2012 University of Virginia High School Programming Contest

Welcome to the 2012 University of Virginia High School Programming Contest. Before you start the contest, please be aware of the following notes:

The Contest

1. There are ten (10) problems in the packet, using letters A-J. These problems are NOT sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Baloon Color
A	Grading Exams	Yellow
B	Reducing Improper Fractions	Green
C	Ant Entrapment	Red
D	Math Tutoring	Black
E	Where's the Rainbow	Orange
F	Painting Party	Blue
G	Back and Forth	Pink
H	Hoo's Afraid of the Big Bad Wolf?	White
I	Canyon Crossing	Purple
J	Yahtzee	Gray

2. Solutions for problems submitted for judging are called runs. Each run will be judged.

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Yes	Your submission has been judged correct.
No - Wrong Answer	Your submission generated output that is not correct or is incomplete.
No - Output Format Error	Your submission's output is not in the correct format or is misspelled.
No - Excessive Output	Your submission generated output in addition to or instead of what is required.
No - Compilation Error	Your submission failed to compile.
No - Run-Time Error	Your submission experienced a run-time error.
No - Time Limit Exceeded	Your submission did not terminate within one minute.

3. A team's score is based on the number of problems they solve and penalty minutes, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty minutes are issued equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty minutes are added for problems that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty minutes.

4. This problem set contains sample input and output for each problem. However, the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. One challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive a "wrong answer" judgment, you should consider what other datasets you could design to further evaluate your program.

5. In the event that you think a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient; no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still think there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

   You may not submit clarification requests asking for the correct output for inputs that you provide. Sample inputs may be useful in explaining the nature of a perceived ambiguity, e.g., "There is no statement about the desired order of outputs. Given the input: . . . , would not both this: . . . and this: . . . be valid outputs?".

   If a clarification is issued during the contest, it will be broadcast to all teams.

6. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first.

   Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams. This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

Your Programs

8. All solutions must read from standard input and write to standard output. In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. The judges will ignore all output sent to standard error (cerr in C++ or System.err in Java). You may wish to use standard error to output debugging information. From your workstation you may test your program with an input file by redirecting input from a file:

```
program < file.in
```

9. All lines of program input and output should end with a newline character (`\n`, `endl`, or `println()`).

10. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

11. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

12. Unless otherwise specified, all numbers in your output should begin with a - if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point should be followed by as many decimal digits as can be printed. This means that for floating point values, use standard printing techniques (cout and System.out.println). The judging will check your programs with 10^{-3} accuracy, so only consider the sample output up until that point.

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure you do not round or use a set precision.

13. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no fractional component. Scientific notation will not be used in input sets unless a problem statement explicitly specifies it.

Good luck, and HAVE FUN!!!

A. Grading Exams

Ms. Garrette needs help grading her multiple choice exams. To prevent cheating, she gave each student an individualized exam. She wants to write a program that, given an answer key and a particular student's responses, calculates the number of incorrect answers. Can you help her?

Input

The first line of input is the number of test cases that follow. Each test case starts with an integer L ($0 < L \leq 100$) representing the number of questions on the exam. The next line contains the answer key, where each question is represented by a single letter (i.e. a, b, c, or d) corresponding to the correct answer. The following line contains the student's responses in the same format.

Output

For each case, output the line "Case x:" where x is the case number, on a single line. This is followed by the number of student responses that did **not** match the answer key.

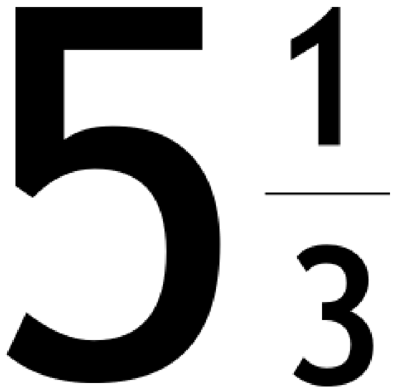
Sample Input

```
2
5
abadd
abada
3
cba
abc
```

Sample Output

```
Case 1:  1
Case 2:  2
```

B. Reducing Improper Fractions



You were invited to a St. Patrick's Day party, but you aren't allowed to leave until you finish your fraction reduction homework. You want to go to the party as soon as possible so you decide to write a program to do your homework for you. Given an improper fraction, calculate its reduced form.

