1. How Much Fuel in the Tank?

Program Name: Fuel.java Input File: fuel.dat

John Hackworth, a nanotech engineer, has come up with a new system for creating fuel. The system involves multiple steps. Fuel plants grow to various heights, but always in increments of whole feet. Leaves grow on the plants in proportion to their height. The leaves are converted to raw fuel slush. (RFS) The raw fuel slush is then converted into pure distilled fuel. (PDF)

At the start of each day fuel plants are harvested. All of the plants on a given day will have the same height. Each plant produces 25 leaves per foot. The leaves need to cure for a day. Additionally, at the start of the day leaves from the previous day are converted to RFS. When turning the leaves into raw fuel it takes exactly 177 leaves to create 1 gallon of raw fuel slush. Fractions of gallons of raw fuel slush cannot be created so any extra leaves are set aside for the next day. The RFS must also cure for a day. Additionally, at the start of the day the RFS that has finished curing is converted into pure distilled fuel. It takes exactly 17 gallons of raw fuel liquid to create 1 gallon of pure distilled fuel. Fractions of gallons of PDF cannot be created so any extra gallons of raw fuel slush are set aside for the next day. Pure distilled fuel does not need to cure and is ready for use as soon as it is available. Assume every step in the process is performed at the start of a day. The next step in the process takes at the start of the next day and so forth.

Here is an example of the productions steps given a set of fuel plants. At the start of day 1 there are 100 fuel plants each 5 feet high.

Day	Fuel Plants and height	Leaves	Gallons of Raw Fuel Slush	Gallons of Pure Distilled Fuel	Comments
0	0	0	0	0	100 plants, 5 feet high at start.
1	100, 5	12500	0	0	Leaves harvested and curing.
2	0	110	70	0	Leaves boiled to form RFS and curing.
3	0	110	2	4	RFS converted to PDF.

The process may be complicated by having new fuel plants available for harvest at the start of the day. Consider the following example with new fuel plants each day:

Day	Fuel Plants and height	Leaves	Gallons of Raw Fuel Slush	Gallons of Pure Distilled Fuel	Comments
0	0	0	0	0	100 plants, 5 feet high at start.
1	100, 5	12500	0	0	
2	100, 5	12610	70	0	
3	80, 5	10043	73	4	
4	50, 5	6381	61	8	
5	0	9	46	11	
6	0	9	12	13	

Write a program that shows the cumulative total number of gallons of pure distilled fuel that have been created just after the daily processing step is completed for a given number of days based on the number of fuel plants and their height available at the start of each day.

Input

- The first line will contain a single integer n that indicates the number of data sets.
- The second line of a data set will be a single positive integer d indicating the number of days in the data set.
- The third line will contain an pairs of integers p and h. p represents the number of plants and will be non negative. h represents the height of those plants and will be greater than 0. Each pair of integers will indicate the number of fuel plants available for harvest at the start of a day and the height of each tree that day. The first pair of numbers represents the number of plants and their individual heights at the start of day 1, the next pair represents the number of plants and their individual heights at the start of day 2 and so forth. The number of pairs may be less than, greater than, or equal to the number of days for the data set. Any extra or unneeded data is ignored. If the number of pairs is less than the number of days assume there are 0 fuel plants per day after the last pair of data.

Output

- For each data set print out `Data Set <N>` on a line by itself where `<N>` is the number of the data set.
- On the next line print out `Day 1 <PDF1>, Day 2 <PDF2>, Day 3 <PDF3>,` and so forth up to `Day <d> <PDFd>` The `<PDF>` values are the total number of gallons of pure distilled fuel produced so far, just after that day's processing steps are completed. `<d>` is equal to the number of days specified for this data set.

