1. Antonia

Program Name: Antonia.java **Input File: None**

Antonia wants to salute the teams who are competing at the State UIL Meet this year.

She has created a message: "BRAVO UIL COMPSCI"

She would like to print it in a very specific way. Each letter is on a different line and is printed in uppercase letters. Each letter will appear the same number of times as its position in the alphabet. An "A" would be printed once, a "B" would be printed two times … a "Z" would be printed 26 times. Empty lines separate the words.

Help Antonia print her cool message.

Input: None

Output: The message as shown below along left edge of screen. Notice the two empty lines.

Sample input: `None`

Sample output:

```
BB
RRRRRRRRRRRRRRRRR
A
VVVVVVVVVVVVVVVVVVVVV
OOOOOOOOOOOOOO

UUUUUUUUUUUUUUUUUUU
IIIIIIIII
LLLLLLLLLLL

CCC
OOOOOOOOOOOOOO
MMMMMMMMMMM
PPPPPPPPPPPPPPP
SSSSSSSSSSSSSSSSS
CCC
IIIIIIIII
```

2. Chongan

Program Name: Chongan.java **Input File: chongan.dat**

Chongan is an elementary school teacher. He is organizing his classroom crayons.

It is the end of the school year and, of course, he is exhausted. This is usually not his favorite task, so he decided to make it into a fun learning experience for his first-graders. He will incorporate a lesson that involves color names, counting, typing on a keyboard, and learning about the concept of "most."

All of the crayons are either primary colors (RED, BLUE, YELLOW) or secondary colors (ORANGE, GREEN, PURPLE). The kids will type in the name of each color and then type in the number of crayons in their bucket that are that color. The program will print out the color with the most crayons.

You are writing this program for Chongan to be sure, but really, it is for the kids.

Input: The first line will contain an integer, `n`, denoting the number of test cases in the data. The number of test cases will be in the range of [5,25]. Each line of data will consist of six pairs of colors and numbers with items separated by one space. The colors will be one of these upper-case words (ORANGE, GREEN, PURPLE, BLUE, RED, YELLOW). Each color will be followed by an integer in the range [0,100] (ORANGE 41 means there are 41 orange crayons). The colors will be in random order, but all six colors will appear once on each line of data. There will be no ties in determining the winner.

Output: For each test case, output one of the six colors indicating which crayon was more numerous. Make sure there are no extra trailing spaces in your lines of output.

Sample input:

```
5
RED 3 BLUE 7 YELLOW 8 ORANGE 11 PURPLE 2 GREEN 1
GREEN 17 YELLOW 3 RED 9 PURPLE 33 ORANGE 4 BLUE 19
PURPLE 35 GREEN 17 RED 48 YELLOW 14 ORANGE 50 BLUE 5
ORANGE 14 BLUE 36 GREEN 44 RED 29 YELLOW 7 PURPLE 26
YELLOW 17 PURPLE 36 GREEN 43 RED 8 ORANGE 19 BLUE 47
```

Sample output:

```
ORANGE
PURPLE
ORANGE
GREEN
BLUE
```

3. Daria

Program Name: Daria.java **Input File: daria.dat**

You and your friend Daria are in Calculus class together, and you both finished the test early. You need a 100 on this test to pass, so you ask your teacher, Mr. Pohade, if there is any way you can get extra credit. He says yes, you can gain some extra credit by grading some assignments for him, now that you have finished the exam. You will be given the answer key for each assignment and each students' answers. Your job is to grade these assignments and print out the average and median scores. Every assignment will be multiple choice, with a given number of questions for each assignment.

Input: The first line will contain an integer, `n`, denoting the number of assignments to be graded by you and Daria. The first line of each test case will consist of 2 integers, `a` and `b`, denoting the number of students who completed the assignment and now need to be graded, and the number of questions in the assignment. The following line will contain a string of `b` characters (A-E), denoting the correct answers for that assignment. Each of the following `a` lines will contain `b` characters, denoting the answers chosen by a particular student.

Output: For each assignment, first, on its own line, output the following string "`Assignment #`", followed by the number of the assignment (the first assignment is 1, and count upwards by 1), followed by a colon. Then, on the following line, output the string "`Mean Score: `", followed by the mean score on the assignment, rounded to two decimal places. Then, on the line following that, output the string "`Median Score: `", followed by the median score on the assignment, formatted to one decimal place. All assignment listings should be followed by an empty line, except for the last one. See sample output for examples.