Input

The first line of input is the number of test cases that follow. Each test case is a line containing two integer values. The first integer value will be the numerator n ($0 \leq n \leq 10^9$), the second integer is the denominator d ($0 < d \leq 10^9$).

Output

For each case output the line "Case x:" where x is the case number, on a single line, followed by a space, and then proper fraction. Each fraction will be of the form "**I N/D**", where **I** is the integer part, **N** is the numerator of the fractional part, and **D** is the denominator of the fractional part. If the integer value is less than than 0, only output "**N/D**". If both the integer value and the reduced numerator are zero, output "**0**". If there is no fractional part, only output "**I**".

Sample Input

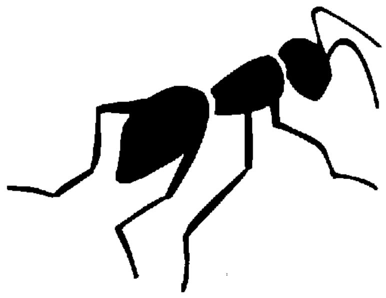
```
4
301 100
89 39
50 25
25 50
```

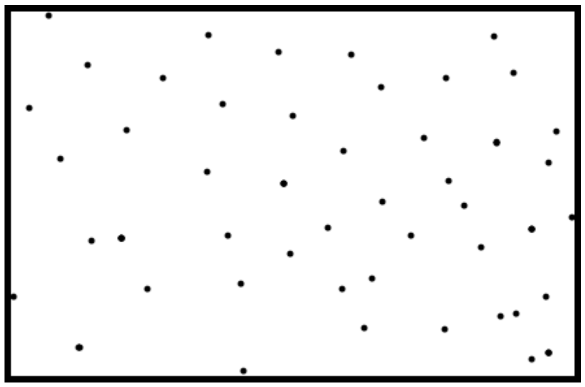
Sample Output

```
Case 1:  3 1/100
Case 2:  2 11/39
Case 3:  2
Case 4:  25/50
```

C. Ant Entrapment

Recently, your friend Oscar purchased an ant farm. He accidentally let the ants loose on his floor and now they're crawling everywhere! You want to fence them off with a single continuous rectangle so that they're not running amok. However, due to technical limitations your fence pieces must be aligned with the X and Y axes. The figure below shows an example of a fence around a set of ants (slight offsets from the border are just for visual effect).





What is the perimeter and area of the smallest fence that contains all of the ants?

Input

The first line of input is the number of test cases that follow. Each test case starts with an integer N $(1 \leq N \leq 100)$ on a line by itself representing the number of ants. The following N lines of input contain two floating-point values X and Y $(-1000.0 \leq X, Y \leq 1000.0)$ representing the position of an ant. You can assume that ants are single points–they have no area.

Output

For each case output the line "Case x:" where x is the case number, on a single line, followed by the string "Area" and the area of the fence as a floating-point value and then a comma, followed by a space and then "Perimeter" and the perimeter of the fence as a floating-point value.

Sample Input

```
2
3
-1.000 0.000
5.000 11.500
3.200 -4.250
2
2.125 0.500
6.875 9.100
```

Sample Output

```
Case 1:  Area 94.5, Perimeter 43.5
Case 2:  Area 40.85, Perimeter 26.7
```

D. Math Tutoring

You are helping a friend with the rule for taking the derivative of a polynomial, but he just can't seem to get it! You've gone over many examples, and finally you decide to just write a program to compute the derivatives for him.

Recall that a polynomial of the form:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

has as its derivative:

$$n a_n x^{n-1} + (n-1) a_{n-1} x^{n-2} + \ldots + 2a_2 x + a_1$$

For example, the derivative of $2x^3 - x + 3$ is $6x^2 - 1$. Likewise, the derivatave of $3x^4 + 2x^3 + 7x^2 + 5x + 7$ is $12x^3 + 6x^2 + 14x + 5$.