Example Input File

```
3
5
100 5
6
100 5 100 5 80 5 50 5
10
100 5 80 4 20 15 80 4 100 6 75 5 75 6 75 4 80 4
```

Example Output To Screen

```
Data Set 1
Day 1 0, Day 2 0, Day 3 4, Day 4 4, Day 5 4
Data Set 2
Day 1 0, Day 2 0, Day 3 4, Day 4 8, Day 5 11, Day 6 13
Data Set 3
Day 1 0, Day 2 0, Day 3 4, Day 4 6, Day 5 9, Day 6 11, Day 7 16, Day 8 20,
Day 9 23, Day 10 26
```

Note, the last line of output would be on one line, but has been wrapped when printed out. Do not wrap your output.

2. Longhorns Runneth Over

Program Name: Longhorn.java Input File: None

Hook 'em!! Welcome to UT, home of the longhorns. The mascot for UT sports is the Longhorn, personified by Bevo. The Bevo that currently appears at home football games is the 14th to serve in that capacity. Write a program to display an ASCII image of Bevo repeated 14 times.

Input
There is no input file for this problem.

Output
A total of 14 longhorn pictures, 2 of which are shown below. Each picture is exactly as shown with exactly one line after it with the number for that Bevo, 1 through 14.

Example Input File
None

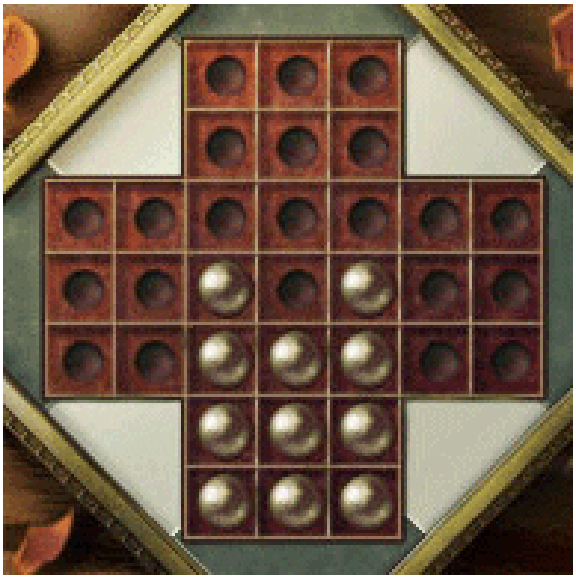
Example Output To Screen:
```
|**********************|
|#######........#######|
|....####......####....|
|.......########.......|
|......##########......|
|........######........|
|..........##..........|
|..........##..........|
|..........##..........|
************************
1
|**********************|
|#######........#######|
|....####......####....|
|.......########.......|
|......##########......|
|........######........|
|..........##..........|
|..........##..........|
|..........##..........|
************************
2
```

Followed by 12 more of the same image and corresponding numbers for a total of 14.

3. Losing Your Marbles

Program Name: Marbles.java **Input File: marbles.dat**

Biscuit House is country themed restaurant that keeps a game on all of its tables for customers to play while waiting. The game takes place on a board with marbles. Marbles can jump other marbles according to specific rules. The goal of the game is to end up with a single marble left on the board. The games uses a board such as this:



The board is represented as rectangular 2d array of chars. Open spaces will indicated with an o. Marbles will be indicated with an m. Cells that cannot be moved into will be represented with an x. The above board will be represented with a 7 by 7 array of chars:

x	x	o	o	o	x	x
x	x	o	o	o	x	x
o	o	o	o	o	o	o
o	o	m	o	m	o	o
o	o	**m**	m	m	o	o
x	x	m	m	m	x	x
x	x	m	m	m	x	x

Note this is simply one example. The board size, arrangement, and initial marble positions will vary.

Marbles can jump over adjacent marbles that are located up, down, left, or right of the marble as long as there is an open spot on the other side of the marble being jumped over. Marbles may not be moved into spaces with an x, spaces that already contain another marble or off the board. When a marble is jumped it is removed from the board. In the example above one of the legal moves would be to take the bolded marble and move it over the marble above it leading to the following configuration where the marble that was jumped over has been removed.

x	x	o	o	o	x	x
x	x	o	o	o	x	x
o	o	**m**	o	o	o	o
o	o	o	o	m	o	o
o	o	o	m	m	o	o
x	x	m	m	m	x	x
x	x	m	m	m	x	x

A move consists of three changes:

1. Picking up the source marble from its spot, which becomes open.
2. Moving the source marble to its destination.
3. Removing the marble the source marble passed over from the board resulting in an open space.