Sample input:

```
3
3 5
ABCED
ABCEC
ABCDE
EDCBA
1 8
ABABCEBA
ABCDEDCB
5 3
AAE
AAB
BAE
ACE
AAC
CCC
```

Sample output:

```
Assignment #1:
Mean Score: 53.33
Median Score: 60.0

Assignment #2:
Mean Score: 25.00
Median Score: 25.0

Assignment #3:
Mean Score: 53.33
Median Score: 66.7
```

4. Gabriel

Program Name: Gabriel.java **Input File: gabriel.dat**

Gabriel is fascinated by boxes. He seems to do everything using some kind of box-like pattern. This is the way his mind likes to "store" information. Gabriel has always studied for tests using this method. It allows him to learn information in a very unique way.

Gabriel needs you to write a program that will create boxes for written text that is provided. He will provide you with the size of the box (we will call it N) and the text message (uppercase letters and spaces).

You shall first, remove any spaces from the message. Next you will organize the boxes using N.

The letters will be organized in N rows and N columns of character strings, each string N-letters wide.

For example, if N=5, you will create a 5x5 chart of strings, each string consisting of 5 letters. Letters will be used one-by-one moving across the string in a row-major order method. It is possible that there are more than enough letters to fill in the box. If so, the remaining letters will not be used.

It is also possible that there are too few letters to fill in the chart. If so, print the message "NOT ENOUGH LETTERS"

Look closely at the 5 examples below.

Input: The first line will contain an integer, `T`, denoting the number of test cases in the data. The number of test cases will be in the range of [5,25]. Each line of data will consist of an integer in the range [1,10], then a space, then a string of characters and spaces. The string will have a length in the range [1,400].

Output: For each test case, each box of output should include N lines of N-letter strings separated by one space. If there are not enough letters, then "NOT ENOUGH LETTERS" will be displayed. Make sure there are no extra trailing spaces in your lines of output.

Sample input: (*Line of data with N = 4, is a long line that continues to an indented second line*)

```
5
2 UIL COMPSCI
1 EARL CAMPBELL
3 WHAT STARTS HERE CHANGES THE WORLD
10 INTERSCHOLASTIC LEAGUE
4 THE SCREEN DOOR SLAMS MARYS DRESS SWAYS LIKE A VISION SHE DANCES ACROSS THE
   PORCH AS THE RADIO PLAYS
```

Sample output:

```
UI LC
OM PS
E
WHA TST ART
SHE REC HAN
GES THE WOR
NOT ENOUGH LETTERS
THES CREE NDOO RSLA
MSMA RYSD RESS SWAY
SLIK EAVI SION SHED
ANCE SACR OSST HEPO
```

5. Hong

Program Name: Hong.java **Input File: hong.dat**

Hong wants your help to write a program to convert positive integer values between 1 and 3999, inclusive, into their equivalent Roman numeral form. The table to the right contains the basic Roman numeral symbols and their decimal values. While the I, X, C, and M can be repeated as needed, there will never be more than one V, L, and D, if they are used.

Roman Symbol	Decimal Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

When the symbols are strictly declining in value from left-to-right (M to I as shown in the table), the decimal value of a Roman numeral is simply the sum of the values of the individual symbols like:

MDCCLXXVI = 1000 + 500 + 100 + 100 + 50 + 10 + 10 + 5 + 1 = 1776

However, a subtraction rule exists to shorten some numerals. When a single symbol of lesser value, like an I, precedes a symbol of greater value, like an X, the smaller value is subtracted from the larger value. This allows shorter numerals like IX for 9 and XC for 90 instead of VIIII and LXXXX. The table to the right shows the allowed combinations. Those combinations would not be followed by the same subtraction symbols so XCXX and even XCL are not allowed because XCXX would be 90 + 10 + 10 = 110 or CX, and XCL would be 90 + 50 = 140 or CXL. Multiple subtractions from a single symbol, like IIX or XXC, are also not allowed. The following example uses both of the above techniques:

Subtraction Combination	Decimal Value
IV	4
IX	9
XL	40
XC	90
CD	400
CM	900

MCMLXXIV = 1000 + (1000 - 100) + 50 + 10 + 10 + (5 - 1) = 1974

However, Hong wants start with a positive integer between 1 and 3999 and produce its Roman numeral form, the reverse of the above process, as shown in the following examples:

1776 is at or over 1000 and needs 1 **M**
Leaving 776, which is at or over 500 and needs a **D**
Leaving 276, which is at or over 100 and needs 2 **C**s
Leaving 76, which is at or over 50 and needs an **L**
Leaving 26 which is at or over 10 and needs 2 **X**s
Leaving 6, which is at or over 5 and needs a **V**
Leaving 1 and needs 1 **I**
Producing **MDCCLXXVI**

1974 is at or over 1000 and needs 1 **M**
Leaving 974, which is at or over 900 and needs a **CM**
Leaving 74, which is at or over 50 and needs an **L**
Leaving 24 which is at or over 10 and needs 2 **X**s
Leaving 4, which is at 4 and needs an **IV**
Producing **MCMLXXIV**

Input: Each test case consists of a single integer $1 \leq \mathtt{R} \leq 3999$ with an unknown number of integers per line separated by whitespace and an unknown number of lines in the input file. There will be less than 100 test cases.

Output: For each test case, output one line formatted as shown in the sample output. The line contains the input integer followed by " -> " and its Roman numeral form.

Sample input:

```
1235 2023 1776 3888 1974
949 19 1
```

Sample output:

```
1235 -> MCCXXXV
2023 -> MMXXIII
1776 -> MDCCLXXVI
3888 -> MMMDCCCLXXXVIII
1974 -> MCMLXXIV
949 -> CMXLIX
19 -> XIX
1 -> I
```

6. Kumar

Program Name: Kumar.java **Input File: kumar.dat**

Kumar is just fascinated by repeating decimals. Repeating decimals are rational numbers whose decimal representation eventually become periodic. In other words, the same sequence of digits repeats indefinitely. For example, the fraction ⅓ is represented as .333333333… = .$\underline{3}$

Repeating decimals are not limited to a single digit, nor does the repeating sequence have to begin directly following the decimal point. For example, the fraction 60611/499950 is represented by .12123412341234…= .12$\underline{1234}$

In this case, the repeating sequence is 1234, but does not start until the third decimal place after the decimal point.

At present, there is no single universally accepted notation or phrasing for repeating decimals. However, this hasn't stopped Kumar from attempting to write code that will be able to read in a decimal number, either repeating, or non-repeating, and output the corresponding fractional number. For the purposes of Kumar's proposed code, input will consist of a decimal number in range of [0,1). The decimal number is assumed to be not repeating, unless the number is followed by an ellipsis (...). For cases where an ellipsis is present, the repeating sequence will be repeated exactly three times. Your job is to help Kumar write source code that will read in the decimal number and output the corresponding, reduced fraction. Think you're up to the task?

Input: The first line will contain a single integer `n` that indicates the number of data sets that follow with `n` guaranteed to be in range [1,20]. Each data set will consist of a single decimal number `d`, where `d` will be in range the [0.0, 1.0). The number of decimal places after the decimal point will not exceed 14. Input cases denoting a repeating decimal will be followed by an ellipsis (...). For cases with an ellipsis, the repeating portion will be guaranteed to repeat three times.

Output: For each data set, you are to output the original decimal number followed by " `is equivalent to` " then the reduced fraction the decimal represents. For a decimal number of .0, you are to output "`0/1`".

Sample input:

```
8
.12345678
.87654654654...
.25
.125
.333...
.12123412341234...
.0
.123412341234...
```

Sample output:

```
.12345678 is equivalent to 6172839/50000000
.87654654654... is equivalent to 29189/33300
.25 is equivalent to 1/4
.125 is equivalent to 1/8
.333... is equivalent to 1/3
.12123412341234... is equivalent to 60611/499950
.0 is equivalent to 0/1
.123412341234... is equivalent to 1234/9999
```

7. Lucas

Program Name: Lucas.java **Input File: lucas.dat**

Lucas wants to find a way to give a value to any word. That value will be based on the specific letters in the word and how many times each letter appears in the word.

He devised a plan that will count the number of times each letter of the alphabet occurs. For each letter, multiply its position in the alphabet times the square of its number of occurrences within the word. Then, find the sum of those products. Letters not appearing in the word would contribute 0 points to the total.