Given a polynomial, provide the derivative. We are only using polynomials of the form presented here.

Input

The rst line of input is the number of test cases that follow.

Each input case appears on a single line, and will start with a single integer, n $(1 \leq n \leq 100)$, which is the highest exponent of the polynomial. $n + 1$ values will follow, which are the coefficients of the terms x^n down to $x^0 = 1$, respectively. All coefficients will be integers between -1000 and 1000, inclusive. The highest exponent will always be positive. All numbers will be separated by a single space.

Output

For each case, output the line "Case x:" where x is the case number, on a single line. The output polynomial is to be formatted in the same manner as the input: the first value being the highest polynomial, and the successive values being the coefficients for the individual terms. Each output case should be on one line, with the values separated by one space.

Sample Input

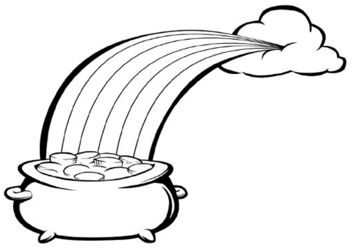
```
4
3 2 0 -1 3
4 3 2 7 5 7
5 6 5 4 3 2 1
1 5 10
```

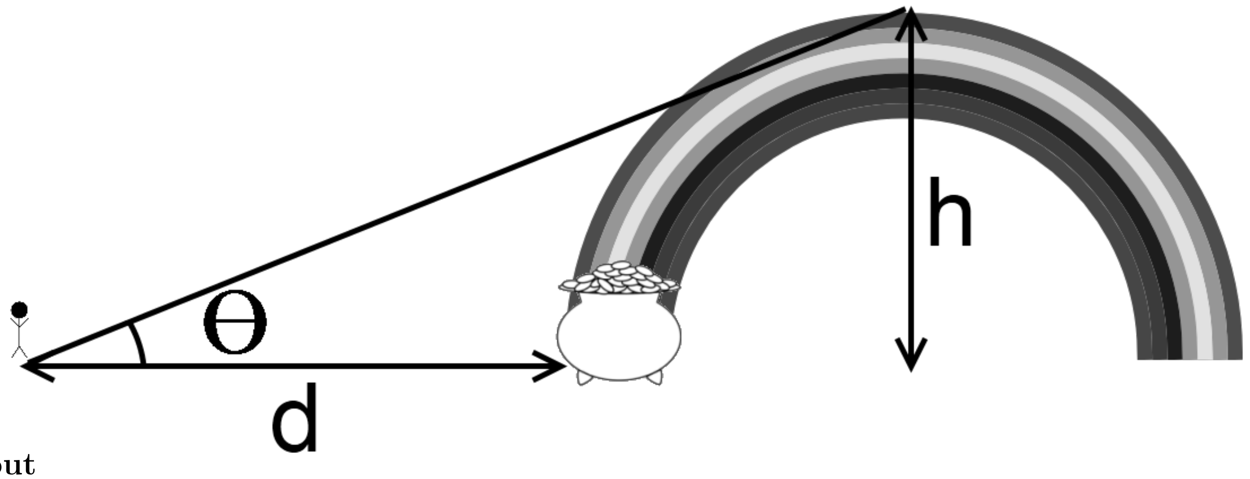
Sample Output

```
Case 1:  2 6 0 -1
Case 2:  3 12 6 14 5
Case 3:  4 30 20 12 6 2
Case 4:  0 5
```

E. Where's the Rainbow

After a long day of working on HSPC problems, you see a rainbow in the sky and want to figure out how far you have to walk to get your pot of gold. You are given a 2D plane and a semi-circle (rainbow) of height h placed with the center somewhere along the x-axis. You are also given an angle of inclination to view the top of the rainbow from the origin. Find how far away the closest point on the rainbow is to you by using the picture below to solve for d, the distance between you and the closest point on the rainbow. You can ignore the height/width of the person; treat them as a point.





Input

The first line of input is the number of test cases that follow. Each successive line represents a single test case, and will be composed of two floating point numbers, separated by a single space. The first value is h ($1 \leq h \leq 10^5$), the height of the rainbow at its highest point. The second is the angle of inclination θ ($0 < \theta < 90$).