If there are no legal moves left the game is over. If there is one marble left on the board it is a win. If there are two or more marbles left on the board it is a loss.

Write a program that given a board and an initial configuration of marbles determines if it is possible to win and if not what is the fewest possible marbles that will be left.

Input

- The first line will contain a single integer n that indicates the number of data sets.
- The first line of each data set will contain 2 integers r and c separated by a single space. r indicates the number of rows and c indicates the number of columns the board in this data set has. r and c will both be greater than 0 and less than 8.
- The next r lines will be the initial configuration of the board for this data set.
- Each line of the next r lines will contain c characters. All characters will be either `m`, for a marble, `o` for an initially open spot, or `x` for a spot that cannot be moved to.

Output

- For each data set print out `Data Set <N> <Result>` where `<N>` is the number of the data set.
- `<Result>` will replaced by one of two options:
    - `SOLVED` if it is possible to make a series of moves from the initial configuration so that only one marble remains on the board
    - `NOT SOLVED <L>` if it is not possible to make a series of moves from the initial configuration so that only one marble remains on the board. `<L>` will be replaced with the smallest possible number of marbles left on the board given the initial configuration.

Example Input File

```
5
7 7
xxoooxx
xxoooxx
ooooooo
oomomoo
oommmoo
xxmmmxx
xxmmmxx
1 5
mmomo
5 5
mmomm
oxooo
oxomm
oxooo
mmomm
7 7
xxomoox
xommmox
ommmmmo
mmmommm
ommmmmo
oxmmmxx
oxomooo
11 7
```

```
ooooooo
oooMooo
ooMMMMo
oooMooo
ooooooo
xxxxxxx
ooooooo
oooMooo
ooMMMMo
oooMooo
ooooooo
```

Example Output To Screen

```
Data Set 1 SOLVED
Data Set 2 SOLVED
Data Set 3 NOT SOLVED 5
Data Set 4 SOLVED
Data Set 5 NOT SOLVED 2
```

4. Lucas Numbers

Program Name: Lucas.java Input File: None

The Lucas numbers are a series of numbers closely related to the Fibonacci numbers. The nth Lucas number, written L_n, is determined as follows:

$$L_n := \begin{cases} 2 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ L(n-1) + L(n-2) & \text{if } n > 1. \end{cases}$$

Thus L_0 equals 2, L_1 equals 1, L_2 equals 3 (2 + 1), L_3 equals 4 (1 + 3), L_4 equals 7 (3 + 4), and so forth. Write a program that prints out the first 31 Lucas numbers, that is L_0 to L_{30}. The 31st Lucas Number, L_{30}, equals 1860498.

Input
There is no input

Output
Output the first 31 Lucas numbers, one number per line.

Example Input File
```
None
```

Example Output To Screen
```
2
1
3
4
7
```

The next 26 values are not shown in this example, but your program is to output them.

5. Mister Mxyzptlk's Mix-up

Program Name: Mixup.java Input File: mixup.dat

Mister Mxyzptlk is a real trickster. He likes turning things around. Whenever you say something he repeats what you said, but says each word backwards. How infuriating. Write program to display the results of Mister Mxyzptlk's mix-up.

Input

- The first line will contain a single integer n that indicates the number of data sets.
- Each data set will consist of a single line
- Each line will contain one or more words separated by a single space.
- Words will consist of one or more lower case letters, a through z.
- There will be no characters besides lower case letters in words and a single space between words. There will be no space following the last word in a line.
- Every line will contain at least one word. If a line contains a single word there will not be any spaces in the line.

