University Interscholastic League

Computer Science Competition

2013 State Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.
2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Names of Problems

Number	Name
Problem 1	Absolute Differences
Problem 2	ADQL
Problem 3	Black Friday
Problem 4	Diamonds
Problem 5	Eagles
Problem 6	Friendlier
Problem 7	Mortgage
Problem 8	Savings Account
Problem 9	Scrabble Swiss Seed Tournament
Problem 10	Sort, By George
Problem 11	Triangles
Problem 12	Word Find 2

1. Absolute Differences

Program Name: Absolute.java   Input File: absolute.dat

When Mary is bored in class, she sometimes likes to play with numbers. One day when Mary was playing with numbers, she wrote down a circular sequence of four non-negative integers and repeatedly took the absolute value of the difference of consecutive integers. To complete the circle, she paired the last integer with the first. Consider the sequence 1, 2, 5, 7. Applying the above algorithm four times results in the sequence containing all twos as shown below.

```
   1 2 5 7 >> 1 3 2 6 >> 2 1 4 5 >> 1 3 1 3 >> 2 2 2 2
```

You have decided to write a program that will determine the number of times the algorithm must be applied before all elements in the sequence are equal and what that integer is.

Input
The first line of input will contain a single integer `n` that indicates the number of sequences to follow. Each of the following `n` lines will contain a sequence of four non-negative integers less than fifty that are separated by a single space.

Output
For each sequence input, you will print the number of times the algorithm must be applied before all elements in the sequence are equal followed by a space and what that integer is.

Example Input File
```
3
1 2 5 7
49 7 3 29
15 1 17 9
```

Example Output to Screen
```
4 2
3 16
3 6
```

2. ADQL

Program Name: ADQL.java Input File: adql.dat

Your boss needs you to write a parser for the Data Query Language specification for your company's database software. Your program will also test your parser by reading in a sample database, running some queries and printing the results of those queries to the screen. A database is just a table, where each column has a name, and each row is a database entry consisting of values for each of the columns.

The language itself is pretty straight forward, where queries are of the form:

```
    SELECT <ITEM> WHERE <NAME><OP><VALUE> [SORT <STYPE> [INT] <COL> [SORT …]]
```

and:

- `SELECT` always begins a query
- `<ITEM>` denotes which columns' data entries to return in the query, and can be:
    - an asterisk (`*`) which means to report the data entries from all columns, or
    - a comma separated list of column names (with a space after each comma) from which you are to report the data entries.
    - **Note**: If there is no comma, you will report only the data entries from the `<ITEM>` column meeting the query criteria.
- `WHERE` starts an optional segment of the query consisting of `<NAME>`, `<OP>` and `<VALUE>` which are used to run a logical check against the data entry to see if it should be included in the results:
    - `<NAME>` is the name of one of the columns.
    - `<OP>` is a logical operator, either =, <, >, <= or >=.
    - `<VALUE>` is the value to compare against. A column's data should be treated a string if the = operator is used, and an integer otherwise.
- `SORT` is an optional segment of the query (denoted by `[…]`) consisting of `<STYPE>`, `<INT>` which is optional, `<COL>`, and another optional embedded `SORT` segment, which denotes how to resort the results:
    - **Note**: if `SORT` is missing, leave the data entries in the order found in the data file, otherwise:
    - `<STYPE>` is either `DESC` for a descending sort, or `ASC` for an ascending sort.
    - `<INT>` is optional, and if it is present the comparison should be done as integers, or done as strings if `INT` is not specified.
    - `<COL>` is the name of the column to sort on.
    - If there is a second optional `SORT` clause:
        - take the results of the first sort and, while preserving the groupings of `<COL>`, resort those groups by a new `<COL>` using a new `<STYPE>` and possibly by `INT`, if so denoted in the second `SORT` clause.
        - if the second `SORT` is missing, leave the data entries in the order resulting from the first sort.