Examples:

"eee" would produce the value of 45 ($5 * 3^2$)
"ccttcc" would produce the value of 128 ($3 * 4^2 + 20 * 2^2$)

Write a program that will help Lucas in this endeavor.

Input: The first line will contain an integer, `n`, denoting the number of test cases in the data. The number of test cases will be in the range of [5,25]. Each line of data will consist of a single string consisting of only lower-case letters. The length of the string will be [1,40].

Output: For each test case output an integer representing the word's value.

Sample input:

```
5
abcdab
aaabbjjjj
zjztz
uuuuiiill
cccompsci
```

Sample output:

```
19
177
264
465
120
```

8. Mohit

Program Name: Mohit.java **Input File: mohit.dat**

Your friend Mohit is creating a new tech start up, and he needs you to write a program to filter out all the resumes he needs for open jobs, and provide him the top 3 candidates for each given job.

You will be given a large pool of applicants, and then a number of jobs with listed qualifications, for each of which you will determine the top 3 applicants from the pool.

Input: The first line will contain two space separated integers, `m` and `n`, denoting the number of applicants in the applicant pool, and the number of jobs to be checked. The following m lines will contain the name of the applicant, the education level of the applicant, the experience of the applicant, the skills of the applicant, and any awards/certifications the applicant has received, in that order, all in the form of strings, in the format shown below (colons between names of fields and values, semicolons between each field, and commas between different attributes within a field, look to the example and structure listed below:

Applicant structure (this will all appear on one line in the actual data):

```
Name: (Applicant name); Education: (Applicant education level) of
(Field); (Skill 1): (# years of experience with Skill 1), (Skill 2): (#
years of experience), (Skill 3): (# years of experience); Awards:
(Award 1), (Award 2)
```

Example applicant (this will all appear on one line in the actual data):

```
Name: Jonathan Majors; Education: Bachelor's of CS; Java: 4 years, C: 3
years, Python: 2 years, IOS Programming: 3 years; Awards: MS
Certification, Java 8 Certification
```

The following n lines will each contain the job title, followed by all qualifications required for the job, in the format shown below:

Job listing structure: (this will all appear on one line in the actual data):

```
Job: (Job name); Education: (Required minimum education level) of
(Required field); (Skill 1): (# years of required experience with Skill
1), (Skill 2): (# years of required experience); (Required award 1)
```

Example job listing: (this will all appear on one line in the actual data):

```
Job: Software Engineer; Education: Bachelor's of CS; Java: 2 years,
Python: 2 years; MS Certification
```

Notes:

- There can be up to 5 required skills or required awards for a single job listing.
- The format shown in the structure sections above will be followed for all student and judge data, so code accordingly.
- An applicant can have an unlimited number of skills or awards, but only one listed education.
- Any of the fields other than name and job title can be `None`, which means that particular field has no attributes, so `Awards: None` would mean that that particular applicant has no awards. A job can also have `None` listed for any of the qualifications, this indicates that the job has no preference when it comes to skills, or education, or awards. If a job has `None` listed for any of the job qualifications, that job element does not disqualify anyone, all applicants would qualify regardless of their related data. So, if a job had `None` for education requirement, everyone qualifies unless disqualified for another reason.

~ Output description and more on next page… ~

~ Mohit, continued… ~

Output: For each job, output the title of the job, followed by a list of at most 3 applicants who are qualified for the job, sorted by the criteria listed in the Sorting Criterion section below. In order for an applicant to qualify for a job, the applicant must meet all the listed requirements for the job except for the `None` elements. For education: There are 3 levels of degree, Master's, Bachelor's, and PhD, PhD > Master's > Bachelor's, meaning, anyone with a PhD in CS is qualified for a job requiring a Master's or Bachelor's in CS, and anyone with a Master's is qualified for a job requiring a Bachelor's in the same field. Any degree in the incorrect field is invalid unless the job requirement is `None`. If the applicant meets the education requirement, we check the skills requirements, meaning that for each skill listed, the applicant must have at least that many years of experience with that skill. If the applicant is qualified for all skills, then we check for awards, meaning that the applicant must have every required award for the job. If all of these are true, then the applicant is qualified. Now that we have all of the qualified applicants, we sort them by the following criteria to find the top (at most) 3 for each job posting.