Output

- For each line print out the words in the same order as they appear in the input but print out the characters in each word in reverse order.

Example Input File

```
3
what starts here changes the world
hook em
the eyes of texas are upon you
```

Example Output To Screen

```
tahw strats ereh segnahc eht dlrow
kooh me
eht seye fo saxet era nopu uoy
```

6. My New Box Set

Program Name: Boxes.java Input File: boxes.dat

You have finished unpacking your stuff as new freshmen at UT. You now have a bunch of leftover cardboard boxes. You want to know what the volume of your boxes is. Each box is a rectangular box whose volume can be calculated by multiplying its width times its height times its length.

Write a program to determine the total volume of the boxes you used and that prints out the volume of the largest and smallest boxes in each data set given the length, width, height of each box.

Input

- The first line will contain a single integer n that indicates the number of data sets.
- The first line in each data set will be a single integer m indicating how many boxes are in this data set. m will be greater than 0 and less than 100.
- The next m lines in a data set will consist of 3 integers, w h l. Each line indicates the dimensions of 1 box in the data set. w indicates the width, h indicates the height, and l indicates the length of the box.
- All integers w, h, and l will be greater than 0 and less than or equal to 100.

Output

- The output for each data set will consist of 3 lines:
  - For each data set print out `Box Set N` where N is the number of that data set.
  - On the next line print out the total volume of the boxes in that data set.
  - On the next line print out the volume of the largest box in the data set followed by a comma and a single space followed by the volume of the smallest box in the data set.

Example Input File

```
3
1
5 2 4
3
1 1 1
2 2 2
10 100 10
2
1 10 1
5 3 4
```

Example Output To Screen

```
Box Set 1
40
40, 40
Box Set 2
10009
10000, 1
Box Set 3
70
60, 10
```

7. Practically Done?

Program Name: Practical.java Input File: practical.dat

Write a program to determine if a given number is a *practical number* or not. An integer n is a practical number if all smaller positive integers can be expressed as the sum of distinct divisors of n.

Consider the integer 12. The divisors of 12 are 1, 2, 3, 4, and 6. In order for 12 to be a practical number we must be able to represent all integers from 1 to 11 as sums of those 5 divisors without reusing any divisor more than once in the sum for a particular value. Here are possible representations for the values 1 through 11 using the 5 divisors of 12:

1 = 1 2 = 2
3 = 3 4 = 4
5 = 1 + 4 6 = 6
7 = 1 + 6 8 = 2 + 6
9 = 3 + 6 10 = 1 + 3 + 6
11 = 2 + 3 + 6

Therefore 12 is a practical number.

14 is not a practical number. Its divisors are 1, 2, and 7. Here is an attempt to represent the values 1 through 13 using the 3 divisors of 14.

1 = 1 2 = 2 3 = 2 + 1 4 = ?

There is no way to represent 4 as the sum of the divisors since divisors may not be reused for a given integer. Therefore 14 is not a practical number.

Input

- The first line will contain a single integer n that indicates the number of data sets. $1 \leq n \leq 30$
- The next n lines will each contain a single positive integer. Each integer will be less than or equal to 1,000,000.

Output

- For each integer in the input file print out `PRACTICAL` if the integer is a practical number or `NOT PRACTICAL` if the integer is not a practical number.

Example Input File

```
6
12
14
1
2
17442
998
```

Example Output To Screen

```
PRACTICAL
NOT PRACTICAL
PRACTICAL
PRACTICAL
PRACTICAL
NOT PRACTICAL
```

8. Rock Paper Scissors … Lizard Spock??

Program Name: Rock.java Input File: rock.dat

Rock, Paper, Scissors, Lizard, Spock is an extension of the class rock, paper, scissors game between two players. The extension was proposed by Sam Kass, a software developer who works for General Dynamics, and Karen Bryla and made famous by its appearance on the television show *The Big Bang Theory*. This version of the game adds two weapons, Lizard and Spock. Each player picks one of the five weapons and the outcome is determined based on those choices. If the same weapon is chosen by each player the round is a tie. If they are different the outcome is determined as follows. The rationale for the outcome is in parenthesis.

