Learn to Dance

Filename: `dance`

Ellie is a great dancer. Recently, she has been trying to learn a new dance by watching YouTube videos. She came up with a revolutionary method to learn the dance: she would write a program that tracks the dancer's footsteps, and save the data as a string of two characters, 'L' and 'R', depending on whether the dancer steps with their left foot or right foot. Ellie wants to figure out how many times she has to jump in this dance. She deduced that she must jump every time the dancer steps with the same foot twice in a row.

The Problem:

Given a string representing a dance, count the number of jumps.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of dances. For each dance, two lines follow. The first contains a single integer $1 \leq n \leq 50$ representing the number of steps. The second line contains the string, representing the steps themselves.

The Output:

For each test case, output a single line saying "`Dance #i: c`" without the quotes, where `i` is the number of the dance, and `c` is the number of jumps in the dance.

Sample Input:

```
2
7
LLRRRLL
4
RLRL
```

Sample Output:

```
Dance #1: 4
Dance #2: 0
```

Flipper-McFlopper

Filename: `flipper`

The star-bellied Sneetches, they want to have fun
Blooping and flooping till Sunday is done.
But one little chore they have left to do:
Test their Flipper-McFlopper machine for dear Sue.

The Flipper-McFlopper machine is quite grand. It takes in a single, quite large, operand.
We'll call it *x*, and so that you know, what happens inside is recorded below:
It does this until *x* ends in a 7 -- first add 1 or 2, then scale by 11.
The Flipper-McFlopper should always choose best to stop extra fast for every new test.

Find the fewest number of scales to get to the end,
assuming optimal picks (each either 1 or 2) for every addend.

The Problem:

Given x, calculate the minimum number of scales the Flipper-McFlopper must do before stopping.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of test cases. *t* lines follow, each containing a single integer, $1 \leq x \leq 10^{18}$, the input to the machine.

The Output:

For each test case, output a single line saying "`Input #i: s`" without the quotes, where `i` is the number of the test case, and `s` the minimum number of scales required.

Sample Input:

```
3
197
4
583920682924895290
```

Sample Output:

```
Input #1: 0
Input #2: 2
Input #3: 4
```

Note: For the first case the Flipper-McFlopper machine will stop instantly since the number already ends in a 7. For the second test case, ((4 +1)*11 +2)*11 = 627. It will stop here after multiplying twice. It is also possible to stop at 737 after 2 scales.

Strange Lottery Simulator

Filename: `lottery`

Billy came up with a unique lottery idea: participants sign up with their names, and a random string *s* is generated in the draw, and the winners are the people whose name has prefix *s*. Yet there is a special rule, sometimes the names of all participants are reversed.

So for example, if Sharon signed up, then his name was reversed, and the string generated was "no," then he would win! However, if the string generated was "sha," he would not win. His name could be reversed an arbitrary number of times.

Billy would like you to help him write a simulator for the lottery. It should handle queries of three types (*s* represents a string):

1 *s*, Add a participant with name *s*.

2 *s*, Generate a string *s* in the draw and print the number of winners.

3, Reverse the names of all participants added.

Can you help Billy write an efficient simulator?

The Problem:

Write a program to simulate Billy's lottery. It is guaranteed that there is at least one query of type 2.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of lotteries. For each lottery, multiple lines follow. The first contains a single integer $1 \leq q \leq 10^5$ representing the queries. Then, *q* lines follow, beginning with a single integer *t*, which is either 1, 2, or 3, denoting the type of the query. In queries of types 1 and 2, the integer will be followed by a space and then a string *s* of lowercase letters, whose length is at most 10^5. The sum of all string lengths in a test case is at most $6*10^5$.

The Output:

For each test case, output a single line saying "`Lottery #i:`" without the quotes, where `i` is the number of the lottery. Then, multiple lines should follow, one for each query of type 2, each with a single integer denoting the number of winners. Print a blank line after each test case.

(Sample Input and Output are on the next page)

Sample Input:

```
2
7
1 silly
1 billy
1 billiam
2 bill
2 ylli
3
2 ylli
5
1 carson
2 no
3
1 sharon
2 no
```

Sample Output:

```
Lottery #1:
2
0
2

Lottery #2:
0
1
```

Note: There is a blank line after the second test case.

