Output on the next page)

Sample Input:

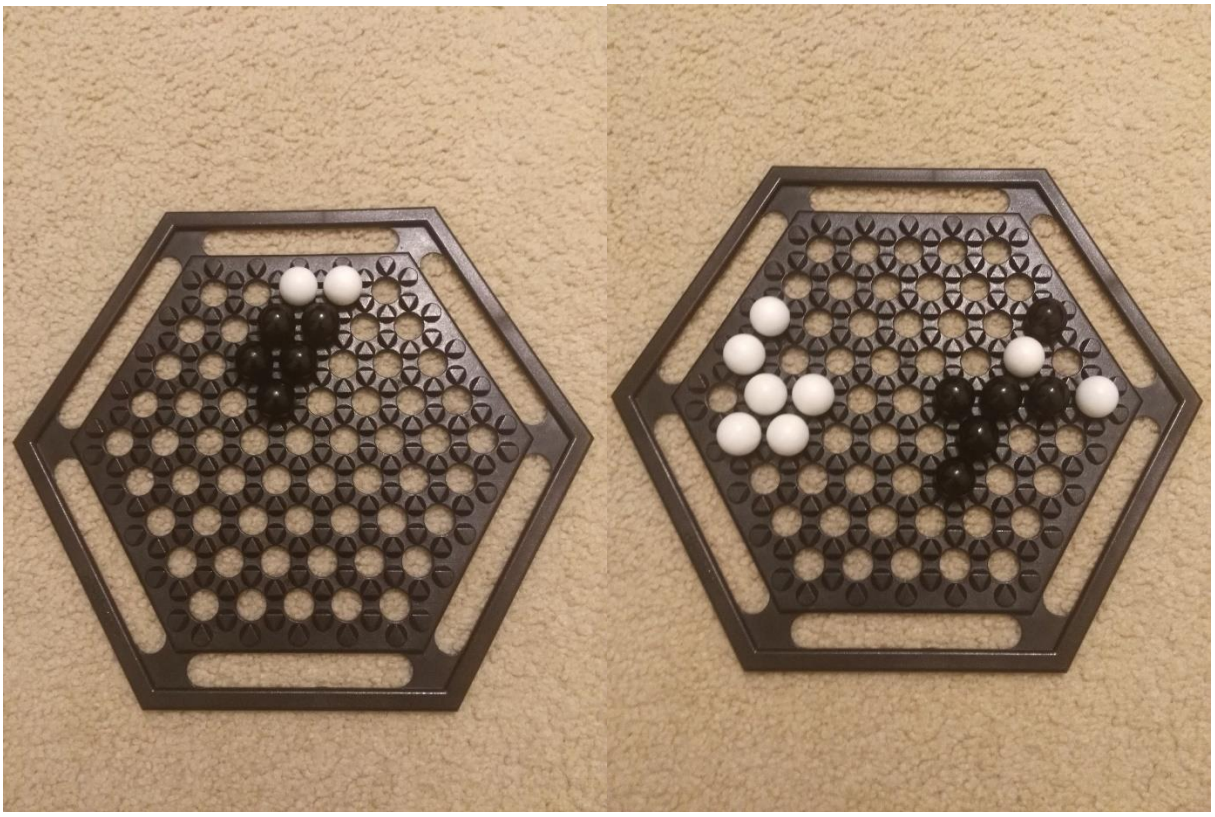
```
2

..WW.

..BB..

..BB...

...B....

.........

........

.......

......

.....

.....

......

W.....B

W.....W.

.WW..BBBW

WW...B..

....B..

......

.....
```

Sample Output:

```
Game #1: 3
Game #2: 2
```

Notes: There may be more or less pieces on the board than there are allowed to be in a real Abalone game. In the context of this problem this is not important.

The following are images of the state of the board in the first sample case (left), and the second sample case (right).



Happy Birthday!

Filename: `birthday`

Remember Lior, one of the main characters from the other two problems? You may not have read them yet. Fun fact: today is Lior's birthday! When I asked him what he wanted for his birthday, he wished for a geometry problem. However, as he may be finding out right now, there is no geometry problem on this set. Instead, for his birthday Lior received an array of *n* integers.

Lior loves math, so just like every math lover he has a set of favorite numbers. Lior wants to know how many times one of his favorite numbers shows up in the array. Your task is to solve this problem before Lior and his team does.