Scissors beats Paper (`Scissors CUTS Paper`)
Paper beats Rock (`Paper COVERS Rock`)
Rock beats Lizard (`Rock CRUSHES Lizard`)
Lizard beats Spock (`Lizard POISONS Spock`)
Spock beats Scissors (`Spock SMASHES Scissors`)
Scissors beat Lizard (`Scissors DECAPITATES Lizard`)
Lizard beats Paper (`Lizard EATS Paper`)
Paper beats Spock (`Paper DISPROVES Spock`)
Spock beats Rock (`Spock VAPORIZES Rock`)
Rock beats Scissors (`Rock CRUSHSES Scissors`)

Write a program that determines the outcome of a series of games between two players of Rock, Paper, Scissors, Lizard, Spock. Each data set (game) will consist of a series of rounds. Each round players are supposed to pick one of the five weapons although it is possible players will pick an invalid weapon. Each player's pick will be represented by a 3 by 3 matrix of characters consisting of hyphens, `(-)` and asterisks `(*)`. The five valid weapons are represented as follows:

```
Rock       Paper      Scissors       Lizard     Spock
***        ---        **-            -**        *-*
***        ***        --*            *-*        *-*
***        ---        **-            -*-        -*-
```

Any 3 by 3 matrix that does not match one of the five weapons is an invalid weapon. Even if the matrix could be rotated to form a valid weapon, it is still invalid. For example this matrix

```
-*-
-*-
-*-
```

does not show a valid weapon, even though it could be rotated 90 degrees to form Paper.

Input

- The first line will contain a single integer n that indicates the number of data sets (games).
- The first line in each data set will be a single positive integer r that represents the number of rounds in the data set.
- The next 3r lines will represent the players' choices for the rounds in the data set.
- Each set of three consecutive lines will represent the two player's choices for that round.
    - The first line of the choice data will be seven characters.
    - The first three characters in the first line will be either hyphens and / or asterisks and will represent the first row of player 1's choice.
    - The next character will be a space
    - The next three characters in the first line will be either hyphens and / or asterisks and will represent the first row of player 2's choice.
    - The second and third lines of the data for a round will follow the same format as the first line and will represent the second and third lines in the players' choice matrix.

Output

- The output for each data set will consist of r + 2 lines where r is the number of rounds in the data set.
- The first line of output for each data set will be `Game <N>:` where `<N>` is the number of that data set.
- For each round in a data set print out the result of the round.
    - If both players pick valid weapons print out the result of the round.
    - If there is a winner print out the corresponding result shown in the parenthesis in the problem explanation regardless of which player picked the winning weapon. Follow the explanation with a period, then a single space, and then the phrase `Player <X> wins.` where `<X>` is either `1` or `2` depending on which player picked the winning weapon.
    - If both players picked the same, valid weapon print out `Both players picked <W>. Tie.` where `<W>` is the name of the weapon both players picked.
    - If one player's 3 by 3 matrix does not match a valid weapon that player loses the round. In this case print out `Player <X> picked an invalid weapon. Player <Y> wins.` where `<X>` is the number of the player that picked the invalid weapon and `<Y>` is the number of the player that picked the valid weapon.
    - If both players' 3 by 3 matrices are invalid print out `Both players picked invalid weapons. Tie.`
- The last line of each data set will be
  `Player 1: <R>, Player 2: <S>, Ties: <T>. Result: <RESULT>.`
  Where `<R>` is the number of rounds player 1 won in the data set, `<S>` is the number of rounds player 2 won in the data set, and `<T>` is the number ties in the data set. `<RESULT>` will be `Player 1 wins` if player 1 won more rounds than player 2 in the data set, `Player 2 wins` if player 2 won more rounds than player 1 in the data set, or `Tie` if each player won an equal number of rounds in the data set.