Kindergarten Playsets

Filename: `playset`

Bobby is a toy manufacturer who makes playsets for children. This week, n kindergarten teachers ordered a playset from him. Due to limitations on his machine, Bobby may only manufacture playsets with x toys, such that $1 \leq x \leq m$, and he must choose the same x for all teachers. Bobby quickly realized that this may mean that he cannot satisfy all the teachers orders, since each teacher wants to be able to divide the toys evenly among the children in their classroom. He began wondering, what is the maximum amount of teachers he can satisfy, and how many unique values of x achieve this goal?

The Problem:

Determine the maximum amount of teachers that can be satisfied and the number of ways in which this can be done.

The Input:

The first line of the input file begins with a single, positive integer, t, representing the number of weeks. For each week, two lines follow.

The first contains two integers $1 \leq n, m \leq 10^3$, representing the number of kindergarten teachers and the maximum amount of toys that can be in a playset. The second line consists of n space separated integers, $1 \leq a_i \leq 10^3$, denoting the number of children in each classroom.

The Output:

For each test case, output a single line saying "`Week #i: c w`" without the quotes, where `i` is the number of the week, `c` is the maximum number of teachers satisfied and `w` is the number of ways to satisfy `c` teachers.

Sample Input:

```
3
5 25
10 3 5 7 20
3 100
20 25 15
2 10
20 35
```

Sample Output:

```
Week #1: 3 1
Week #2: 2 3
Week #3: 0 10
```

Note: In the second test case, it is possible to have a playset with 60 toys (divisible by 20 and 15), 75 toys (divisible by 25 and 15), or 100 toys (divisible by 25 and 20). It is impossible to satisfy all three teachers since we can make at most 100 toys.

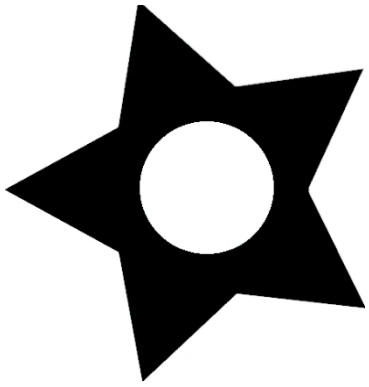
In the third test case, any playset of size 1 through 10 can be made. However, this does not satisfy the teachers.

Shuriken

Filename: `shuriken`

A shuriken is a geometric shape that can be described as a combination of multiple shapes. The following are instructions to draw an *n*-sided shuriken:

- Draw a filled-in regular polygon with *n* sides of length *s*.
- Each of the polygon's sides is the base of a filled-in isosceles triangle of height *h*.
- Erase a circle of diameter *d* from the middle of the shuriken. The circle must be entirely within the regular polygon.



A diagram representing a 5-sided shuriken.

The area of the above shuriken corresponds to the area of the shape in black. To become a true ninja master, you must be able to calculate it. Use the value of $\pi = 3.14159265359$ in your calculations.

The Problem:

Compute the total area of a shuriken.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of shurikens. For each shuriken, one line follows, each with four integers: $3 \leq n \leq 1000$, $1 \leq s, h, d \leq 1000$.

It is guaranteed that the shuriken is valid - the diameter is small enough for the circle to fit entirely inside the polygon.

The Output:

For each shuriken, output a single line saying "`Shuriken #i: c`" without the quotes, where `i` is the number of the shuriken, and `c` is the shuriken's area rounded to the nearest hundredth (1.234 rounds to 1.23 and 1.235 rounds to 1.24).

(Sample Input and Output are on the next page)

Sample Input:

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

Sample Output:

```
Shuriken #1: 28.86
Shuriken #2: 103.80
```

Several-Part Staircase

Filename: `staircase`

David's new company, the Ultimate Carpentry Foundation, has just come out with a new life changing product: the world's First Heavyweight Several-Part Staircase. As you may know, heavyweight staircases are difficult to manufacture, so the Ultimate Carpentry Foundation has instead decided to create several staircase parts and weld them together to make a single enormous staircase.