The Problem:

Given an array and a set of integers, determine the count of integers in the array that belong to the set.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of test cases. For each test case, three lines follow. The first of which has two integers, $1 \leq n,m \leq 10^4$, representing the number of elements in the array and the number of elements in the set, respectively. The second line has *n* integers, representing the array Lior received for his birthday. The third line contains *m* integers, representing the set of Lior's favorite numbers. The array may contain duplicate integers, but the set will not. Every element in the array or set will be a positive integer less than or equal to 10^9.

The Output:

For each test case, output a single line saying "`Birthday #i: c`" without the quotes, where `i` is the number of the test case, and `c` is the count of integers in the array that belong to Lior's set of favorite integers.

Sample Input:

```
2
5 3
1 2 3 4 2
1 2 4
10 3
1 2 3 4 5 6 7 8 9 10
35 73 115
```

Sample Output:

```
Birthday #1: 4
Birthday #2: 0
```

Grading Issue

Filename: `grading`

Arup's evil teaching assistant, John, is very careless. He haphazardly grades papers, giving random grades based on his mood. This causes issues, since the grading scale in his class only allows for grades between 0 and 100 (inclusive), and John sometimes gives grades that are much higher than 100, or even much lower than 0! Arup must now deal with the consequences of John's sloppiness, but since he is very busy, you must help him. Write a program that will determine whether the grade given by John is valid or not.

The Problem:

Given a number, determine if it is a valid grade.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of graded papers. Then, *t* lines follow, each with a single integer $-10^6 \leq n \leq 10^6$, representing the grade given by John for each paper.

The Output:

For each paper, output a single line saying "`Grade #i:`" followed by a space and either "`No Change`" or "`Change`" without the quotes, where `i` is the number of the test paper, depending on whether the grade is valid or invalid, respectively.

Sample Input:

```
5
94
-3
100
1000
12
```

Sample Output:

```
Grade #1: No Change
Grade #2: Change
Grade #3: No Change
Grade #4: Change
Grade #5: No Change
```

Game Gyms

Filename: `gyms`

Sharon was playing a game where he has to challenge a series of gyms in order to become champion. Each gym has a type, which is super effective against some other types. What Sharon noticed was very interesting; every gym in the series follows a gym that it is super effective against. For example, if the water gym has to be challenged right after the rock gym, it means that water is super effective against rock. Sharon was wondering if for a specific type chart - a chart that shows whether a type is super effective against another - it is possible to create a valid gym ordering. There must be exactly one gym of each type.

The Problem:

Given a type chart, find the lexicographically smallest (alphabetically first) gym ordering such that each gym is super effective against the prior gym in the series, or state that none exist.

The Input:

The first line of the input file begins with a single, positive integer, t, representing the number of type charts. For each type chart, the first line consists of a single integer $2 \leq n \leq 8$. Then, n lines follow. Each line consists of a string of at most 20 uppercase letters, representing the name of the type, followed by n integers, either 0 or 1. No two types will have the same name. The ith integer on that line is 1 if the current type is super effective against type i, in the order given in the input. A type may or may not be super effective against itself. Two types may be super effective against each other. See sample input for more info.

The Output:

For each game, output a single line saying "`Type Chart #i:`" without the quotes, where `i` is the number of the type chart, followed by a space and the lexicographically smallest ordering of the gyms, or print "`Impossible`" if there is none.

(Sample Input and Output are on the next page)

Sample Input:

```
2
6
ROCK 0 0 0 0 1 0
GRASS 1 0 0 0 0 1
POISON 0 1 0 0 0 0
PSYCHIC 0 0 1 0 0 0
FIRE 0 1 0 0 0 0
GROUND 1 0 1 0 1 0
3
ROCK 0 1 0
SCISSORS 0 0 1
PAPER 1 0 0
```

Sample Output:

```
Type Chart #1: FIRE ROCK GROUND GRASS POISON PSYCHIC
Type Chart #2: PAPER SCISSORS ROCK
```

Primedigital Numbers

Filename: `primedigital`

Lior likes showing off to Danny how well he can calculate numbers in his head. But one day as they were exploring the ruins of an ancient castle they discovered a new mathematical concept that they have never seen before. They have deemed this mathematical concept super important, because it dealt with numbers that only include prime digits, and they already know that prime numbers are important. Therefore, they have named all numbers that only include prime digits "Primedigital" numbers. For example, "3", "27" and "333" are primedigital numbers, while "6", "413", and "520" are not. In order to properly master this knowledge, they must become good at counting the number of primedigital numbers in any range. Your task is at follows: given an integer l and an integer r such that $l < r$, find the number of primedigital numbers that are greater than or equal to l and less than or equal to r.