Sorting Criterion:

1) First, sort by education. Collect all the applicants that meet the education requirement, and sort by the criterion: PhD > Master's > Bachelor's, meaning PhD comes before Master's comes before Bachelor's, as long as the degree is in the correct field or there is no education or field requirement.
2) If the education of two applicants are equal, we then sort based on skill score, or S, as shown below:

$$S = \sum_{i=1}^{n} (applicant\ years\ of\ experience) - (required\ years\ of\ experience)$$

   Where n = # of skills required by the job posting. We calculate this for each of the applicants for the job, and sort in decreasing order.
3) If the skill scores of two applicants are the same, then we sort alphabetically by name, and no 2 applicants will ever have the exact same name.

After sorting, we print the name of the job, on its own line, followed by (at most) the top 3 applicant's names, with the numbers shown, followed by an empty line following the format of the sample output. If there are no applicants that meet the requirements, output the name of the job, as in all the other cases, followed by a line stating `"No Applicants meet Requirements."`

Sample input: (*Each entry will consist of exactly one line, indented lines are continuations*):

```
6 2
Name: Jonathan Majors; Education: Bachelor's of CS; Java: 4 years, C: 3 years, Python:
      2 years, IOS Programming: 3 years; Awards: MS Certification, Java 8
      Certification
Name: Samantha Samuels; Education: PhD of Medicine; Surgery: 8 years, Bones: 3 years,
      Neuroscience: 1 year; Awards: Nobel Peace Prize
Name: Neil Neil; Education: Master's of CS; C: 7 years, Java: 7 years, Python: 3
      years; Awards: MS Certification
Name: Beatrice Bengal; Education: Bachelor's of CS; Java: 4 years, Python: 4 years;
      Awards: MS Certification, Java 8 Certification
Name: Dan Daniels; Education: Master's of CS; C: 7 years, Java: 7 years, Python: 1
      year; Awards: None
Name: Bonnie Armstrong; Education: Master's of Accounting; Tax Law: 3 years,
      Accounting: 8 years; Awards: MS Certification
Job: Software Engineer; Education: Bachelor's of CS; Java: 2 years, Python: 2 years;
      MS Certification
Job: Accountant; Education: Master's of Accounting; Accounting: 4 years; MS
      Certification
```

Sample output:

```
Software Engineer:
1: Neil Neil
2: Beatrice Bengal
3: Jonathan Majors

Accountant:
1: Bonnie Armstrong
```

9. Pilar

Program Name: Pilar.java **Input File: pilar.dat**

Pilar has been studying multi-dimensioned arrays and writing programs to work with them. In addition, she recently found out that not all languages use zero-base arrays which makes them more difficult to work with for a Java programmer. With zero-based arrays, the first column of a 1-D array is element [0], the first row of a 2-D array is row [0], and the first layer of a 3-D array is layer [0]. Some other programming languages treat arrays differently than Java. In Java, a 3-D array of floats with 2 layers, each layer with 3 rows, and each row with 4 columns would be declared as follows:

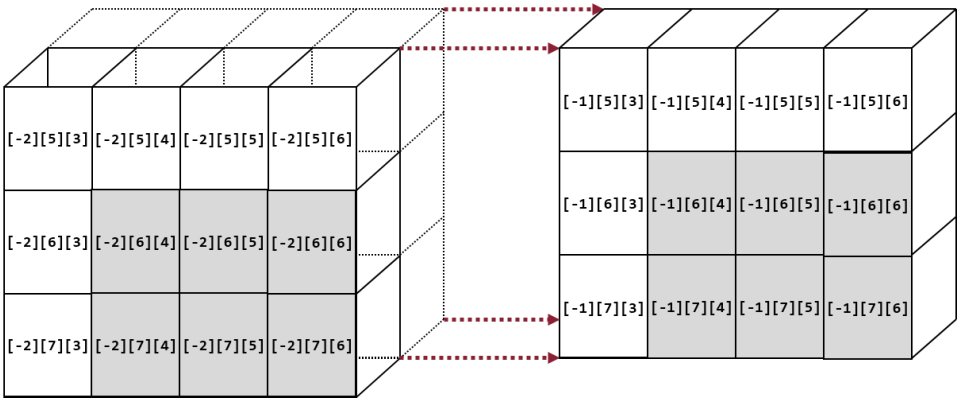
```
float [][][] data = new float[2][3][4];
```

Suppose another language allows the programmer to specify a range for the index values instead of the size of that dimension. Perhaps, something like the following:

```
float [][][] data = new float[-2:-1][5:7][3:6];
```

The arrays are actually the same size but the index values in the program code would have the stated ranges instead of 0..1, 0..2, and 0..3, respectively.

