University of Texas at Austin - UIL Computer Science Contest

General Notes:

1. Do the problems in any order you like. The problems do not have to be completed in order from 1 to 12.

2. For problems that read from a data file: When the program is judged, the data file will be in the same directory as your program. Do not include any extraneous path information in your program, just the name of the data file.

3. Submit your source code, the .java file, for judging via the PC^2 software.

4. All problems are worth 60 points. Incorrect submissions receive a deduction of 5 points, but **ONLY** if the problem is ultimately judged correct. Problems may be reworked and resubmitted as many times as you like.

5. There is no extraneous input. All input is exactly as specified in the problem. Integer input will not have leading zeros unless specified by the problem.

6. Your program shall not print extraneous output. Follow the format exactly as given in the problem statement and as shown in example output.

7. The time limit for problems is 120 seconds. If the program has not terminated after 120 seconds on the judge's computer, the program will be judged as an incorrect submission.

Number	Problem Name
1	Base
2	Climb
3	Dell
4	Grade
5	Line
6	Match
7	Neighbor
8	Runs
9	Score
10	Square
11	Stones
12	Texas

1. Base Tetracontakaidi

Program Name: Base.java Input File: base.dat

Write a program to convert a base 42 number to a base 10 number.

The digits for the base 42 number are 0 through 9, a through z (with place values of 10 through 35), and the following digits:

base 40 digit	place value
!	36
?	37
@	38
#	39
$	40
%	41

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will be a single line with a number in base 42 using the format described above. There will be no extraneous spaces between numbers. There will be no leading zeros. There may be a single leading plus or negative sign.
- When converted from base 42 to base 10 the numbers will be in the range -2^{31} to $2^{31} - 1$.

Output

For each data set output the equivalent base 10 number. Do not include any leading zeros or leading plus signs.

Example Input File

```
9
1111
gggggg
!?
%%%@a
utcs!
0
10
-101
+a!
```

Example Output To Screen

```
75895
2142061168
1549
130691074
95521812
0
42
-1765
456
```

2. He Makes Them Suffer in the Mountains

Program Name: Climb.java Input File: climb.dat

Famous Austinite Lance Armstrong won the Tour De France seven times. One of the hardest parts of a bike race is when the road goes uphill. Uphill roads or "climbs" are given a rating in the Tour based on their length, steepness, point in the race where they occur, and the number of other climbs during that day's race, also known as a stage.

Write a program that given the length, steepness, whether the climb is near the end of a stage, and the total number of climbs in the stage determines the classification of the climb. The classifications are, from easiest to hardest, 4, 3, 2, 1 and HC or Hors Catégorie, which means beyond categorization: a fancy way of saying a climb of the hardest type.

A climb's difficulty score is obtained by determining the elevation gain in meters. This is simply the length of the climb in meters times the average steepness or gradient. So for example a climb of 1.5 kilometers with an average gradient of 3% has an elevation gain of 1.5 km * 1000 m / km * .03 = 45 meters.

The difficulty is modified in the following ways:

- if the climb occurs near the end of stage add 250 to the difficulty score.
- if there are a total or 4 or 5 climbs in a given stage add 50 to the difficulty score
- if there are 6 or more climbs in a given stage add 100 to the difficulty score.

The classifications based on difficulty score rounded to the nearest integer, DS, are:

- `DS <= 150 -> CAT 4`
- `150 < DS <= 320 -> CAT 3`
- `320 < DS <= 550 -> CAT 2`
- `550 < DS <= 1000 -> CAT 1`
- `DS > 1000 -> HC`

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will consist of 1 line of the form L G E T where L is a floating point number equal to the length of the climb in kilometers, G is the gradient of the climb expressed as a percentage with one decimal place, E is a string equal to YES if the climb is near the end of the stage or NO if it is not, and T is an integer equal to the total number of climbs in the stage including this one.

Output

For each data set print out a single line with the number of the data set, the difficulty score rounded to the nearest integer, and the category based on the difficulty score.

Example Input File

```
4
43.0 4.5 NO 3
11.7 8.5 NO 1
4.7 6.0 YES 6
1.3 6.4 NO 5
```

Example Output To Screen

```
1 1935 HC
2 995 CAT 1
3 632 CAT 1
4 133 CAT 4
```

3. Building Dell Hall

Program Name: Dell.java **Input File: NONE**

Construction is underway on Dell Hall, the new home to the computer science department. Dell Hall is named after Michael and Susan Dell whose contribution made the building possible. Dell Hall is the northern building of the Bill and Melinda Gates Computer Science Complex. The name for the south building of the complex is still available if you want to donate $20,000,000 to the UTCS department.

