Thirty-first Annual
University of Central Florida

High School Programming Tournament

Problems

Problem Name	**Filename**
Naughty or Nice?	naughty
Yertle the Turtle	yertle
The Power of Two is a Curious Thing	power
Seahorse Shoes	shoes
Every Day I'm Shuffling	shuffling
Knights, Pirates, Ninjas	knights
Shifting Gears	shift
Jumping Fish	jump
Rolling Burritos	burrito
Diamonds in the Rough	diamond
Wheel of Fortune Cookie Monster Mash	wheel
Chutes and Ladders	ladders

Call your program file: *filename*.c, *filename*.cpp, *filename*.java, or *filename*.py

For example, if you are solving Seahorse Shoes:
Call your program file: shoes.c, shoes.cpp, shoes.java, or shoes.py
Call your Java class: shoes

Naughty or Nice?

Filename: `naughty`

It is widely known that Santa Claus delivers presents to houses all over the world in December. However, not many people know what he does during the rest of the year. You see, Santa enjoys a jolly good prank, and he'll often find himself pranking entire cities when he's not busy making or delivering toys.

One place that he particularly enjoys pranking is the South Pole. There are h houses in the South Pole, which are conveniently numbered from 1 to h. Whenever Santa is in the mood for pranking, he'll start at a particular house, s. He rings the doorbell, then runs to another house. He always visits houses in order, but just so he doesn't arouse suspicion, he only knocks on every k^{th} house (for example, every 2^{nd} or 3^{rd} house), where $k \geq 2$. This is because the residents (who are penguins) would know that something's up if one or both of their next door neighbors were pranked as well. Once Santa pranks p houses that day (or passes house number h), he goes back home to the North Pole.

After pranking houses for d days, though, Santa feels bad for pranking the penguins so many times. To make things right, he decides that next Christmas he will award the biggest and best gift to the penguin who got pranked the most. If there are multiple houses that got pranked the most, he will award the gift to the penguin in the lowest numbered house.

The Problem:

Help Santa by telling him which house should receive the biggest and best gift the following Christmas. Santa, of course, wants to make amends each and every year so he needs your suggestion for each one.

The Input:

The first line contains a single, positive integer, y, representing the number of years that Santa goes to the South Pole to prank. The first line of each year contains two integers, h and d ($1 \leq h \leq 100{,}000$; $1 \leq d \leq 100{,}000$), representing the number of houses and the number of days that Santa pranked the South Pole that year, respectively. The next d lines each contain three integers, s, k, and p ($1 \leq s \leq h$; $2 \leq k \leq 10$; $1 \leq p \leq h$), denoting the starting house, the number representing his frequency of visiting houses (he skips k-1 houses), and the maximum number of houses he pranks that day. Note that he will visit p houses unless he passes the very last house.

The Output:

For each year, output a single line:

```
House g should get the biggest and best gift next Christmas.
```

where g is the house number that should receive the gift for that year.

Sample Input:

```
2
4 3
1 2 1
3 4 2
2 2 2
10 2
2 3 2
3 2 5
```

Sample Output:

```
House 1 should get the biggest and best gift next Christmas.
House 5 should get the biggest and best gift next Christmas.
```