In addition, some programming languages allow for indexing of "slices" of an array. Using the above example, `data[-2:-1][6:7][4:6]` would be a "slice" consisting of the portion of layers `[-2]` and `[-1]` containing only rows `[6]` and `[7]` containing only columns `[4]`, `[5]`, and `[6]`. The "slice" would be 12 cells, those that are shaded in the following illustration with the "back" layer pushed to the right.



"Slices" of a 3-D array could result in either a 2-D or a 1-D portion of the array like the following examples:

- `data[-2:-2][6:7][4:6]` contains only the shaded cells in the front layer (2-D)
- `data[-1:-1][6:7][6:6]` contains only the shaded cells along the right edge of the back layer (1-D)
- `data[-2:-1][6:6][3:6]` contains only the cells in the second row of both layers (2-D)

Your challenge is to implement such an array and addressing mechanism via Java program code.

Input: First line contains a single integer, number of test cases `T`, which will not exceed 20. For each test case, the next line contains three comma-separated index ranges for a 3-D array, in the form $LB_1:UB_1,LB_2:UB_2,LB_3:UB_3$ where LB_1, LB_2, and LB_3 are lower bounds and UB_1, UB_2, and UB_3 are upper bounds for the layers, rows, and columns, respectively. The bounds will not exceed the range -5000:5000 but there will be no more than 1000 data elements in the array. The following lines will contain one comma-separated row of float data to populate the first layer's first row and then each subsequent line fills the next row, followed by rows of data for subsequent layers. The data is followed by a single line with a single integer `N`, the number of "slices" to be processed, which will not exceed 20. That line is followed by `N` lines, each containing a single comma-separated "slice" in the form $L_1:L_2,R_1:R_2,C_1:C_2$ where `L`'s are layer indexes, `R`'s are row indexes, and `C`'s are column indexes. All index values will be valid for the defined array with the first value ≤ the second value.

~ Pilar continues on next page… ~

~ Pilar, continued… ~

Output: For each test case, output a single line containing the number of elements in the "slice" or "chunk" and the sum of the data items in that "slice" of the array with two decimal places, separated by a single colon **":"** and no additional spacing. Display a line containing 20 equal signs **"=========="** after each test case.

Sample input:

```
2
-2:-1,5:7,3:6
181.79,281.44,403.51,391.91
263.49,378.95,183.85,302.14
142.40,359.64,103.03,370.77
141.24,157.40,377.87,370.56
172.24,104.71,268.41,437.07
283.82,270.07,377.71,380.63
5
-2:-1,6:7,4:6
-2:-2,6:7,4:6
-1:-1,6:7,6:6
-2:-1,5:7,3:6
-1:-1,7:7,5:5
-2:2,216:219,-5:1
942.32,885.71,1422.91,997.57,1641.72,731.13,946.22
975.76,1324.37,654.44,876.39,682.12,998.73,930.60
1652.45,1178.42,1526.27,695.67,1445.09,1652.55,1636.50
717.32,825.24,922.28,683.85,1377.75,542.52,975.56
818.08,987.91,1443.32,1465.33,799.53,1283.15,1038.03
933.23,769.96,1115.08,1522.71,1354.24,1376.36,797.33
869.43,769.70,1432.87,1444.79,1395.87,955.11,834.51
1269.61,880.45,1086.11,515.52,1283.08,1461.40,618.78
864.47,1140.16,876.82,1021.34,1010.39,855.54,942.49
1126.07,674.36,1655.41,563.44,1401.03,821.88,1339.96
718.68,1416.45,1449.02,1580.02,1251.35,607.76,1322.51
969.41,791.67,956.94,1473.29,1537.63,887.91,1107.66
994.01,980.29,1654.21,674.13,760.57,924.61,1326.91
1052.71,1021.48,1016.85,1632.13,1605.99,1195.87,810.72
890.97,1276.26,1540.13,1265.17,984.05,1488.21,1239.79
1021.71,1347.82,1391.01,871.86,1282.98,504.53,941.62
1341.02,1464.87,864.14,1220.91,994.54,757.11,875.26
702.96,1540.39,755.44,1354.18,957.68,1402.33,593.84
1305.26,1343.95,977.22,855.98,561.74,1077.65,1034.47
655.68,1377.09,917.65,1512.59,1524.67,962.17,741.69
6
-1:1,217:218,-4:0
-2:0,216:218,-3:-1
-1:2,217:217,-2:0
0:0,217:219,1:1
-2:2,216:219,-5:1
0:0,218:218,-2:-2
```