Output

For each case output the line "Case x:" where x is the case number, on a single line, followed by d, the distance from you to the rainbow.

Sample Input

```
2
50.0 45.0
100.0 75.0
```

Sample Output

```
Case 1:  0.00
Case 2:  73.20508
```

F. Painting Party

A start-up company wants to hire you to write a painting program. As part of your interview, they have asked you to write a program that can draw filled and empty rectangles of different colors on a square grid of pixels. An empty rectangle will have a border that is one pixel thick. If a new rectangle is requested, it should completely overwrite what, if anything, was in that area.

Input

The first line of input is the number of test cases that follow. Each test case starts with an integer N ($1 \leq N \leq 100$) on a line by itself which is the width and height of the pixel grid; all pixel grids are square. The next line contains an integer M ($0 \leq M \leq 100$) which represents the number of rectangles to draw. The next M lines start with either the string "Filled" or "Empty" stating whether to draw a filled or an empty rectangle. This is followed by 4 space-separated integers: X Y W H where X and Y are the position of the bottom-left corner of the rectangle, W is the width of the rectangle, and H is the height. This is then followed by a space and then a single character C, representing the color to draw the rectangle. C will be an uppercase letter between A and Z. X is the horizontal axis, meaning the number of characters from the left. Y is the vertical axis, meaning the number of characters from the bottom. All rectangles will be completely contained by the pixel grid. Note that the bottom-left corner of the grid is represented by the point (1,1).

Output

For each case output "Case x:" where x is the case number, on a single line by itself. Then output N lines of N characters each representing the final drawing of the paint program. If nothing was drawn at a particular pixel, a '.' should be outputted instead.

Sample Input

```
2
5
1
Filled 2 1 3 3 G
5
2
Empty 1 1 5 5 Y
Filled 1 2 2 1 B
```

Sample Output

```
Case 1:
.....
.....
.GGG.
.GGG.
.GGG.
Case 2:
YYYYY
Y...Y
Y...Y
BB..Y
YYYYY
```

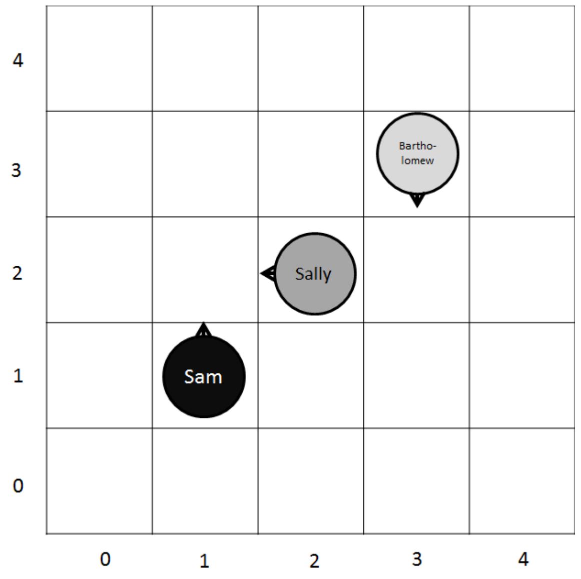
G. Back and Forth

The neighborhood kids have come up with another crazy game. Everybody runs back and forth within bounds, but they can only move in straight lines! The rules of the game are as follows:

- Before the game starts, someone is chosen to be "it."
- The game goes for a set amount of rounds, during which each player may take one step in their given direction.
- If more than two players land on a spot at one time, they all reverse their direction.
- If only two collide, then they will switch directions for the next step as follows. The one with the longer name starts moving in the direction that the other would. The one with the shorter name takes the reverse of the direction of the other player. (This only works because all of the kids in the neighborhood have different length names.)
- If a player's next step would put the player out of bounds, they reverse direction before taking their step.
- The winner is the person who ends up closest to the "it" player, with ties going to the person with the shorter name.

John thinks that this game is silly, since he could predict the outcome just by knowing the initial configuration. He wants you to write a program to do just that so he can show the other kids and convince them to play better games.

Below is the representation of the first test case. Note that (0,0) is the bottom-left corner.