The Problem:

Determine the number of primedigital numbers in between l and r.

The Input:

The first line of the input file begins with a single, positive integer, t, representing the number of test cases. Then, t lines follow, each with two integers $1 \leq l, r \leq 10^{18}$, representing the range of numbers as stated above.

The Output:

For each paper, output a single line saying "`Range #i: c`" without the quotes, where `i` is the number of the range and `c` is the number of primedigital numbers in the range, respectively.

Sample Input:

```
3
1 4
1 10
222 235
```

Sample Output:

```
Range #1: 2
Range #2: 4
Range #3: 7
```

Prism Pyramids

Filename: `pyramid`

Danny and Lior consider themselves extraordinary engineers, and many marvel at their exceptional architectural expertise. However, they have a task ahead of them. They are given *n* bricks from top to bottom with properties *w*, *l*, and *h*, denoting width, length, and height respectively, and they must determine if the bricks make a proper pyramid. The bricks make a proper pyramid if the brick on top has a strictly smaller volume, width and length than the brick below it. The bricks are given in order and there is only one brick per level, so this should be easy. Your job is to solve this problem faster than Danny and Lior.

The Problem:

Given *n* prisms which make up a pyramid, you must determine whether or not is it a proper pyramid.

The Input:

The first line consists of one integer, *t*, denoting the number of pyramids. Then for every test case, an integer $1 \leq n \leq 100$ follows, and then *n* lines describing the bricks in order from top to bottom, with integers *w*, *l*, and *h*, such that $1 \leq$ w, *l*, $h \leq 100$.

The Output:

Output *t* lines formatted as "`Pyramid #i: Proper Pyramid`" or "`Pyramid #i: Improper Pyramid`" without the quotes, where `i` is the test case number, depending on whether or not the pyramid is proper according to the problem statement.

Sample Input:

```
3
1
1 2 3
4
1 1 1
2 2 2
3 3 3
4 4 4
3
1 1 5
2 2 1
4 5 6
```

Sample Output:

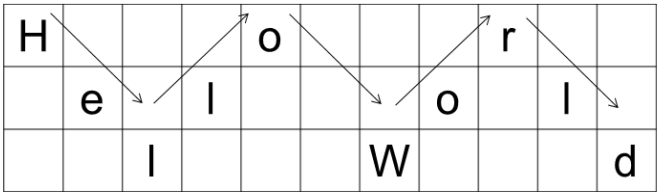
```
Pyramid #1: Proper Pyramid
Pyramid #2: Proper Pyramid
Pyramid #3: Improper Pyramid
```

Rail Fence Attackers

Filename: `railfence`

We are at war with the fencelings! Fortunately, we have intercepted their attack plans - we just need to decrypt them. The fencelings are true to their name, and they encrypt their messages using a "rail fence" cipher. In this cipher, the text is written diagonally and downwards on successive "rails" of an imaginary fence, then moving up when the bottom rail is reached. When the top rail is reached, the message is written downwards again until the whole text is written out. The message is then read off in rows. For example, if the text was "Hello World" and the number of rails was 3, then the process would look like this:

Original Message: Hello World



Encrypted Message: Horel ollWd

The fencelings have no concept of grammar, so they write their messages using strictly lowercase letters. Given a message written by the fencelings, print the decrypted message.

The Problem:

Decrypt a message using the rail fence cipher, and print the decrypted message.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of messages. Then, 2**t* lines follow, two for each test case. The first of which has two integers $2 \leq n, k \leq 10^6$, representing the length of the message and the number of rails used, respectively. The second line of a test case is a string of *n* lowercase characters representing the message itself.

The Output:

For each paper, output a single line saying "`Message #i: c`" without the quotes, where `i` is the number of the message and `c` is the decrypted message, respectively.

(Sample Input and Output are on the next page)

Sample Input:

```
2
19 2
wwlatcasxmeiltaktip
10 3
telherisra
```

Sample Output:

```
Message #1: wewillattackatsixpm
Message #2: threerails
```