2018 Consortium for Computing Sciences in Colleges
Programming Contest
Saturday, November 3rd
Roanoke College
Roanoke, VA

There are nine (9) problems in this packet. Each team member should have a copy of the problems. These problems are NOT necessarily sorted by difficulty. You may solve them in any order.

Remember input/output for the contest will be from `stdin` to `stdout`. `Stderr` will be ignored. Do not refer to or use external files in your source code. Extra white space at the end of lines is ignored, but extra white space at the beginning or within text on a line is not ignored. Have a lot of fun and good luck! ☺

Problem 1. password strength

Problem 2. Echoes

Problem 3. Silly Math

Problem 4. Pumpkinfest

Problem 5. Star Wars Compliance

Problem 6. Equatorial Real Estate

Problem 7. Sensors

Problem 8. Latin Squares

Problem 9. Networking

Problem 1

Your new online banking app has five rules for a valid, strong password. Write a program to determine whether a given password is valid according to these five rules. Report any rules violated on an invalid password.

Rule 1. The minimum length is 6 and the maximum length is 20.
Rule 2. At least one uppercase and lowercase letter is used.
Rule 3. At least one digit is used.
Rule 4. At least one printable symbol other than $ is used.
Rule 5. There cannot be any blank spaces.

Input
The first line of input will contain a single integer *n*, which will indicate the number of test cases. You may assume that $1 \leq n \leq 100$. This will be followed by *n* strings, one per line, for checking validity of as a password. Each input string may contain letters, digits, printable symbols, and blank spaces.

Output
For each test case, your program needs to print one of the following statements:

- `Password <number>: is valid`
- `Password <number>: is invalid by rule(s) <x>`

The `<number>` is number of the test case and `<x>` is rule number(s) violated. Test case numbers begin at 1. The output should be in the exact format seen below with a single space after any rule numbers violated. If more than one rule number is violated, they should be printed in ascending order by rule number.

Sample Input

```
7
Ha$
happycode2018
1,000,000
Roanokecollege1842
Is it strong?
UsingJQueryAndCss2
C++JavaPython3
```

Output Corresponding to Sample Input

```
Password 1 is invalid by rule(s) 1 3 4
Password 2 is invalid by rule(s) 2 4
Password 3 is invalid by rule(s) 2
Password 4 is invalid by rule(s) 4
Password 5 is invalid by rule(s) 3 5
Password 6 is invalid by rule(s) 4
Password 7 is valid
```

Problem 2

Echoes

A monastery in a cave hidden high in a mountain in the Himalayas is home to a set of mystical bells. When a bell is rung, its clear, sweet tone echoes all through the cave. Each echo produces another tone, which echoes causing more tones, all sounding together. We would like to determine, after one or more bells are rung, how many different tones can be heard at any particular moment in time. Let t be the current time, in seconds. At the instant the bell(s) is/are rung, $t=0$. Let b be the number of bells that are rung, $b > 0$. Let s be the number of seconds that a tone lasts until it can no longer be heard, $s > 1$. Assume that at each time step, all audible tones cause one echo (additional tone). For example, for b = 3 bells, s = 3 seconds:

t	# tones	explanation
0	3	3 bells rung at start
1	6	3 tones echo, doubling total
2	12	6 tones echo, doubling total
3	18	original 3 tones finish, remaining 9 echo
4	30	3 new tones from $t=1$ finish, remaining 15 echo
5	48	6 new tones from $t=2$ finish, remaining 24 echo
6	78	9 new tones from $t=3$ finish, remaining 39 echo

Input

The first line of input will contain a single integer c, which will indicate the number of test cases. You may assume that $1 \leq c \leq 100$. This is followed by c lines consisting of three numbers separated by spaces of the form n b s, where n is the number of total seconds, b is the number of bells rung, and s is the number of seconds that a tone is audible. The three integers n, b, and s will all be positive and ≤ 50.

Output

The output should be a number representing the number of tones audible after n seconds. The output for each case should be on its own line.

Sample Input

```
2
6 3 3
8 2 4
```

Output Corresponding to Sample Input

```
78
324
```

Problem 3

Silly Math

There are many different proofs that 1 = 2. However, sometimes, one sees more spectacular proofs such as 2/10 = 2/1. The following "proof" was passed on recently as a demonstration that some people know just enough math ideas to be dangerous. For example:

$$\frac{2}{10} = \frac{two}{ten} = \frac{wo}{en} = \frac{23+15}{5+14} = \frac{38}{19} = \frac{2}{1}$$

where the *t*'s are cancelled in one step and the letters are changed into numbers based on their position in the alphabet. Note: if all the letters, in the numerator or denominator are cancelled using this rule, then the string encodes to 0. Non-letters such as hyphens are ignored. You are to determine if two pairs of numbers are equal using this silly math.