Input

The first line of input is the number of test cases that follow. Each test case begins with a line containing integers M, N, and P: the field's width, its height (both measured in "steps"), and the number of players, respectively ($M, N, P < 10$). The next P lines contain space-separated values, starting with the name of one kid, followed by the x and y coordinates of their starting position on the field and the direction in which they start (N, S, E, or W). No two kids will start in the same spot. The first of the kids listed is "it" for the game. The last line of the test case (≤ 1000) is the number of rounds to play.

Output

For each case output "Case x:" where x is the case number, on a single line, followed by a space, and the name of the winner for that test case.

Sample Input

```
2
5 5 3
Sam 1 1 N
Sally 2 2 W
Bartholomew 3 3 S
20
5 5 4
Sam 0 0 N
Ed 4 0 W
Rebecca 0 4 E
Sally 4 4 S
3
```

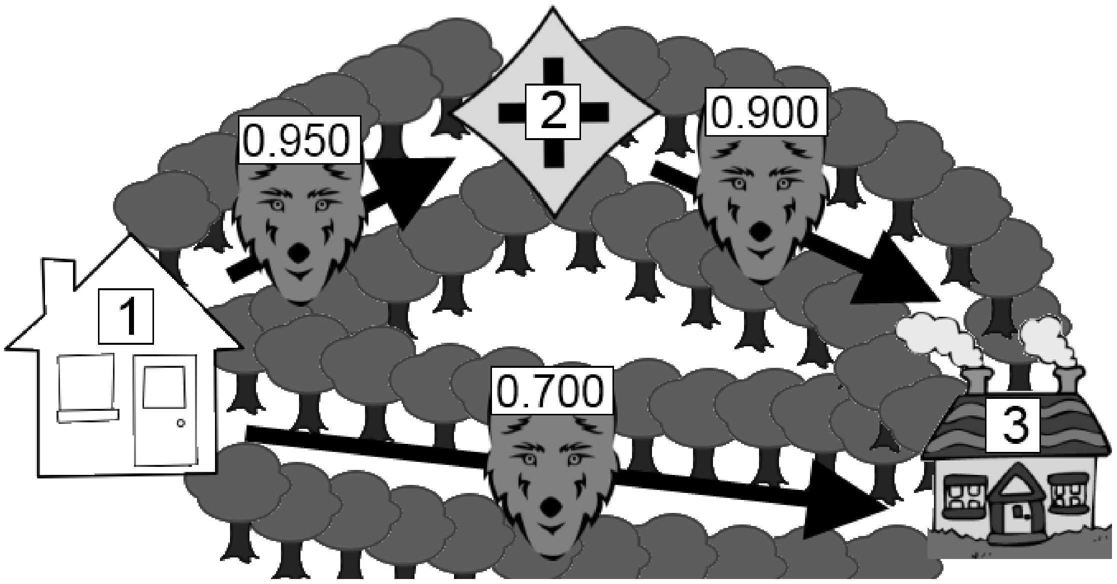
Sample Output

```
Case 1:  Sally
Case 2:  Ed
```

H. Hoo's Afraid of the Big Bad Wolf?

Little Red Riding Hood is walking to visit her Grandmother's house. Thankfully, Little Red Riding Hood is an avid reader of the Bid Bad Wolf's blog, which details the paths he and his friends are guarding. The Big Bad Wolf is no technological slouch, and knows the importance of keeping information private; thus his blog only states the likelihood that a path won't be guarded by a wolf. Should Little Red Riding Hood take a path that a wolf is guarding, she will be devoured, which is never a good thing. Paths through the forest are one-directional, and Little Red Riding Hood may not go backwards along a path. What route should Little Red Riding Hood take to maximize the chance of making it to Grandmother's?

Below is a diagram representing the first test case.