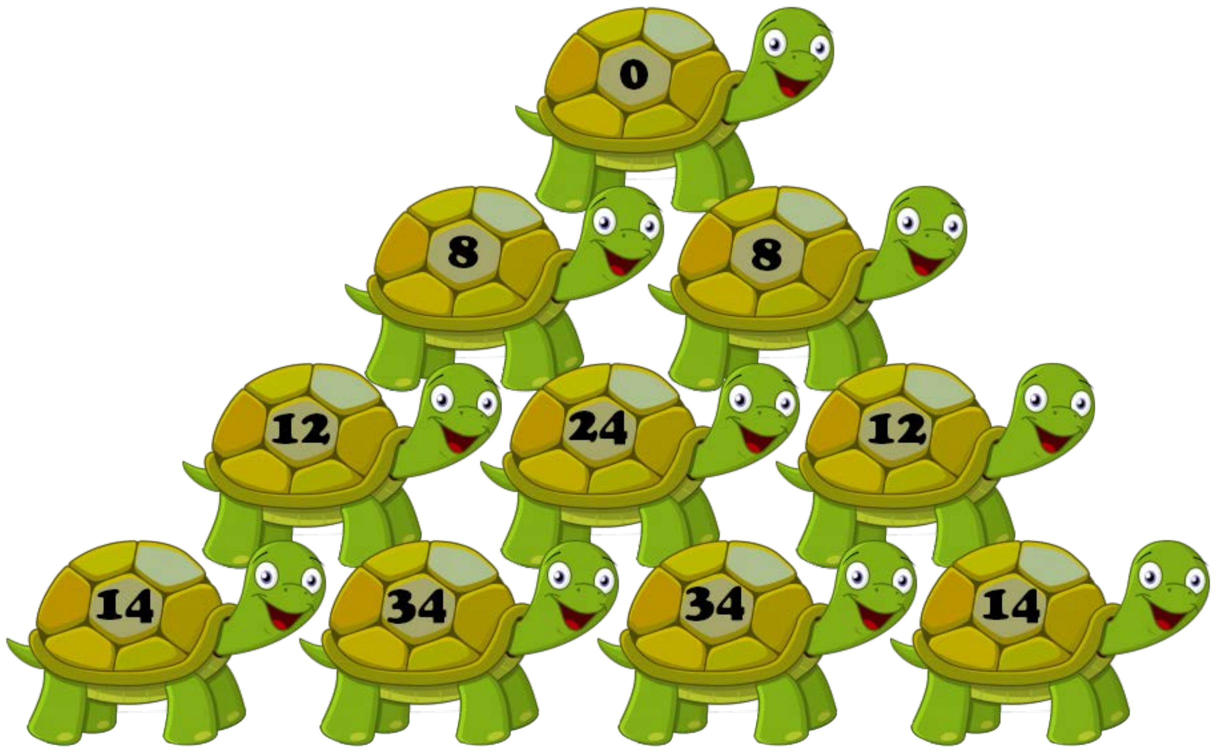
Yertle the Turtle

Filename: `yertle`

Yertle the Turtle is the king of his pond. As king, he enjoys being up high in the sky so he can rule over more land. One day, he ordered the turtles in his pond to stack themselves on top of each other so Yertle could climb up the tower and be king of the clouds. Unfortunately, there was a huge mishap that resulted in the king falling in the mud! After the turtles suggested they all work together towards a better solution (and Yertle relented), they realized that turtles shouldn't be stacked in a vertical line. Instead, they should be stacked in a pyramid to take some of the weight off of the lower turtles' backs. Yes, this is the answer!

Yertle knows how much weight each turtle in his pond can support on their backs and he knows that every turtle weighs *w* pounds. He wants to know how high he can build a pyramid of turtles. Yertle will be at the top. The row below him will have 2 turtles, then 3 turtles, and so on. Each turtle below Yertle has to support half of the weight of the turtle or pair of turtles directly above it plus half of the weight *those* turtles support. Of course, Yertle doesn't support any weight because he's on top.

Here is an example of a pyramid of height 4 which shows the minimum weight each turtle must support if all of them weigh 16 pounds. The turtles in the second row from the top must support 8 pounds each because they each support half of Yertle's weight. In the next row, the turtle in the center supports 24 pounds because the two turtles directly above him weigh a total of 32 pounds and they are supporting a total of 16 pounds and 32/2 + 16/2 = 24.



The Problem:

Help Yertle by telling him how high he can build his pyramid without exceeding the weight limit of any turtle. The turtles can be arranged in any order and some turtles may not be used.

The Input:

The first line contains a single, positive integer, *p*, representing the number of ponds (Yertle visits many different ponds to find the largest turtle population). For each pond, two integers will be given: *n* and *w* ($1 \leq n \leq 10{,}000$; $1 \leq w \leq 100$), denoting the total number of turtles in the pond, including Yertle, and the weight of each turtle, respectively. The next line will contain *n*-1 integers, s_i ($1 \leq s_i \leq 10{,}000$), representing the amount of weight in pounds that the *i*th turtle can support (note that if *n* is 1, this line will be empty but will still appear). Yertle is not included because he doesn't support any weight.