Input

The first line of input will contain a single integer *n*, which will indicate the number of test cases. You may assume that $1 \le n \le 250$. This is followed by *n* lines consisting of four integers *a*, *b*, *c*, and *d*. The four integers will all be nonnegative and < 100. The expression *a*/*b* is to be manipulated using the format given above.

The resulting silly expression, call it *a'*/*b'*, equals *c*/*d* provided *da' = cb'*. The conversion is only to be applied to the first expression *a*/*b*.

Output

Use the string "~==~" to indicate silly equality and "~=!!=~" to indicate silly inequality.

Sample Input

```
3
2 10 2 1
2 10 1 2
99 90 1 0
```

Output Corresponding to Sample Input

```
2/10 ~==~ 2/1
2/10 ~=!!=~ 1/2
99/90 ~==~ 1/0
```

Problem 4

Pumpkinfest

The State Fair of Virginia in Caroline County is holding a Pumpkinfest contest for the single largest pumpkin patch. They are hoping you can help them use satellite imagery to help determine the winner of this year's contest. It will take an image of any sized garden filled with both squash and pumpkins, and calculate the size of the largest patch. For example, suppose we have this 6 x 6 matrix where `p` represents a pumpkin and `s` represents a squash. It contains three pumpkin patches sized 1, 3, and 9.

s	s	s	s	s	p
s	p	s	s	s	s
s	p	s	s	p	p
s	p	s	s	s	p
s	p	s	s	s	s
s	p	p	p	p	p

Input

The first line of input will contain a single integer *n*, ($1 \leq n \leq 100$), which represents the number of gardens. The first line of input for each garden will be the dimensions of the garden, *r*, the number of rows, and *c*, the number of columns, where $1 \leq r \leq 40$ and $1 \leq c \leq 40$. Following the dimensions will be *r* lines with *c* characters on each line. Each character will be a lowercase `p` for pumpkin or `s` for squash. You may assume there always be at least one pumpkin in a garden.

Output

For each garden, output the garden number and the size of the largest patch in the format below.

Sample Input

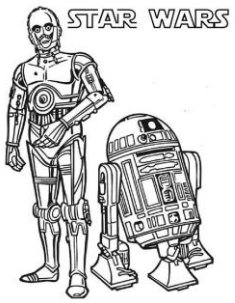
```
3
6 6
sssssp
spssss
spsspp
spsssp
spssss
sppppp
10 10
psssssssp
psspsssssss
pppsssssss
sssssssss
sssppssss
sssppssps
sssssssps
sssssssps
sssspssps
sssppppps
3 4
pppp
pppp
pppp
```

Output Corresponding to Sample Input

```
Garden #1: largest pumpkin patch size is 9
Garden #2: largest pumpkin patch size is 10
Garden #3: largest pumpkin patch size is 12
```

Problem 5

Star Wars Compliance



The Disney imagineers at Walt Disney World in FL and Disneyland in CA are preparing for the opening of next year's newest land: Star Wars: Galaxy's Edge. It will be filled with new attractions and restaurants at both resorts. They have decided to make sure R2D2 and C3PO are hidden in the language of their secret menu at the new Mos Eisley cantina, a dimly-lit watering hole. The secret menu is available only by request. Their secret menu has several hidden sentences that are *Stars Wars Compliant* meaning they should contain all five of the attributes below.

1. The letter R should be repeated exactly two times in the sentence.
2. The letter D should be repeated exactly two times in the sentence.
3. The letter C should be repeated exactly three times in the sentence.
4. The sentence should contain exactly one letter P and one letter O.
5. No word can be duplicated anywhere else in the sentence regardless of case.

The sentence "Darth Vader has step coccus" is an example of a sentence that is Stars Wars Complaint since all five attributes are true. The sentence "Luke was accidentally called Paul over and over" is not complaint since it violates rules 4 and 5.

Input

The first line of input will contain a single integer *n*, indicating number of test cases. You may assume that $1 \leq n \leq 100$. This will be followed by *n* sentences, one per line, for checking for Star Wars Compliance. Each sentence will be made up of one or more words separated by a single space. Each word will contain letters only and be of length ≥ 1. Case does not matter.

Output

For each test case, your program needs to print the following statement:

- `Sentence <number> <is` or `is not> Star Wars Compliant.`

Test case numbers begin at 1. The output should be in the exact format seen below.