A query will return:

- if `<ITEM>` is an asterisk (`*`):
    - it will return all the data entries in the rows that match the `WHERE` segment, or
    - all the data entries in all rows if the `WHERE` segment is missing.
- otherwise:
    - it will return the data entries of the columns from the `<ITEM>` part of the query if there is matching data from the `WHERE` segment, or
    - `<EMPTY>` if there is no matching data.

(Continued on next page)

2. ADQL (cont.)

Input

The input file contains first the database data and then the queries.

- Database data:
    - The first line of the input file will contain two integers `c r` separated by a space, where the `c` is the number of columns and `r` is the number of database entries.
    - The second line will contain a list of column names for each column `c` of the database, where the names are separated by the pipe character "|" as shown below.
    - The next `r` lines will each contain a pipe delimited list of values for each column `c` of the database. A value for the last column is optional, as shown below.
- Queries:
    - Following the database data, there will be a line containing a single integer n that indicates the number of queries to follow.
    - Each of the following `n` lines will contain a single query in the format described above.

Output

For each query you will print the string `Query #`*`n`* on a line by itself, where *`n`* is the number of the query, starting at `1`. Then you will print the data from the query, one data entry per row. Each item from a data entry should be separated by a space. If the query returns no data, then you should print the string `<EMPTY>` on a line by itself. There should be a blank line after each set of query output.

Example Input

```
8 13
Name|ID#|Grade|ELA|Math|Science|SS|Notes
ROBERTS, MATIANA|212090|11|P|N|N|P|jr grad
BARROWS, GEORGE|102559|11|P|N|N|P|jr grad
BROOKS, ROGER|102661|12|N|N|N|P|2011
BROOKS, IAN|104132|11|P|N|P|P|jr grad
BUCHANAN, ALEXANDER|102453|12|N|N|N|P|jr grad
DELGADO, ABIGAIL|540250|10|P|P|N|N|2011
DODSON, NICOLE|240289|11|N|X|X|N|jr grad
ROGERS, ROBERT|742034|10|P|N|N|P|
JONES, RICHARD|770945|11|P|P|N|P|
ALLEN, ALAN|770814|12|P|N|N|P|2012
ROBERTS, ROBERT|770460|11|X|X|N|X|2012
JONES, ROGER|770152|11|P|N|N|P|
ROGERS, MARTIN|587716|10|P|N|P|P|jr grad
2
SELECT Name, ID# WHERE SS=P SORT DESC INT Grade
SELECT * SORT ASC SS SORT DESC Name
```

(Continued on next page)

2. ADQL (cont.)

Example Output to Screen
```
Query #1
BROOKS, ROGER 102661
BUCHANAN, ALEXANDER 102453
ALLEN, ALAN 770814
ROBERTS, MATIANA 212090
BARROWS, GEORGE 102559
BROOKS, IAN 104132
JONES, RICHARD 770945
JONES, ROGER 770152
ROGERS, ROBERT 742034
ROGERS, MARTIN 587716

Query #2
DODSON, NICOLE 240289 11 N X X N jr grad
DELGADO, ABIGAIL 540250 10 P P N N 2011
ROGERS, ROBERT 742034 10 P N N P
ROGERS, MARTIN 587716 10 P N P P jr grad
ROBERTS, MATIANA 212090 11 P N N P jr grad
JONES, ROGER 770152 11 P N N P
JONES, RICHARD 770945 11 P P N P
BUCHANAN, ALEXANDER 102453 12 N N N P jr grad
BROOKS, ROGER 102661 12 N N N P 2011
BROOKS, IAN 104132 11 P N P P jr grad
BARROWS, GEORGE 102559 11 P N N P jr grad
ALLEN, ALAN 770814 12 P N N P 2012
ROBERTS, ROBERT 770460 11 X X N X 2012
```

Notes:
- A blank line at the end of the output is optional.
- A space at the end of each line is optional.