Input

The first line of input is the number of test cases that follow. Each test case starts with an integer N ($1 \leq N \leq 100$) on a line by itself representing the number of intersections. Then there will be a single line with two integers, X and Y ($1 \leq X, Y \leq N$), separated by a single space, indicating the numbers of the start (X) and end (Y) intersections. There will always be a path from the starting intersection to the ending intersection. Then the input will contain a single line with an integer M ($0 \leq M \leq 5000$), indicating the number of directed paths. M lines will follow, each containing three values separated by spaces: the start intersection A, the end intersection B, and the likelihood represented as a floating point number ($0.000 < P \leq 1.000$) that a path is safe–there is no wolf on that path. There can be multiple paths between the same two intersections.

Output

For each case output "Case x:" where x is the case number, on a single line, followed by the chance that Little Red Riding Hood makes it to Grandmother's house if she takes the safest path.

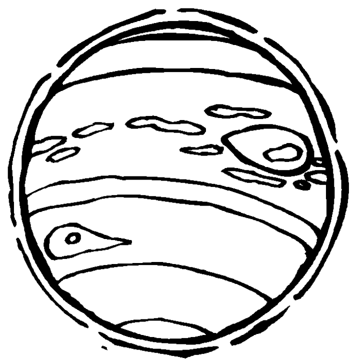
Sample Input

```
2
3
1 3
3
1 2 0.950
1 3 0.700
2 3 0.900
5
1 5
6
1 2 0.850
2 3 0.550
1 3 0.500
1 5 0.200
3 5 0.500
2 3 0.700
```

Sample Output

```
Case 1:  0.855
Case 2:  0.2975
```

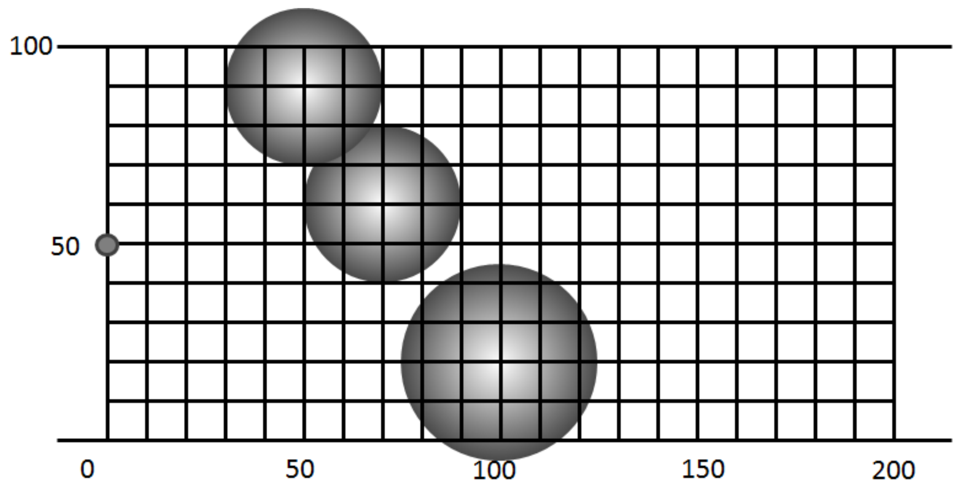
I. Canyon Crossing



Friendship One is a brand new rover commissioned by the Astronautical Center for Machinery to explore Triton, a moon of Neptune. It is being launched into space on March 18th, and the pressure is on to finish the software in time. Your boss has given you a very important task crucial to the completion of the mission.

In order to gather all of the necessary samples the rover must cross a canyon. Friendship One is equipped to travel on all sorts of rocky terrain, but this canyon is littered with circular craters. Your boss is concerned that if Friendship One takes a path that travels through one of these craters, Friendship One will fall over and the mission will end in failure. Some of these craters overlap with each other, which could create large impassable regions inside the canyon. It is up to you to program Friendship One to decide whether a canyon is passable or not. Are you up to the task?

The diagram below indicates the first test case in the sample data. Note that Friendship One is small enough compared to the size of the craters that it can be treated as a point; it has no area.



Input

The first line of input is the number of test cases that follow. Each test case starts with a line containing three integers, H, W, and N. H and W ($1 \le H, W \le 10000$) indicate the height and width of the canyon, respectively. N ($0 \le N \le 1000$) indicates the number of craters that follow. Each crater appears on a line by itself and contains three floating point values, X, Y, and R separated by spaces. X and Y ($0 < X - R; X + R < W$), ($0 \le Y \le H$) represent the center of the crater, while R is the radius of the crater.