Example Input File

```
2
3
*** ***
*** ___
*** ___
___ *_*
___ *_*
___ _*_
___ ___
_** **_
_** __*
7
*_* _**
*_* *_*
_*_ _*_
*_* ***
*_* ***
_*_ ***
___ ___
*** ***
___ ___
_*_ **_
_*_ __*
_*_ **_
**_ ___
__* ***
**_ ___
___ *_*
*** *_*
___ _*_
_** ___
*_* ***
_*_ ___
```

Example Output To Screen

```
Game 1:
Player 2 picked an invalid weapon. Player 1 wins.
Player 1 picked an invalid weapon. Player 2 wins.
Both players picked invalid weapons. Tie.
Player 1: 1, Player 2: 1, Ties: 1. Result: Tie.
Game 2:
Lizard POISONS Spock. Player 2 wins.
Spock VAPORIZES Rock. Player 1 wins.
Both players picked Paper. Tie.
Player 1 picked an invalid weapon. Player 2 wins.
Scissors CUTS Paper. Player 1 wins.
Paper DISPROVES Spock. Player 1 wins.
Lizard EATS Paper. Player 1 wins.
Player 1: 4, Player 2: 2, Ties: 1. Result: Player 1 wins.
```

9. Score Update

Program Name: Scores.java **Input File:scores.dat**

This is Texas! There has to a football question, right? Write a program to determine the winner of a football game. The only information you need are the scoring plays each team makes. These will be represented with upper case characters. The scoring play, character codes, and points each scoring play is worth are as follows:

Scoring Plays	Character Code	Points Score is Worth
Touchdown	T	6
Point after attempt made, kicked	K	1
Point after attempt made, pass or run	W	2
Point after attempt blocked, returned to other goal by defense	B	2
Field Goal	F	3
Safety	S	2

Write a program that given the names of two teams and a list of the scores they made in a football game determines the winner of the game. Note, your program DOES NOT need to ensure point after attempts occur after touchdowns.

Input

- The first line will contain a single integer n that indicates the number of data sets.
- Each data set will consist of 4 lines
  - The first line of a data set will be the name of the first team in a game.
  - The second line of a data set will only contain the characters T, K, W, B, F, and S indicating the scores the team made in the game. There will be no spaces between characters. If the team did not make any scores the line will be blank.
  - The third and fourth lines of a data set will contain the name of the second team and the scores it completed in the game against the first team.

Output

- For each data set print out `<Name1> <Score1> <Name2> <Score2> <Winner> wins!` where `<Name1> <Score1>` are the name and score for the first team in the data set, `<Name2> <Score2>` are the name and score for the second team in the data set, and `<Winner>` is the team that had the higher score at the end of the game. There will never be ties.

Example Input File

```
4
Stanford
F
Baylor

Texas
TKTKSFTKTW
Texas A & M
F
Texas
TKTKTFSTWTK
Texas Tech
BTKTK
Rice

Navy
FFFFF
```

Example Output To Screen

```
Stanford 3 Baylor 0 Stanford Wins!
Texas 34 Texas A & M 3 Texas Wins!
Texas 40 Texas Tech 16 Texas Wins!
Rice 0 Navy 15 Navy Wins!
```