3. Black Friday

Program Name: BlackFriday.java   Input File: blackfriday.dat

To avoid the yearly rampage, fighting and dismemberment of Black Friday shopping, a local store is instead having a drawing daily for prizes. Each day shoppers will be able to get free tickets for the sale items they want. At the end of the day there will be a drawing for each item, in alphabetical order, to see who gets to purchase it at the low, low price. The store wants you to write a program to perform the drawing and print the information for each day's worth of drawings.

For the drawings, the store has purchased a large roll of ticket stubs which start at number 100 and increment by 1. You can assume there is no limit to how high the ticket numbers go. Every ticket in a drawing for a given item has an equal chance of winning. When determining the winner of that item via `Random`, put the tickets for that item in the order they were distributed and choose one at random using the ticket location as an array index. The winner can then be determined from the winning ticket number.

For each day's drawing, you will need to construct one object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

Input

The first line of input will contain a single integer `n` that indicates how many days' worth of drawings there will be. For each day's drawings, there will be:

- a single line containing a `long` which is the seed for the random object you should use for the drawing.
- a single line containing a single non-negative integer `m` denoting the number of people who got tickets for the current day.
- `m` lines, each containing the name of a person, followed by a space and a space delimited list of the names of each of the sale items that person got tickets for.

Tickets will be given out in the order the items are read in from the file. The next day's tickets will continue where the first day's left off, so if 10 tickets were given out on the first day, then the first ticket given out the second day is numbered 110.

Output

For each day, you will print, on a single line, `Day` *`X`*`:` *`Y`* `tickets were given out`, where *X* is the day number, starting at `1`, and *Y* is the number of tickets given out on that day. Then, for each drawing that happened, in alphabetical order by item name, you will print three spaces (to indent the line), then `X: P won with ticket Y`, where `X` is the name of the item, `P` is the name of the person and `Y` is the ticket number that won.

Example Input File

```
1
354887534
6
Bob Pants Shirt TV
Steve Shirt Car Shoes TV
Stacy Pants Shirt Shoes DVD TV
Mickey Car Shoes Pants TV
James Keyboard Socks DVD TV
Tickles TV
```

Example Output to Screen

```
Day 1: 21 tickets given out
   Car: Mickey with ticket 112
   DVD: Stacy with ticket 110
   Keyboard: James with ticket 116
   Pants: Mickey with ticket 114
   Shirt: Bob with ticket 101
   Shoes: Steve with ticket 105
   Socks: James with ticket 117
   TV: Tickles with ticket 120
```

Random numbers generated:

```
354887534: 1 0 0 2 0 0 0 5
```

4. Diamonds

Program Name: Diamonds.java   Input File: diamonds.dat

Mary is playing a two person game of solitaire called Diamonds with her friend. The object of the game is to accumulate the diamonds with the most value.

The game is started by lining up all 14 of the diamonds on a board in order of value. The value of the cheapest diamond is $15 and they increase in value by $5 each until the last diamond has a value of $80. Each person's turn consists of a coin flip. If the coin lands on heads, the player will select the most expensive diamond available and if the coin lands on tails, the player will select the cheapest diamond available. The players will alternate taking turns.

You have been asked write a program that will simulate the outcome of a given number of games.

For each game, you will need to construct one object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

For each game, there will be two players, A and B. The player A will always go first. The random object will be used to determine the outcome of the coin flips necessary for that game. The coin flip will always result in either a zero or a one. A zero will indicate the coin landed on heads and a one will indicate the coin landed on tails.

Input

The first line of input will contain a single integer `n` that indicates the number of games to follow. Each of the following `n` lines will contain a long which will be the seed for the random object for the simulation.

Output

For each line of input, you will print one of the following:

- `PLAYER A WON` if player A won
- `PLAYER B WON` if player B won, or
- `TIE` if there is a tie.