Friendship One always starts at $X = 0$, with the destination being at $X = W$. Note that no point on the crater will overlap with $X = 0$ or $X = W$, so Friendship One can move vertically at these positions without difficulty.

Output

For each case output "Case x:" where x is the case number, on a single line, followed by the string "Clear To Go" if the canyon is passable and "Find Another Path" of the craters block the path.

Sample Input

```
2
100 200 3
50.000 90.000 20.000
70.000 60.000 20.000
100.000 20.000 25.000
10 20 2
3.000 2.000 7.000
4.000 5.000 6.500
```

Sample Output

```
Case 1:  Clear To Go
Case 2:  Find Another Path
```

J. Yahtzee

As a child, Colleen loved the game "Yahtzee". In Yahtzee, a player rolls 13 sets of five dice and assigns each set to various categories, which give points towards a cumulative total. This is a one-to-one matching, meaning that each set can only count for a single category and each category can only contribute points from a single set. This means that each category will be used exactly once, and each set of rolls will be used exactly once.

Now that Colleen has become an experienced programmer, she wants to determine the maximum possible score that she could achieve from a given set of dice rolls. Often times in Yahtzee, a player is allowed to select dice to re-roll; Colleen chose to ignore this rule to make things easier for herself.

The categories to assign dice roll sets in standard Yahtzee are as follows.

- **Ones**: Multiply the number of ones on the dice by 1. Example: (1 2 3 4 5 = 1 point)
- **Twos**: Multiply the number of twos on the dice by 2. Example: (3 1 1 5 6 = 0 points)
- **Threes**: Multiply the number of threes on the dice by 3. Example: (4 1 3 3 3 = 9 points)
- **Fours**: Multiply the number of fours on the dice by 4. Example: (4 1 1 6 5 = 4 points)
- **Fives**: Multiply the number of fives on the dice by 5. Example: (2 1 6 5 5 = 10 points)
- **Sixes**: Multiply the number of sixes on the dice by 6. Example: (6 2 1 6 3 = 12 points)

If the sum of the above six categories is at least 63, an additional 35-point bonus is added to the total.

- **Chance**: Sum up all of the dice. Example: (1 2 3 4 5 = 15 points)
- **Three of a Kind**: If there are at least three of a single dice roll, sum up all of the dice. Example: (1 1 1 5 6 = 14 points, 1 2 3 4 5 = 0 points)
- **Four of a Kind**: If there are at least four of a single dice roll, sum up all of the dice. Example: (1 1 1 1 6 = 10 points, 1 2 3 4 5 = 0 points)
- **Short Straight**: 25 points if at least four of the dice form a sequence. Example: (1 2 3 4 1 = 25 points, 1 2 3 1 2 = 0 points)
- **Long Straight**: 35 points if all five of the dice form a sequence. Example: (1 2 3 4 5 = 35 points, 1 2 3 4 1 = 0 points)
- **Full House**: 40 points if three of the dice are equal and the other two dice are also equal. Example: (5 2 5 5 2 = 40 points, 5 2 5 5 1 = 0 points)
- **Yahtzee**: 50 points if all of the dice are equal. Example: (1 1 1 1 1 = 50 points, 1 1 1 1 2 = 0 points)

Input

The first line of input is the number of test cases that follow. Each test case contains 13 lines of 5 integers each, where each line represents the five dice rolled in a single set. Each dice roll is between 1 and 6, indicating the value that appeared on that die.

Output

For each case output "Case x:" where x is the case number, on a single line, followed by a space, followed by the maximum Yahtzee score possible with these dice roll sets.

Sample Input

```
2
5 4 2 6 1
1 4 5 2 6
5 3 2 5 1
3 5 3 6 2
1 3 6 4 6
6 4 2 3 4
3 3 4 5 6
1 2 3 3 3
6 6 6 3 2
1 4 5 3 2
5 3 2 3 3
1 1 1 1 2
2 4 2 5 2
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

Sample Output

```
Case 1:  154
Case 2:  110
```