Sample output:

```
12:3536.98
6:1698.38
2:817.70
24:6704.65
1:377.71
==========
30:36583.55
27:32724.74
12:15187.84
3:3770.13
140:152095.68
1:1580.02
==========
```

10. Ruslan

Program Name: Ruslan.java **Input File: ruslan.dat**

Your Algebra teacher has made a new seating chart! Now your only friend in the class, Ruslan, is on the other side of the class. You will have to use your knowledge of high school social structures to send a message across the classroom to Ruslan.

Input: The first line will contain a variable n, denoting the number of data sets to follow. Each data set will consist of an unknown number of entries, each on their own line, succeeded by a line of 5 '=' characters, like "=====". Each entry in the data set will consist of two strings, each representing a name of a student, signifying their connection, or your ability to get a note passed from student to student. For example, if the entry was as follows: "`Daniel Johnny`", we know that we can pass message from Johnny to Daniel, and vice versa . The name "`Ruslan`", signifying your friend Ruslan, will be present at least once in each data set, as will the name "`You`", signifying you.

Output: The output for each test case should be the number of unique ways to get a message from you to Ruslan. None of the paths should ever visit a student more than once, as this will arouse suspicion from the teacher. You will output the number of distinct paths from you to Ruslan as an integer, on its own line, as shown in the sample output.

Sample input:

```
3
Samuel Stanley
You Stanley
Samuel Alice
Stanley Alice
Alice Ruslan
=====
You Beth
Beth Candace
Candace You
Ruslan Ben
=====
Ruslan Vlad
Vlad Alice
Alice Bob
Steven Vlad
Bob Ruslan
Daniel Steven
Wesley Matthew
Matthew You
Wesley Daniel
Alice Daniel
=====
```

Sample output:

```
2
0
4
```

11. Sofia

Program Name: Sofia.java **Input File: sofia.dat**

Sofia has previously written a program to calculate wind chill or "feels like" temperatures when it is cold, but many areas of Texas do not experience the extreme cold weather. Instead, Texans often suffered the dreaded high temperature and high humidity combination that leads to dangerous heat warnings, so she has decided to write a similar program for calculating heat index or "apparent" temperature.

Sofia's search across the Internet has revealed heat index (**HI**) calculations are more complicated than windchill and involve several formulas and special rules. It starts with the ambient air temperature (**t**) measured in Fahrenheit (°F) and relative humidity (**h**) as a percentage between 0 and 100, inclusive.

The first step is to use the following formula which handles the lower side of the temperature range. When the result is less than 80, this is the calculated **HI**.

$$\mathbf{HI} = (\mathbf{t} + 0.5 * (\mathbf{t} + 61.0 + ((\mathbf{t} - 68.0) * 1.2) + (\mathbf{h} * 0.094))) / 2$$

Otherwise, continue with the general formula and two possible adjustments:

$$\mathbf{HI} = -42.379 + 2.04901523\mathbf{t} + 10.14333127\mathbf{h} - 0.22475541\mathbf{th} - 6.83783 \cdot 10^{-3}\mathbf{t}^2 - 5.481717 \cdot 10^{-2}\mathbf{h}^2 + 1.22874 \cdot 10^{-3}\mathbf{t}^2\mathbf{h} + 8.5282 \cdot 10^{-4}\mathbf{t}\mathbf{h}^2 - 1.99 \cdot 10^{-6}\mathbf{t}^2\mathbf{h}^2$$

1. When humidity is less than 13% and temperature is at or between 80 °F and 112 °F, the following adjustment is subtracted from **HI**:

   $$\mathbf{ADJUSTMENT} = ((13 - \mathbf{h}) / 4) * \text{SQRT}((17 - \text{ABS}(\mathbf{t} - 95)) / 17)$$

2. When humidity is greater than 85% and temperature is at or between 80 °F and 87 °F, the following adjustment is added to **HI**:

   $$\mathbf{ADJUSTMENT} = ((\mathbf{h} - 85) / 10) * ((87 - \mathbf{t}) / 5)$$

Finally, when the calculated heat index is over 140 °F or below 75 °F it is considered invalid and should not be displayed as a heat index.