Example Input File

```
3
54322355
32445664322
345632266777
```

Example Output to Screen

```
PLAYER B WON
PLAYER A WON
PLAYER A WON
```

Random numbers generated for each seed:

```
54322355: 0 0 0 0 1 0 1 1 0 0 1 0 0 0
32445664322: 0 1 0 0 0 0 0 1 0 1 1 0 0 1
345632266777: 0 1 0 1 0 1 0 1 0 1 0 0 1 0
```

5. Eagles

Program Name: Eagles.java   Input File: eagles.dat

Hermit Joe lives in a desolate area of Alaska. The Alaskan Wildlife Management team has given him several maps of the area surrounding his cabin that show the location of the homes of different types of animals. The characters on the map represent different animals that live in the area. For example, the letter `E` represents an eagle's nest, the letter `B` represents a brown bear den, the letter `M` represents a moose habitat, etc. Being an avid bird watcher, Joe is especially interested in where in the area that the eagles have nests. Being afraid of both bears and moose, Joe will go around any moose habitat and any brown bear den. He will go through the home or habitat of any other animal. You are to write a program that, given a 10x10 rectangular grid of the area, will determine the closest to his cabin he can find an eagle's nest.

Input

The first line of input will contain a single integer `n` that indicates the number of maps to be checked. For each map, the first line will contain two integers `r c` that indicate the coordinates of the cabin followed by ten lines (rows) with ten characters (columns) and no spaces on each row. The coordinate of the upper left corner of the map is `1 1` and the coordinates of the lower right corner is `10 10`. All characters in the map will be either an uppercase letter of the alphabet that represents an animal's home or a period (`.`) that represents an area without a major animal home. There will be exactly one `J` on the map to denote the location of Joe's cabin.

Output

For each map, you will print the least number of cells Joe must traverse to arrive at an eagle's nest without entering a moose habitat or a brown bear's den. Joe may move horizontally, vertically, or diagonally. Do not count the cell with Joe's cabin but do count the cell containing the eagle's nest.

Note: There will always be an eagle's nest that Joe can reach.

Example Input File

```
1
9 10
E...E..B.M
...B....ME
BBBCC.BBMM
EBBCCFFBMM
MMMM..B..M
MMM.B..FF.
MMMMFFFMMM
CCCCM..MMM
BECMM...FJ
..CMMFF.B.
```

Example Output to Screen

```
8
```

6. Friendlier

Program Name: Friendlier.java   Input File: friendlier.dat

After the success of your “Friendly” encryption that encrypts messages between individuals in your group of friends you have decided to tweak the algorithm to be a bit stronger for the best friends in the group. You used all of your creative juices to come up with the name for the new algorithm: Friendlier.

The messages you and your best friends will be sending are of the form `<NAME>: <MSG>` where `<NAME>` is the first name of the person sending the message and `<MSG>` is a series of ASCII characters which comprises the message.

The first thing the algorithm needs to do is create a seed that will be used to encrypt the text. This is done by starting with the prime number 97 then `XOR`’ing that value with each letter of the sender’s name, from left to right. To encrypt the message, you start by treating the message as a single, large binary value consisting of all the characters in order. Then you take the large binary value and rotate all the bits to the right by 5, making sure that the bits that fall off the right shift back on to the left. Once the message has been rotated you should take each byte and `XOR` it with the seed, which generates the final encrypted message.

Take, for example the first input value `Bob: :)`

Generate Seed	
97=0x61=	01100001
B=0x42=	01000010
XOR=	00100011
o=0x6F=	01101111
XOR=	01001100
b=0x62=	01100010
seed: XOR=	00101110

Encrypt message using seed		
:) = 0x3A 0x29 =	00111010	001**01001**
Rotate 5 bits from right onto left	**01001**001	11010001
XOR each byte to seed	00101110	00101110
Encrypted message	01100111	11111111
	0x67	0xFF

Input

The input file will contain an unknown number of lines, with one message to encrypt per line.