The Output:

For each pond, output a single line beginning with "`Pond #p: `" where *p* is the pond number (starting with 1). Immediately following this, if the height of the pyramid is at least 2, output "`The pyramid is t turtles high!`" where *t* is the height of the tallest possible pyramid of turtles. Otherwise, if Yertle cannot build a pyramid, output "`Poor Yertle.`"

Sample Input:

```
3
10 16
8 24 8 14 12 12 34 14 34
5 1
5 4 1 9
9 10
1 1 1 1 1 1 1 1
```

Sample Output:

```
Pond #1: The pyramid is 4 turtles high!
Pond #2: The pyramid is 2 turtles high!
Pond #3: Poor Yertle.
```

The Power of Two is a Curious Thing

Filename: `power`

"*The power of ~~love~~ two is a curious thing; Make a one man weep, make another man sing*"

The number 31 is an interesting number in that it is exactly one less than an integer power of two! Specifically, $2^5 - 1 = 31$. The number 31 is also prime, and furthermore is an example of a Mersenne Prime, which is a prime number that can be written in the form of $2^n - 1$ for some integer n.

In our study of Mersenne Primes, we first need to compute various values of $2^n - 1$ (we will save the actual checking if they are prime for later). This is where you come in!

The Problem:

Given an integer value for n, determine the value of $2^n - 1$.

The Input:

The first line will contain a single, positive integer, c, representing the number of values of n to be processed. Following this, each of the next c lines will each contain a single, positive integer, n ($1 \le n \le 30$).

The Output:

For each integer, n, to be processed, output the value $2^n - 1$ on its own line.

Sample Input:

```
3
3
6
14
```

Sample Output:

```
7
63
16383
```

Seahorse Shoes

Filename: `shoes`

Mr. Hippocampus is currently pregnant with his and his wife's baby seahorses (recall that male seahorses are the ones that carry the developing babies). Nurse Shark has recently done an ultrasound on Mr. Hippocampus and has informed Mr. and Mrs. Hippocampus exactly how many baby seahorses they will be having. Furthermore, through the help of Delphinapterus, the Beluga (and his sonar abilities), Mr. and Mrs. Hippocampus know in advance what shoe size each of their baby seahorses will have. Thus, the newly expecting parents are preparing for their coming young and are buying all of the seahorse shoes they will need.

Seahorses, as you know, only need one shoe for their tail. However, Cobbler the Crab only sells shoes in pairs. Mr. and Mrs. Hippocampus want to get shoes for all of their baby seahorses but they want to spend the least amount of money possible. Furthermore, they do not want their children to be too uncomfortable. Thus, if a baby seahorse has a certain shoe size, he/she will only be comfortable wearing a shoe of either that size or one which is exactly one size too big or one size too small. Formally, if a baby seahorse has a shoe size of *x*, he/she will be comfortable wearing either a shoe of size *x, (x + 1),* or *(x - 1)*. Cobbler the Crab sells shoes in all positive integer sizes and each pair of shoes costs the same amount of money. However, Cobbler the Crab only sells shoes in pairs and, thus, will *not* sell single shoes. Thus, Mr. and Mrs. Hippocampus may have extra shoes because they can only buy them in pairs. However, they are okay with this as long as they buy the least number of pairs of shoes possible while still assuring that each baby seahorse gets a shoe which will be comfortable for him/her to wear.

The Problem:

Given the number of baby seahorses Mr. and Mrs. Hippocampus are having and all of their respective shoe sizes, calculate the minimum number of pairs of shoes that Mr. and Mrs. Hippocampus need to purchase to guarantee that each baby seahorse gets a shoe that he/she can wear comfortably (if it is either is his/her size, or is one size too big or too small).

The Input:

The first line of input will contain a single, positive integer, *t*, representing the number of litters to follow. Following this, each litter will be contained on two lines. The first line for each litter will contain one positive integer, *n* ($n \leq 2{,}500$), representing the number of baby seahorses Mr. and Mrs. Hippocampus will be having. The next line contains *n* positive integers representing the shoe sizes of each baby seahorse. No baby seahorse will have a shoe size which is larger than 1,000.