Sample Input

```
5
The force is with me when drinking cappuccino today
Jedi beholds pure accuracy
The Jedi beholds the pure accuracy
Dear Mrs. Leia, we saw u an accomplice that day
DARTH Vader HAS STEP COCCUS
```

Output Corresponding to Sample Input

```
Sentence 1 is not Star Wars Compliant.
Sentence 2 is Star Wars Compliant.
Sentence 3 is not Star Wars Compliant.
Sentence 4 is Star Wars Compliant.
Sentence 5 is Star Wars Compliant.
```

Problem 6

Equatorial Real Estate

Luke has a passion for investing in real estate. And to get away from the cold weather, he's decided to find some place hot … the equator! Through the help of a real estate agent, he has found property listings for parcels of land in the equatorial territory of New Albion.
In New Albion, all vacant land parcels are in the shape of triangles. Luke has found a list of parcels that lie near the equator, and he would like to know how big they are.

You will write a program to help Luke compute their areas. More importantly, he is interested in finding land parcels that have substantial amount of land on either side of the equator. So, instead of simply calculating the total area of the triangle, Luke really wants you to calculate two things:

- The area of that part of the triangle north of the equator
- The area of that part of the triangle south of the equator

New Albion uses the traditional English system of units. In its real estate law, the standard unit of distance is the chain, which equals 66 feet. According to the geodetic datum used in New Albion, we assume that the earth is flat. Each parcel of land is defined by three ordered pairs, (x_1, y_1), (x_2, y_2), and (x_3, y_3), representing the vertices of the triangle.

Within each ordered pair, the value of `x` is the distance from 0 degrees longitude. A negative `x` value means the point is `x` chains west of 0 longitude, and a positive value of `x` means the point is `x` chains east of 0 longitude. Similarly, `y` measures the distance from the equator. A positive value of `y` means that the point is `y` chains north of the equator, and a negative value of `y` means that the point is `y` chains south of the equator.

Input
You may assume that in the input, the three points are ordered such that $y_1 \geq y_2 \geq y_3$.
For your program, the first line of input will contain an integer n, the number of land parcels.

You may assume that $2 \leq n \leq 100$. Each of the next n lines will contain three ordered pairs of the form: `(x1, y1) (x2, y2) (x3, y3).`

Each of the six numbers used in the ordered pairs is a real number, measured in chains.

Output
For each triangle, your program must print a line of output indicating the triangle's number (starting at number 1), followed by the area north of the equator, and the area south of the equator, in square chains.

Note that it is possible for one of these areas to be zero. If all the values of `y` are nonnegative, then the triangle lies entirely in the northern hemisphere. Similarly, if all the values of `y` are nonpositive, then the triangle lies entirely in the southern hemisphere.

The areas must be rounded to the nearest thousandth and displayed to exactly three decimal places. No units are to be printed. The format of the output is:

```
Triangle <integer>:  <real> north, <real> south
```

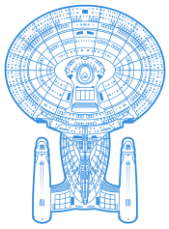
Be sure to print two spaces after the colon, and one space after the comma.

Sample Input
```
3
(7, 2) (2, -4) (11, -5)
(3, 2) (1, 1) (9, -5)
(0.5, -6) (-5, -9) (12, -11.8)
```

Output Corresponding to Sample Input
```
Triangle 1:  2.810 north, 26.690 south
Triangle 2:  4.048 north, 5.952 south
Triangle 3:  0.000 north, 33.200 south
```

Problem 7

Sensors



You have been working on a new deep-space communications system. In particular, you have been focused on a small component used for clocking signals. It measures a regular pulse and uses that pulse to time other system components. However, you have recently noticed that the sensor is faulty *p* percent of the time. It misses an incoming clock signal which will cause it to report a different clock signal rate. For instance, the sensor could report delta time between pulses as *t*, even though it missed two pulses in the middle! In that case, the true time between pulses would be *t / 3*.

Since you do not know exactly how many (or if any at all) pulses occurred between two known pulses, you have to make your best guess knowing the failure probability p of the sensor. For instance, to miss two pulses in a row, but then catch one would occur with probability:

$$p \cdot p \cdot (1 - p)$$

Similarly, to fail *f* times then succeed would occur with probability:

$$p^f \cdot (1 - p)$$

So, you can examine all possible numbers of failures (0 to infinity) and see how likely that is to occur. Moreover, the sensor has taken multiple measurements over time. This can help you because it allows you to take an average time delta over all measurements. However, you will have to keep in mind that a single measurement could represent multiple circumstances that occur with different probabilities such as receiving the pulse with no missed pulses, receiving the pulse after missing one pulse, receiving the pulse after missing two pulses, and so on.