Output

You should print out one encrypted message per line, including the sender’s name followed by a colon and a space. The encrypted text should be printed out as hexadecimal numbers representing the byte value of each ASCII character, with a space after each, as shown below.

Example Input File

```
Bob: :)
Brick: I love lamp!
Ron: I'm Ron Burgundy?
```

Example Output to Screen

```
Bob: 0x67 0xFF
Brick: 0x3A 0x79 0x33 0x53 0x4B 0x83 0x19 0x33 0x53 0x3B 0x5B 0xB1
Ron: 0xC8 0x7B 0x09 0x5B 0x30 0xA1 0x49 0x43 0x30 0x21 0x99 0xA1 0x09 0x99
0x41 0x11 0xFB
```

Notes:

- The encrypted message for `Ron` extends across two lines in the printed version above but constitutes only one line in the actual output.
- A space at the end of each line is optional.

7. Mortgage

Program Name: Mortgage.java Input File: mortgage.dat

Your parents want to buy a new house and are curious how much house they can afford. They want you to write a program that, given the house price, their down payment, the APR (annual percentage rate) and the number of months to pay off the loan, will output the monthly payment, expressed as whole dollar amounts.

The algorithm for finding a monthly payment is:

$$P = (H - D) * [(A/12) * (1 + (A/12))^N] / [(1 + (A/12))^N - 1]$$

where:

- `P` is the monthly payment
- `H` is the amount of the house price
- `D` is the down payment
- `A` is the APR as a double from `0` to `1`
- `N` is the number of months for the loan

Input
The input file will contain an unknown number of lines, each of which will contain 4 values: the amount of the house price, the down payment, the APR as a value between `0` and `1`, and the number of months for the loan. You should read in and process all values as doubles.

Output
For each line of input you will print a single line, starting with a `$` followed by the monthly payment in whole dollar amounts with no decimal places, where any fractional dollar amount is treated as the next whole dollar, as shown below.

Example Input File
```
220230 20000 0.06 360
300000 15000 0.075 200
```

Example Output to Screen
```
$1201
$2501
```

8. Savings Account

Program Name: Savings.java   Input File: savings.dat

Your favorite, rich uncle, Ebenezer, has decided to help you with your college funds. Rather than just give you a lump sum, he has decided that he will give you some money each day for up to a year and let you put it in your college savings account.

Uncle Ebenezer has decided that he will give you $1 on the first day, $2 on each of the next two days, $3 on each of the next three days, and so forth for a given number of days. He has asked you to write a program that will let you know how much money you would have after a given number of days.

Input

The only line of input will contain a space delimited list of positive integers less than 365 that indicate the number of days for each test case.

Output

For each test case and on a separate line, you will print the number of days followed by a space, a dollar sign (`$`) and the amount of money that you would have at the end of the given number of days.

Example Input File

```
15 37 74 25
```

Example Output to Screen

```
15 $55
37 $213
74 $602
25 $119
```

9. Scrabble Swiss Seed Tournament

Program Name: Scrabble.java   Input File: scrabble.dat

A seeding method called Swiss Seeding was first used in Zurich, hence called Swiss Seed, to seed a chess tournament in 1895. This seeding method is frequently used in chess, squash, scrabble and other tournaments in which there is not enough time or facilities to play a round-robin type tournament to determine the seeding for the tournament. The Swiss Seeding process allows all players (or teams) to participate in several rounds of competition without being eliminated and without playing any person twice before being seeded for head-to-head single elimination. This method also allows players of similar ability to play each other in the initial rounds.

Your school is hosting a scrabble tournament in the near future, and as an incentive for players to attend, you want to guarantee them that they will be able to play at least four games before being eliminated. You are to write a program that will do the Swiss Seeding for your tournament. Your facility will accommodate no more than 40 players.