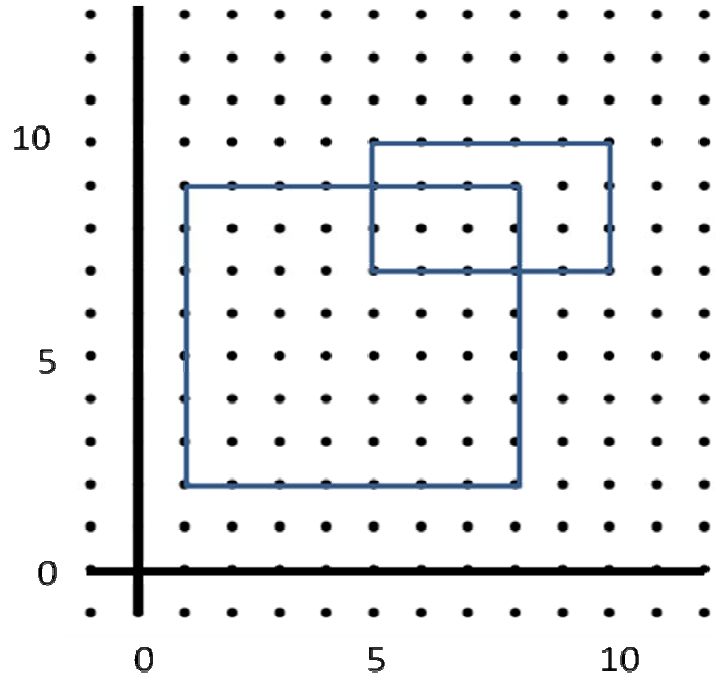
10. Shared Interests

Program Name: Shared.java **Input File: shared.dat**

Given the coordinates for a pair of opposite corners of one rectangle and the coordinates for a pair of opposite corners of another rectangle, determine the area of the overlap between the rectangles. The coordinates of the rectangle's corners will be expressed as x and y coordinates. All coordinates will be integers. The coordinates for the pair of opposite corners will be expressed as $\mathbf{x_1\ y_1\ x_2\ y_2}$.

$x_1\ y_1$ will specify the coordinates of one corner of the rectangle and $x_2\ y_2$ will specify the opposite corner. Which set of corners is used for each rectangle (upper-right and lower-left OR upper-left and lower-right) will vary. Likewise the order of coordinates for a particular rectangle can vary. (It could be upper-right, lower-left OR lower-left, upper-right) You must determine which corners the coordinates represent for each rectangle.

The rectangles are positioned on a traditional Cartesian coordinate system. The example below shows a rectangle with corners at (10, 10) and (5, 7) and another rectangle with corners at (8, 2) and (1, 9). The overlap is a total of 6 units.



Input

- The first line will contain a single integer n that indicates the number of data sets. The next 2n lines will be the data sets for the problem.
- Each data set will consist of two lines
- The first line in a data set will consist of 4 integers $x_1\ y_1\ x_2\ y_2$ specifying the coordinates of a pair of opposite corners of the first rectangle.
- The second line in a data set will consist of 4 integers $x_3\ y_3\ x_4\ y_4$ specifying the coordinates of a pair of opposite corners of the second rectangle.
- All integers will be greater than or equal to -200 and less than or equal to 200
- All rectangles given will have an area of at least 1.

Output

- For each data set print out the size of the intersection between the two rectangles in the data set as an integer.

Example Input File

```
3
10 10 5 7
8 2 1 9
-5 -4 1 -1
1 5 2 4
-8 -5 8 8
4 4 -5 -2
```

Example Output To Screen

```
6
0
54
```

11. UT Towers

Program Name: Towers.java **Input File: towers.dat**

Shelby is playing with her blocks building towers. (She is building replicas of the UT Tower because she wants to be a Longhorn some day.) All of the blocks have a base of 1 inch by 1 inch, but they have various heights, always in increments of 1 inch. Shelby is trying to determine for a given a collection of blocks of various heights and a target number of towers, what is the highest equal height she can make all of the towers. The blocks have to be placed straight up with the 1 by 1 base on the bottom. In other words a 1 by 1 by 3 inch block can't be laid on its side to have a height of 1. It must be placed straight up with a height of 3.

So for example if Shelby wants to build 3 towers and has blocks of heights 1, 1, 1, 2, 2, 2, 2, 3, 3, 6, and 15 what is the largest height she can make each tower so that the towers are all of equal heights? Blocks cannot be split, reused or laid on their sides. It is not necessary to use all of the blocks.