Input

The first line of input will contain a single integer *c*, which will indicate the number of test cases. You may assume that $1 \leq c \leq 100$. This is followed by *c* cases. Each case will begin with two numbers *n* and *p*, the number of samples in this case and the failure rate of this particular sensor. The next *n* lines will each contain a single floating point representing a time between measurements.

Output

For each case, you should output the estimated average time between pulses taking into account all samples, and all possible cases with missed pulses (0 missed pulses, 1 missed pulse, 2, 3, etc.) factoring in how likely they are to occur. Each output will be printed at the start of a new line rounded to thevim l nearest thousandth and displayed to exactly three decimal places.

Sample Input

```
1
3 0.5
2.0
2.0
2.0
```

Output Corresponding to Sample Input

```
1.386
```

Problem 8

Latin Squares

A two-dimensional square array of integers is a *Latin Square* if the following conditions are true.

- The first row has no duplicate values.
- All values in the first row of the square appear in each row of the square.
- All values in the first row of the square appear in each column of the square.

Input

The first line of input will contain a single integer *n*, which represents the number of test cases. You may assume that $1 \leq n \leq 100$. This will be followed by *n* test cases. Each test case consists of a single line with a positive integer *x*, $1 \leq x \leq 100$ representing the number of rows and columns in the matrix followed by its *x* rows of *x* integers. Your input will be in row major order.

Output

For each test case, your program needs to print the following statement:

- `Matrix <number>: <is` or `is not> a Latin Square.`

The `<number>` is the number of the test case. Test case numbers begin at 1. The output should be in the exact format seen below.

Sample Input

```
3
3
1 2 3
2 3 1
3 1 2
4
10 30 20 0
0 20 30 10
30 0 10 20
20 10 0 30
2
1 2
1 2
```

Output Corresponding to Sample Input

```
Matrix 1: is a Latin Square.
Matrix 2: is a Latin Square.
Matrix 3: is not a Latin Square.
```

Problem 9

Networking

NASA has decided to send networking transmitters to various places on the Martian surface to help enable communication between human settlements. However, due to the rough conditions on Mars, some transmitters experience an associated signal loss, expressed as a percentage. In order to enable long-range communication, NASA has enabled the transmitters to forward signals received from other transmitters to transmitters further away. Unfortunately, this comes with the caveat that signal loss is compounded as it moves between stations. For instance, if a signal passes through two stations, one with a 10% signal loss, and the other with a 20% loss, the resulting signal afterwards would only be (100% - 10%) * (100% - 20%) = 72% of the original signal indicating a total signal loss of 28%.

Because the transmitters are reasonably close, the distance between stations does not affect signal loss. However, transmitters only operate at certain frequencies which means that some transmitters cannot communicate with each other directly. In this case, the two transmitters A and B would have to communicate by relaying through another transmitter C that can operate at frequencies of A and B. For example, if A operates at 1.2 GHz, and B at 1.4 GHz, C would need to operate at both 1.2 GHz and 1.4 GHz in order to function as a relay. Given the signal losses associated with each transmitter and different operating frequencies of each transmitter, NASA would like for you to determine the minimum signal loss attained between two stations. Assume all transmitters will be able to communicate in some way.

Input

The first line of input will contain a single integer n, $1 \leq n \leq 9$, the number of test cases. Each case will begin with a positive integer $k < 20$, indicating the number of antennae. The following k lines denote an antenna description. Each antenna description begins with a single alphabetic character id uniquely identifying transmitter. The next field will be a positive integer < 100 for the amount of loss associated with that transmitter. Next, a positive integer $f < 5$ for number of frequencies the transmitter can operate at. There will be f field(s) following, each with a positive integer denoting an operating frequency the transmitter can operate. After k transmitter descriptions, a line follows with two transmitter ids separated by a space.

Output

For each case, output minimum possible loss from getting from the two provided transmitter ids. Note that the loss occurs at the transmitter, so start and end transmitters must be included in loss calculation. Each output will be printed at start of a new line rounded to the nearest thousandth and displayed to exactly three decimal places.

Sample Input

```
1
5
a 1 2 1 2
b 5 2 1 3
c 2 2 1 4
d 2 2 4 5
e 3 2 3 5
a e
```

Output Corresponding to Sample Input

```
7.773
```