The process for this Swiss Seeding for the Scrabble tournament is:

1. Randomly select the order for the players in round one.
2. Players will play in pairs: player 1 will play player 2, player 3 will play player 4, etc.
3. After round one is finished, players will be seeded for the second round using the following rules:
    a. The players' scores are entered and the players are resorted, high to low, by their cumulative score which is the sum of all of the scores from all of the games the player has played so far.
    b. If two or more players are tied, these players are resorted by the reverse order of their seeding in the preceding round. For example, if Jake was seeded 10th and Alex was seeded 15th in round one and they tied for the 5th seed in round two, Alex would be seeded 5th and Jake would be seeded 6th because Alex had the lower seed in the preceding round.
    c. If a player is paired with player that he has already played (e.g. players A and B are paired with player A seeded higher than player B and they have previously played each other):
        - the highest seeded player below player B that player A has not yet played will move to player B's spot.
        - player B and all other players between player B and the player that moved up to player B's spot will be moved down the list while maintaining their order. For example, if player 3 had already played players 4 and 5 but had not player 6, player 6 would move up to the player 4 slot and play player 3 while the previous players 4 and 5 would move down the list and become players 5 and 6 respectively. This process will be repeated until every player is playing someone they have not previously played.
        - If the pair of players seeded last had previously played each other, the top seed in this last pairing will switch places with the lowest seed above him that his last place opponent has not played and whose opponent he has not yet played. For example, consider that players A and B described above are in the final pairing and have already played each other. Also assume that players J and K above him are paired together and player A has not played player J and player B has not played player K. Player A will switch places with player K so the final round will have players J and A playing against each other and players K and B playing against each other.
4. After the second round is complete, the seeding for the third round will be made using steps 2 and 3 above.
5. After the third round is complete, the seeding for round 4, which is the first elimination round, will be made using steps 2 and 3 above.

Input

The first line of input will contain a single integer `n` that indicates the number of test cases that you will have. The first line of each test case will contain a single even integer `m` that indicates the number of players in the tournament. Each of the following `m` lines in the test case will contain, in their initial random order, the unique first name of a player with no spaces followed by three positive integers in the range [25..600] representing the scores for that player for the first three rounds. Each of the items will be separated by a space.

Output

For each test case, on a single line you will print `TEST CASE #x ROUND y:` , where *x* is the test case number and *y* is the round number, followed by the names and cumulative points for the seeding order for rounds 2, 3, and 4 using the format below. Print a blank line after each test case.

9. Scrabble Swiss Seed Tournament (cont.)

Example Input File

```
1
10
Angel 245 324 278
Bonnie 345 234 430
Caleb 230 185 221
Don 245 172 213
Evie 192 97 220
Frank 215 200 221
George 345 286 275
Henry 420 235 249
Iris 345 289 276
Jack 128 158 387
```

Example Output to Screen

```
TEST CASE #1 ROUND 2:
Henry 420
Iris 345
George 345
Bonnie 345
Don 245
Angel 245
Caleb 230
Frank 215
Evie 192
Jack 128

TEST CASE #1 ROUND 3:
Henry 655
Bonnie 579
Iris 634
George 631
Angel 569
Frank 415
Don 417
Evie 289
Caleb 415
Jack 286

TEST CASE #1 ROUND 4:
Bonnie 1009
Iris 910
George 906
Angel 847
Henry 904
Jack 673
Caleb 636
Evie 509
Frank 636
Don 630
```

Note: A blank line at the end of the output is optional.

10. Sort, by George

Program Name: Sort.java   Input File: sort.dat

George wants to sort a list of words alphabetically, but not in normal alphabetical order. He has reassigned the order of the 26 letters of the alphabet to the order he thinks they should be. He needs you to write a program that will sort, in ascending order, a list of words using his alphabetic ordering system.