The Output:

For each litter, output a line containing "`Litter #`*`x`*`: `*`m`*" where *x* is the litter number in the order given in the input (starting with 1) and *m* is the minimum number of pairs of shoes that Mr. and Mrs. Hippocampus need to buy for that litter.

Sample Input:

```
2
5
2 3 1 1 1
10
4 1 9 4 1 8 9 1 8 4
```

Sample Output:

```
Litter #1: 3
Litter #2: 6
```

Every Day I'm Shuffling

Filename: `shuffling`

There is a party rock in the house tonight, so naturally everybody is having a good time. But sitting in the corner, Redfoo and Sky Blu are shuffling their deck of cards and simply proclaiming, "Every day I'm shuffling."

Redfoo and Sky Blu have a new deck of n cards which originally are in order where the ith card from the top has the number i written on it (1, 2, …, n-1, n). Just as their song is repetitive, so are Redfoo and Sky Blu's shuffling efforts. They have been meticulously shuffling their deck once a day for many days now. The process they use is simple. They deal out the n cards from left to right from the top, maintaining the order of the cards. The leftmost card (originally the top card) has position 1, while the card on the right has position n. They then refer to their special permutation of the numbers 1 through n (p_1, p_2, …, p_{n-1}, p_n), which tells them in what order they pick up each card. The ith card they pick up should be the one in position p_i. Each card they pick up is put on the bottom of the growing pile. Picking up a card does not affect the position numbers of all the remaining cards still laid out. For example, a deck of [2, 1, 3, 4] (from top to bottom) with a special permutation of [2, 3, 1, 4] will result in a new deck in the order [1, 3, 2, 4] (from top to bottom). They then repeat this process with the new order of cards the next day, still using the same special permutation from day 1.

Unfortunately, after many, many days of shuffling their cards, Redfoo and Sky Blu have noticed that the numbers on each card have faded away. They wish to return their cards to their original brilliance but would not like to renumber any cards. Thus, they need your help to determine what number should be on each card given the special permutation they use and the number of days they have shuffled their deck.

The Problem:

Given the special permutation that Redfoo and Sky Blu use to shuffle their special deck of cards every day and the number of days, print the deck from top to bottom after they have shuffled it for that many days.

The Input:

The input consists of multiple shuffling sprees. The first line of the input will contain a single, positive integer, t, the number of sprees that follow. Each spree will consist of two lines. The first will contain two integers, n_i and k_i ($1 \le n_i \le 100{,}000$; $0 \le k_i \le 10^9$), representing the number of cards in Redfoo and Sky Blu's special deck of cards, and the number of days they have been shuffling their deck, respectively. The second line will contain n_i integers, the special permutation of the numbers 1 to n_i which represents the pattern used to shuffle the deck on that particular spree.

The Output:

For each shuffling spree, output the n_i integers from 1 to n_i in the order of the deck after it has been shuffled for the k_i days in that spree using the appropriate special permutation. The jth number on each line should represent the number written on the jth card from the top.

Sample Input:

```
2
4 2
2 3 1 4
5 10
4 3 5 1 2
```

Sample Output:

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

Knights, Pirates, Ninjas

Filename: `knights`

Ankit is a high school student who is inspired by numbers and the digits of our decimal system. He is fascinated that the digit 0 (zero) can be placed before any number (or after its decimal point) without changing its value. It is like there are always zeroes there, even if you can't see them! Of course, a zero will always appear within a number when it is significant. Ankit has realized he never thinks about zeroes until he sees one, and then it always reminds him of ninjas—they can be invisible, but when you see one, it really matters!

Certain other digits in our decimal system have a different property. When you double their value, you have to carry a 1 into the next higher digit position. For example, when you double the number 92, doubling the 2 results in a 4 (which is no extra work, so 2 does not have this property) but then doubling the 9 results in 18, so you must carry the 1 into the hundreds place (so 9 has this property). All of the digits that have this property of "carry when doubling" tend to make Ankit think of pirates because, as everyone knows, pirates carry doubloons.