Each staircase part has a start and end height. A staircase part can be attached to another staircase part at either end, as long as the two ends that are being joined are the same height. Staircase parts can be used in either direction: going up or going down, but each part may only be used once in only one direction or not at all. Can David merge zero or more parts to create a staircase going from height 0 to height k, without using any part more than once?

The Problem:

Given the end heights of n staircase parts, check whether it is possible to combine some parts to make a staircase from height 0 to height *k*, using each staircase part no more than once.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of test cases. Each test case begins starts with two integers *n* and *k* ($1 \leq n \leq 3*10^5$, $0 \leq k \leq 10^9$), representing the number of staircase parts and the desired end height. *n* lines follow each containing two integers, a_i and b_i ($0 \leq a_i, b_i \leq 10^9$), representing the heights of the two ends of the *i*'th staircase.

The Output:

For each test case, output a single line saying "`Staircase #i: p`" without the quotes, where `i` is the number of the test case, and `p` is either "Possible" or "Impossible" depending on whether the target height can be reached.

(Sample Input and Output are on the next page)

Sample Input:

```
3
5 10
0 3
3 6
3 7
3 8
7 10
2 6
1 5
5 6
3 100
0 200
150 100
150 200
```

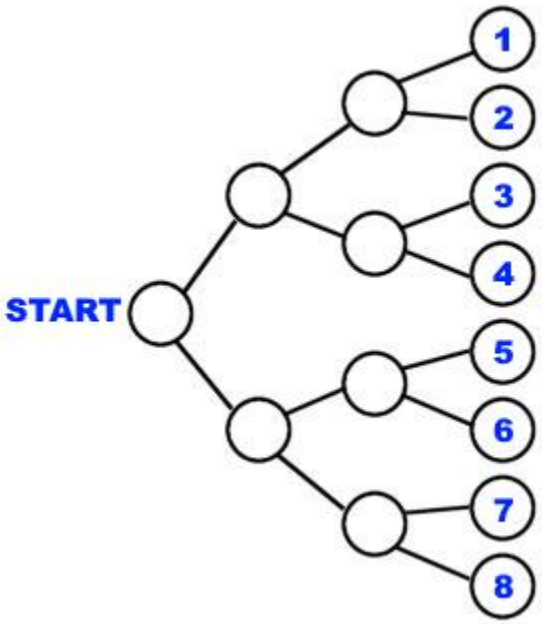
Sample Output:

```
Staircase #1: Possible
Staircase #2: Impossible
Staircase #3: Possible
```

Trolley Tracks

Filename: `trolley`

The Minigame Trolley is having its grand opening! It is a brand new trolley that takes its riders to play various minigames. The trolley starts on the left, and at each intersection it can go up or down, depending on the riders' votes. Each rider has a favorite minigame. If their favorite minigame is reachable, they will vote up or down, depending on the direction of their minigame. If their minigame is unreachable, they will abstain. The votes will be recounted at each intersection, and the trolley will go down if there are more "down" votes than "up" votes, and up otherwise. Finally, when the trolley reaches a minigame (numbered 1 through n), it will stop.



A diagram representing the trolley tracks.

As it is opening night, Pauline the trolley conductor would like to save time by asking the riders what their favorite minigame is at the start. After doing so, she would like to figure out the minigame the trolley will be headed to. This way, the trolley won't have to stop to recount the votes each time!

The Problem:

Given each rider's intended destination, determine the trolley's final destination.

The Input:

The first line of the input file begins with a single, positive integer, t, representing the number of trolleys. For each trolley, two lines follow. The first contains a single integer $1 \leq n \leq 18$, where 2^n represents the number of minigames. The second line consists of an array a of 2^n integers, where $1 \leq a[i] \leq 10^9$ represents the number of people whose favorite minigame is i.

The Output:

For each test case, output a single line saying "`Trolley #i: c`" without the quotes, where `i` is the trolley number, and `c` is the minigame that the trolley will reach in the end.

Sample Input:

```
2
3
2 4 2 1 6 4 7 1
1
100 100
```

Sample Output:

```
Trolley #1: 5
Trolley #2: 1
```