For example, if:

- George's alphabetic ordering system is the string `QWERTYUIOPASDFGHJKLZXCVBNM`
- He wants to sort the words:
```
PICK
KEYBOARD
PICKLE
ZEBRA
KEYS
ANTEATER
```
- The result would be:
```
PICK
PICKLE
ANTEATER
KEYS
KEYBOARD
ZEBRA
```

Input
The first line of input will contain a single integer `n` that indicates the number of sets of words to be sorted. For each set of words:

- The first line will contain a string of 26 distinct uppercase letters of the alphabet with no spaces. This string will be used as the alphabetic ordering system for the words in the set.
- The next line will be a single integer `m` ≤ 20 that will indicate the number of words to be sorted.
- The next `m` lines will contain the list of words to be sorted. Each of the `m` lines will contain a single word composed of 15 or fewer uppercase letters of the alphabet.

Output
For each set of input, you will print the list of words in increasing order using the alphabetic ordering system for that set. Print one word per line and print at least one blank line after each set.

Example Input File
```
1
POIUYTREWQASDFGHJKLMNBVCXZ
11
COMPUTER
SCIENCE
FUN
IS
SCIENTIFICALLY
FUNDAMENTAL
UIL
STATE
IS
CHALLENGING
AND
```

Example Output to Screen
```
IS
IS
UIL
AND
STATE
SCIENTIFICALLY
SCIENCE
FUN
FUNDAMENTAL
COMPUTER
CHALLENGING
```

Note: A blank line at the end of the output is optional.

11. Triangles

Program Name: Triangles.java   Input File: triangles.dat

You and your friend have been playing a game involving random sticks that you have found in your yard. You put these sticks in a pile and randomly choose three sticks and then see if you can form a triangle with them. If you can, you win one point. The stick is replaced and your friend does the same. You are curious about how many different combinations of sticks would allow you to win a point.

You remember from your Geometry class that the sum of the lengths of any two sides of a triangle must be greater than the length of the third side. You have decided to write a program to find out how many different combinations of the sticks in a pile could form a triangle.

Input

The first line of input will contain a single integer `n` that indicates the number of piles of sticks that will follow. Each of the following `n` lines will contain a space delimited list of five positive integers denoting the lengths of five sticks.

Output

For each pile of sticks, you will print the number of distinct combinations of sticks will form a triangle.

Example Input File

```
3
3 3 3 3 3
4 6 8 3 7
6 8 10 13 2
```

Example Output to Screen

```
10
8
4
```

12. Word Find 2

Program Name: WordFind2.java   Input File: wordfind2.dat

Most newspapers have a word find puzzle for their readers. A Word Find puzzle is a rectangular matrix of alphabetic letters and the player is given a list of words that appear in the puzzle either vertically, horizontally, or diagonally in any direction. The object of the game is for the player to locate in the puzzle all instances of a given word in the given word list. For our word find game, the letters in the word must be contiguous either vertically, horizontally, or diagonally in any direction and the same letter cannot be used more than once in a given instance of the word.

You are to write a solution to this word game.

Input

The first line of input will contain a single integer `n` that indicates the number of games to follow. For each game:

- The first line will contain two integers `r c` that indicate the number of rows and columns, respectively, of the matrix to follow.
- The next `r` lines will each contain `c` uppercase alphabetic characters.
- The last line will be a word list that contains an unknown number of space delimited words which may or may not appear in the puzzle.

Output

For each word in the word list, you will print the word and a space followed by the number of times the word appears in the puzzle.

Example Input File

```
2
7 7
RECSQSE
CRICOSC
EOPPIDT
NIMUTET
CVRETCN
ERFCVGE
MLOSECD
COMPUTER SCIENCE
6 8
BUNLHRTV
BLITOORD
OCSUSUNT
NTACKTOR
CESUDTDN
KTINSLIL
AUSTIN LUBBOCK HOUSTON
```

Example Output to Screen

```
COMPUTER 4
SCIENCE 1
AUSTIN 4
LUBBOCK 0
HOUSTON 7
```