Finally, when Ankit sees the digit 1, it always brings to mind the UCF Knights, because UCF is his first choice for a college education. He plans to major in Computer Science and try out for the UCF Programming Team!

The Problem:

Given an integer, output whether its digits will make Ankit think of knights, pirates, and/or ninjas.

The Input:

The input will begin with a single, positive integer, t, representing the number of integers Ankit is investigating. On each of the following t lines, there will be a single, positive integer, n ($n \leq 2{,}000{,}000{,}000$), representing an integer to determine whether it makes Ankit think of knights, pirates, and/or ninjas. The integer will not have leading zeroes—even though we know they *could* be there.

The Output:

For each integer, first output the number. On the same line, output each of the words "knights", "pirates", and "ninjas"—always keeping that relative order—according to whether any of the digits would ever make Ankit think of each of those things (output each word just once if appropriate for that number). Separate the words from each other, and from the number, using one space. If none of the digits cause Ankit to think of any of them, output just the number. Trailing spaces at the end of the line are okay.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
4
910
1000
23
92
```

Sample Output:

```
910 knights pirates ninjas
1000 knights ninjas
23
92 pirates
```

Shifting Gears

Filename: `shift`

After typing up an essay for school, you want to know how many times you pressed the shift key. You know that you didn't use the Caps Lock key at all and you held down the shift key when it was needed for multiple characters in a row. You never held it down when you didn't need to, such as when you pressed on the space bar.

The Problem:

Determine how many times the shift key was pressed to type each of the given lines of text. Assume that only characters on a standard US keyboard layout were used.

The Input:

The first line contains a single, positive integer, *p*, representing the number of paragraphs. Each paragraph will contain a single line of text containing no more than 100 characters. It will not start or end with white space. However, it may contain numbers and letters (lower case and upper case) as well as any of the punctuation symbols listed below. Note that upper case letters require the shift key, while lower case letters (and the spacebar) do not.

The following symbols use the shift key: `~ ! @ # $ % ^ & * ( ) _ + { } | : " < > ?`

The following symbols don't use the shift key: `` ` [ ] \ ; ' , . / ``

The Output:

For each paragraph, output the single line “`The shift key was pressed` *k* `times.`” where *k* is the number of times the shift key was pressed.

Sample Input:

```
4
AbC
Hello world
Welcome to the UCF High School Programming Tournament
***Don't forget about "punctuation"?!?!
```

Sample Output:

```
The shift key was pressed 2 times.
The shift key was pressed 1 times.
The shift key was pressed 6 times.
The shift key was pressed 3 times.
```

Jumping Fish

Filename: `jump`

In the popular movie, *Finding Dory*, as Hank and Dory are speeding down the highway in a truck, they take some sharp turns and drive pretty crazily. In the back of the truck are a bunch of fish tanks arranged in a rectangular grid. When Hank and Dory suddenly accelerate or decelerate, this causes the fish to jump between tanks. For example, if they suddenly stop, each (and every) fish will jump to the tank directly in front of it. Similarly, if the truck makes a sudden turn to the left or right, the fish will jump between tanks either to the right or left, respectively.

There are walls on all four sides of the grid of tanks. Thus, each sudden movement will cause all of the fish to move one tank over either forwards, backwards, left, or right, and any fish that don't have a tank to move into in that direction will hit the wall and fall back into the same tank. Luckily, both Hank and Dory are safe in the cab of the truck so they will never fall back into any of the tanks.

The Problem:

Given a description of the tanks in the back of Hank and Dory's stolen truck and a list of the directions their crazy driving will cause the fish to move, determine how many tanks have fish in them after each sudden movement. The tanks in the truck are in a rectangular arrangement. Each tank initially contains at least one fish.

The Input:

The first line of input contains a single, positive integer, *t*, representing the number of trips to follow. Each trip will be contained on multiple lines. The first line of each trip will contain three positive integers, *x*, *y* and *n* ($x \leq 10{,}000$; $y \leq 10{,}000$; $n \leq 10{,}000$), where *x* represents the number of tanks in the left-right direction, *y* represents the number of tanks in the forward-backward direction and *n* represents the number of sudden movements to follow. The next *n* lines each contains a single letter: either '`F`', '`B`', '`L`', or '`R`' representing the direction that the sudden movement will cause the fish to move between tanks (representing, Forwards, Backwards, Left or Right, respectively).

The Output:

For each trip, output a line "`Trip #i:`" where *i* is the trip number in the order given in the input (starting at 1). For each trip, output *n* integers, each on their own line, representing the number of tanks that contain fish after *each* sudden movement in the same order as given in the input. Output a blank line after the output for each trip.

Sample Input:

```
2
3 3 5
L
L
F
F
B
3 3 4
L
R
R
L
```

Sample Output:

```
Trip #1:
6
3
2
1
1

Trip #2:
6
6
3
3
```

Rolling Burritos

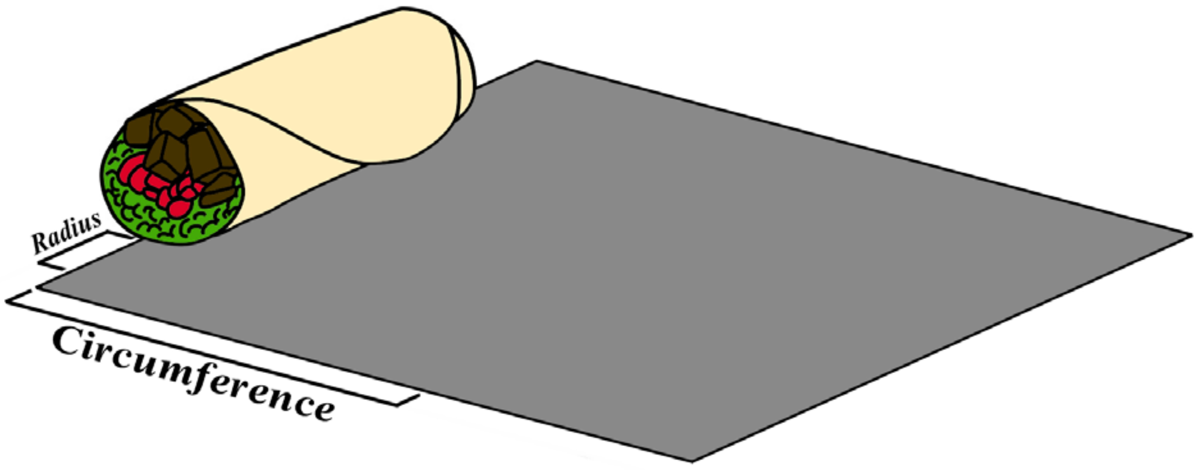
Filename: `burrito`

Everyone knows that Ali loves circles, but sometimes he needs to stop admiring his circles and take a food break. Ali tends to eat the burritos from the Unincorporated Center of Fast-food, as they are one of the most delicious circular foods on this planet. But sometimes his burritos don't fit in the foil wrappers, leading to a disappointingly cold burrito.

As everyone knows, burritos are cylindrical which means they have a radius, length, circumference, and volume. A burrito can be completely wrapped if the rectangular foil is at least as long or as wide as the circumference, and the excess foil at each end of the burrito is at least its radius. The foil may not be able to wrap the burrito in its original orientation, so Ali will suggest that it can be rotated 90 degrees to ensure the burrito is warm.

$$volume = \pi * radius^2 * length$$

$$\pi = 3.141592653589793$$



Above is an example of a burrito that can be completely wrapped.

The Problem:

Given the volume and radius of a burrito, and the width and height of the foil, determine whether or not the burrito will fit and stay warm. Use 3.141592653589793 as the value of π.

The Input:

The first line of the input is a single, positive integer, *n*, representing the number of burritos in the input to check. Each burrito and foil are described in a pair of lines. The first line describes the burrito and will have a line containing two integers: the burrito's volume, v ($0 < v \leq 100$), and its radius, r ($0 < r \leq 20$), respectively. The second line for each burrito will denote the foil and will contain two integers: the foil's width, w ($0 < w \leq 50$), and its length, l ($0 < l \leq 50$), respectively. The burrito's volume is given in cubic inches where as the burrito's radius and the foil's dimensions are given in inches.

The Output:

For each burrito, first output "`Burrito #`*`i`*`: `" where *i* is the number of the burrito in the input (starting with 1), and then immediately follow this with either:

```
Don’t worry, the burrito fits!
```

if the burrito will fit in the foil, or:

```
Looks like a cold burrito today.
```

if it will not.

Sample Input:

```
2
5 20
1 1
2 1
50 50
```

Sample Output:

```
Burrito #1: Looks like a cold burrito today.
Burrito #2: Don’t worry, the burrito fits!
```

Diamonds in the Rough

Filename: `diamond`

Far off in Ascii Land a new high tech diamond mining company is looking to obsolete their competition. In Ascii Land diamonds are made out of forward slashes and backslashes, but they must be found in the correct symmetrical configuration. Informally, the slashes must make a '`V`'-shape with an equal size '`Λ`'-shape on top of it. In the past, after a slab of text was mined and cleaned of all unnecessary character types, the extraneous slashes needed to be removed by hand. This new mining company plans to automate the process.

Diamonds contain no less than two forward slashes and two backslashes and always contain an equal number of forward slashes and backslashes. For example:

```
../\............/\......
..\/.../\....../..\.....
....../..\..../././\.\....
......\../....\.\/./....
.......\/......\../.....
................\/......
```

Four valid diamonds

```
../\..../\....../\......
.../.../..\..../..\.....
....../....\./....\.....
......\..../..\..../....
........\/.....\../.....
................\/......
```

No valid diamonds

Diamonds are capable of intersecting, overlapping, and containing other diamonds (as shown above; other examples are shown in the Sample Input).

The Problem:

Given a slab of text containing characters '`.`'(period), '`\`' (backslash), and '`/`' (forward slash), output the same slab of text with any forward slash or backslash that is *not* a part of any valid diamond replaced with a period.

The Input:

The first line of the input contains a single, positive integer, *s*, representing the number of slabs in the input. This is followed by *s* descriptions of slabs. Each description starts with a line containing two integers, *h* and *w* ($1 \le h \le 50$; $1 \le w \le 50$), representing the height and width of the slab, respectively. On each the following *h* lines, there will be *w* characters, depicting the slab. All slab characters are only '`.`' (period), '`\`' (backslash), or '`/`' (forward slash).

The Output:

For each slab, output a line of the form "`Slab #i:`" where *i* represents the number of the group you are processing (starting from 1) followed by *h* lines of width *w* containing the depiction of the slab with all extraneous slashes removed (replaced by a '`.`').

Sample Input:

```
3
2 8
\//\\//\
\/\//\/\
6 8
/.//\\.\
.///\\\.
////\\\\
\\//\\//
.\\/\//.
\.\\//./
10 24
..///../..\/\\\\..../...
.....\/./\.\.../\../....
..\./.\/./\/../\.\/.....
./.\...\/\/\./\\\/...../
../\/\..\/.././\.\..../.
././\.\./..//\\/./\../..
.\.\/./\...\\//\.\/./...
..\/\/..\...\.\/./\/....
.././........\.././.....
././......\...\/\/......
```

Sample Output:

```
Slab #1:
../\....
..\/....
Slab #2:
.../\...
../..\..
././\.\.
.\/..\/.
..\/\/..
...\/...
Slab #3:
........................
......../\..............
......./././\.../\........
.......\/\/../..\.......
../\/\..\/.././\.\......
././\.\....//\\/./\.....
.\.\/./....\\//\.\/.....
..\/\/......\.\/./......
.............\../.......
..............\/........
```

Wheel of Fortune Cookie Monster Mash

Filename: `wheel`

Ali has recently been appointed to produce a segment on the Wheel of Fortune. Specifically, the *Before and After* section of the show, where contestants must join together two phrases. In order to join two phrases, the last word in first phrase must match the first word in the second. For instance, "WHEEL OF FORTUNE" and "FORTUNE COOKIE" becomes "WHEEL OF FORTUNE COOKIE". Since the joining would not be apparent otherwise, phrases will always have more than one word. In addition, a phrase will never start and end with the same word.

As he prepares for the next segment, he wonders: what is the longest possible such sequence he could generate if he allowed as many phrases to be joined together as possible? Ali would like to have super long puzzles, if possible, so he will consider many, many phrases within each single puzzle.

The Problem:

Given a list of phrases, determine the longest chain of phrases that can be constructed such that each phrase is linked to the next by one word.

Ali gets bored when he sees the same phrases over and over again, so he will ensure that there will be no possible cycles in the given words. For example, "MONSTER COOKIE" and "COOKIE MONSTER" will never appear in the same input. Indeed, after a phrase has been used once, there will never be another opportunity to use it again in the same chaining of phrases.

The Input:

The input will begin with a single, positive integer, *t*, representing the number of puzzles. Each puzzle will begin with a single integer, n ($1 \le n \le 100{,}000$), representing the number of phrases in that puzzle. This will be followed by n lines, each with a string p (whose length is at least 1 and at most 50), made up of only the letters A to Z and spaces. No line will start or end with a space.

The Output:

For each puzzle, first output "`Puzzle #i: `" where *i* is the number of the puzzle in the input (starting at 1). Immediately follow this with the maximum number of phrases that may be joined into a single chain for that puzzle.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
5
WHEEL OF FORTUNE
FORTUNE COOKIE
COOKIE MONSTER
MONSTER MASH
NOT ME
2
ORDER CAN BE CHANGED
THEY ARE NOT GIVEN IN ORDER
```

Sample Output:

```
Puzzle #1: 4
Puzzle #2: 2
```

Chutes and Ladders

Filename: `ladders`

Oh no, Matthew has been trapped in Chutes and Ladders World! He needs your help to determine if he can escape.

Chutes and Ladders World is like the popular game, but in 3 dimensions! Chutes and Ladders World is composed of levels (planes), and each level is made of squares. When Matthew moves within a level, he can move horizontally in all four cardinal directions (not diagonally), but cannot normally move vertically between levels (for that, he needs to use a chute or a ladder!).

However, if he enters a chute, he will fall down to the end of the chute, and then remain in the bottom of the chute until he moves out of it (the bottom of the chute being the square containing the chute, on the lowest level the chute reaches). Additionally, he can enter a chute from any level that the chute is present on, not just from the start of the chute. However, he cannot bypass chutes – if he moves onto a square with a chute, he will enter that chute.

Furthermore, if he moves onto a ladder, he can then move up or down that ladder. While on a ladder, he can get off at any level that ladder reaches, including the level that he entered the ladder on. He does not have to go all the way up or all the way down while on a ladder.

Due to the physics of Chutes and Ladders World, chutes and ladders are always only perfectly vertical. Additionally, both chutes and ladders will always cover at least two levels and be contiguous. If there is a level that splits two chutes or two ladders, these chutes or ladders are obviously not connected, even if one is above another. However, a ladder can otherwise be directly above a chute, and vice versa.

The Problem:

Given a map of Chutes and Ladders World, determine if Matthew can make it to the exit.

The Input:

The first line of the input will be a single, positive integer, *n*, representing the number of maps to analyze. Following this, each map will be defined across multiple lines. For each of these maps, the first line will contain three positive integers, *l*, *w* and *h* ($1 \le l \le 50$; $1 \le w \le 50$; $1 \le h \le 50$), each separated by a single space, representing the length, width, and height of the map, respectively. Following this, there will be *h* slices of the map (each slice is represented by *l* lines of *w* consecutive characters) with the first slice representing the highest point of the map, and each slice following one level lower. Each element will be a character: '`.`' for an empty space, '`#`' for a ladder, '`*`' for a chute, '`S`' for Matthew's starting point, and '`E`' for the exit point. There will be exactly one '`S`' and one '`E`' per map.

The Output:

For each map, output a single line containing “`Map #i: `” where *i* represents the number of the map being processed from the input (starting with 1), followed immediately by either “`Yes`” if Matthew can escape or “`No`” if he cannot.

Sample Input:

```
2
3 3 3
S..
.*.
...
...
.*E
...
...
.*.
...
3 3 3
...
#.E
...
...
#..
...
...
#.S
...
```

Sample Output:

```
Map #1: No
Map #2: Yes
```