Input: First line of input will contain a single integer as the number of test cases, **N**, with range $2 \le \mathbf{N} \le 50$. Each test case consists of two floating point values on one line separated by whitespace: ambient air temperature **t** (°F) with range $50.0 \le \mathbf{t} \le 140.0$ and relative humidity **h** with range $0.0 \le \mathbf{h} \le 100.0$.

Output: A single line starting with "**`Test case #`**" followed by no space and the test case number and a colon as shown in sample output. That is then followed by exactly 1 space and then either the computed heat index rounded to 1 decimal place or the phrase "**`Heat Index #.# NOT VALID!`**" with **`#.#`** as the calculated value.

Sample input:

```
10
92.7 61.5
82.3 11.4
89.6 34.9
87.3 87.9
90 100
130.0 15.0
102.1 66.2
111.1 32.5
73.3 72.0
94 85
```

Sample output:

```
Test case #1: 107.7
Test case #2: 79.6
Test case #3: 88.6
Test case #4: 109.0
Test case #5: 131.6
Test case #6: 138.8
Test case #7: Heat Index 146.3 NOT VALID!
Test case #8: 128.1
Test case #9: Heat Index 73.5 NOT VALID!
Test case #10: 135.2
```

12. Vijay

Program Name: Vijay.java **Input File: vijay.dat**

You have hired a DJ for your annual "**END OF THE YEAR BIG TIME BIG BOY BIRTHDAY BASH**", for your friend Kevin. The dj's name is Vijay, but he is having some technical difficulties. It seems he left all of his super important dj CDs in your friendly neighborhood 3D space with no gravity. You need to quickly retrieve the CDs and escape before your show starts.

Input: The first line will contain an integer, `n`, denoting the number of test cases in the data. The first line of each test case will contain 5 integers, `h`, `r`, `c`, `o`, and `s`, denoting the # of levels in the maze ($0 < h < 20$), the # of rows in the maze ($0 < r < 20$), the # of columns in the maze ($0 < c < 20$), the # of CDs located within the maze ($0 < o < 10$), and the amount of time, in seconds, that you have to get all the CDs ($0 <= s <= 25000$). The next `r` lines will contain `c` characters denoting the layout of the maze. These `r` lines denote the first layer of the maze, while the next `r` denote the second layer of the maze, and so on. These characters can be any of the following:

- `'#'` – Denotes a wall.
- `'.'` – Denotes an empty space.
- `'O'` – Denotes the location of a CD within the maze.
- `'S'` – Denotes the starting location for you.
- `'E'` – Denotes the exit of the maze (may be more than one).

In terms of movement within the maze, every step you take in any direction will take one second. You can move in any of the main 6 directions, but only one direction and step at a time, so you can only move up, down, forward, backward, right, and left. Note: backtracking IS allowed.

Output: For each data set, if you cannot successfully navigate your way out of the maze within the time limit, output the string "`None shall pass.`". If you can successfully get out of the maze, output the string "`# of steps achieved: `", followed by the number of steps you took to escape the maze.

Note: if you have 18 seconds to escape the maze, and on your 18th step you get to the exit, you made it!

~Sample input and output on next page, to make it easier to read~

~ Vijay, continued… ~

Sample input:

```
3
2 5 5 2 10
##S##
#...#
#O..#
#...#
##E##
#####
#...#
#...O
#...#
#####
3 4 4 2 8
####
#S.#
#.O#
####
####
#..#
#.O#
####
####
#.E#
#..#
####
1 7 7 3 15
#######
#S..O.#
#..O..#
#.....#
#...E.#
#O....#
#######
```

Sample output:

```
None shall pass.
# of steps achieved: 5
# of steps achieved: 14
```