In this case the maximum height that can be achieved for each tower is 7. One answer uses the following blocks:

Tower 1: (1, 6) -> total height of 7
Tower 2: (1, 3, 3) -> total height of 7
Tower 3: (1, 2, 2, 2) -> total height of 7

This results in two unused blocks of size 2 and 15.

Input

- The first line will contain a single integer n that indicates the number of data sets.
- The first line in each data set will be a two integers t and b separated by a single space. t indicates the number of towers to build for this data set. $0 < t < 6$. b represents the number of blocks in the data set. $0 < b < 12$
- The second line in the data set will be b integers separated by a single space. The integers represent the height of each individual block. Every integer will be greater than 0 and less than 26.

Output

- For each data set print out `Data Set <N> Maximum Possible Height <H>.` where `<N>` is the number of the data set and `<H>` is the maximum possible height for the number of towers in the data set and the blocks in this data set. Recall each tower in the data set must be equal in height to all of the other towers in the data set.

Example Input File

```
4
3 11
1 1 1 2 2 2 2 3 3 6 15
3 5
7 6 1 4 2
1 5
3 15 1 10 4
5 11
2 2 1 2 3 6 2 3 15 1 1
```

Example Output To Screen

```
Data Set 1 Maximum Possible Height 7
Data Set 2 Maximum Possible Height 0
Data Set 3 Maximum Possible Height 33
Data Set 4 Maximum Possible Height 3
```

12. Win or Lose

Program Name: Wins.java Input File: wins.dat

A statistic often used to describe a baseball team's record during the season is "games over five hundred." A team's record consists of the number of games they have won and lost. A team's winning percentage is the number of games won divided by the total number of games rounded to the thousandths place. A winning percentage of .500, usually spoken "five hundred", indicates the team has won and lost an equal number of games. A team is said to be "over five hundred" if they have won more games than they have lost and said to be "under five hundred" if they have lost more games than they have won. If a team is over five hundred they are a certain number of "games over five hundred." The number of games over five hundred is equal to the number of games the team has won minus the number they have lost. Conversely if a team is under five hundred the number of games under five hundred is equal to the number of game lost minus the number of games won.

Given an input String that represents the results of a team's games, in the order they are played, display the number of games over or under five hundred after all the games have been completed.

Input

- The first line of input will be a single integer `n` indicating the number of data sets to process.
- The first line of each data set will be two integers, `t` and `g` separated by a single space.
  - `t` indicates the number of teams in this data set. $0 < t < 30$
  - `g` indicates the number of games each team in this data set played. $0 < g < 163$
- The next `2t` lines will be the names and win-loss data for each team in the data set.
- The first line of a team's data will be the name of the team.
- The second line of a team's data will be `g` characters all equal to `W` or `L`. A `W` indicates the team won that game and an `L` indicates the team lost that game.

Output

- For each data set output `Data set n` where n is the number of that data set.
- The output for each team in a data set will consist of two lines:
  - The first line consists of the name of the team.
  - The second line will be the team's relation to 500 after all games have been played. There are three possible relationships:
    - `finished M games over 500` (Where `M` is the number of games over 500 the team finished. If `M` is one display `game` instead of `games`.)
    - `finished M games under 500` (Where `M` is the number of games under 500 the team finished. If `M` is one display `game` instead of `games`.)
    - `finished at 500` (Indicating a team won and lost an equal number of games.)
- Print out a single blank line after the last team in every data set, including the last data set.

Example Input File

```
2
2 5
Chicago Cubs
LLLWW
St. Louis Cardinals
WWWWW
3 10
Chicago Cubs
WWWWLLLLLW
New York Mets
WLLWLLWLWL
Texas Rangers
LWWLLLLLLW
```

Example Output To Screen

```
Data Set 1
Chicago Cubs
finished 1 game under 500
St. Louis Cardinals
finished 5 games over 500

Data Set 2
Chicago Cubs
finished at 500
New York Mets
finished 2 games under 500
Texas Rangers
finished 4 games under 500
```