Write a program that shows the construction of Dell Hall over time.

Output

Output an image of Dell Hall as it grows over time. There are 6 stages of construction. After the first stage the first floor is complete, after the second stage the second floor is complete and so forth. Print out the stage number on a line by itself followed by the current version of Dell Hall.

The first, second, and sixth stages are shown below. Do print any trailing spaces!

Example Output To Screen

```
1
------------------
 |||.|..|.|..|.|||
 |||.|..|.|..|.|||
 |||.|..|.|..|.|||
2
------------------
 |||.|..|.|..|.|||
------------------
 |||.|..|.|..|.|||
 |||.|..|.|..|.|||
 |||.|..|.|..|.|||
// Replace this line with stages 3 through 5
6
)()()()(--))()()()(
------------------
 |||.|..|.|..|.|||
 |||.|..|.|..|.|||
------------------
 |||.|..|.|..|.|||
------------------
 |||.|..|.|..|.|||
------------------
 |||.|..|.|..|.|||
------------------
 |||.|..|.|..|.|||
------------------
 |||.|..|.|..|.|||
 |||.|..|.|..|.|||
 |||.|..|.|..|.|||
```

4. Basketball Performance Grading

Program Name: Grade.java Input File: grade.dat

Write a program to assist UT Women's Basketball coach Gail Goestenkors in grading her player's performance for a game.

Each player is assigned a score based on their statistics for a game. The statistics that are tracked are rebounds (REB +1), assists (AST +1), steals (STL +2), blocks (BLK +1), turn overs (TO -4), field goals (FG +2), 3 point field goals (3FG +3), fouls (PF -1) and free throws (FT +1).

Players are assigned a score based on their statistics for a game and the points for each event as shown above. Players are then graded based on a scheme and given one of the following grades: CAREER GAME, SPORTS CENTER, SOLID, MORE PRACTICE, BENCHED.

Write a program that reads in a player's statistics and assigns grades based on the following criteria:

```
CAREER GAME: score >= 60
SPORTS CENTER: 35 <= score < 60
SOLID: 20 <= score < 35
MORE PRACTICE: 5 <= score < 20
BENCHED: score < 5
```

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will start with a single number that is the player's number. Numbers may have leading zeros.The player's statistics for a game will follow separated by single spaces. The valid statistics are REB, AST, STL, BLK, TO, FG, 3FG, PF, and FT. It is possible for invalid symbols to appear. An invalid symbol is any symbol that does not match one of the valid symbols ignoring case. It is possible a player will have no statistics for a game.

Output

For each player print out her number, her score, and her grade. Ignore invalid symbols in the statistics unless there are 5 or more invalid symbols. If there are 5 or more invalid symbols print out ?? for the player's score and INVALID for her grade.

Example Input File

```
3
12 3FG 3FG FG fg G Fg 3fg STL BlK AST FG 3FG TO to 3FG 3FG 3fg
15
37 3fg 3fg 3fg 3 3 3 3 3 BLK BLK TO TO TO TO to
```

Example Output To Screen

```
12 25 SOLID
15 0 BENCHED
37 ?? INVALID
```

5. Line it Up

Program Name: Line.java Input File: line.dat

You have a collection of cards. Each card is split in half with one color on the left side and another color on the right side. It is possible for the color on the left and right side to be the same. You want to form a line with the cards laying them down from left to right. Determine the length of the longest line you can create from the cards with the requirement that the right hand color of a card must match the left hand color of the card next to it in line.

The left color of the first card does not have to match any other color and the right color of the last card does not have to match any other color.

Consider the following example. The cards are green-red, blue-green, red-blue, and red-yellow. The longest possible line with these cards is equal to 4.

red-blue, blue-green, green-red, red-yellow

In this case it is possible to form a line out of all the cards.

The cards may not be rotated. So a red-blue car may not be rotated to become a blue-red card.

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 be an integer M that indicates the number of cards in the data set. 1 <= M <= 30
- The next line will contain the cards in the data set. Each card will be of the form c1-c2 where c1 is the left hand color of the card and c2 is the right hand color of the card. There will be a space separating cards in a data set.
- Colors names will consist of 1 or more lower case letters.

Output

For each data set print out the length of the longest line that may be formed with the cards in the data set.

Example Input File (Although lines shown below are wrapped, the cards for a given data set will be on a single line.)

```
5
4
green-red blue-green red-blue red-yellow
1
red-blue
3
blue-green red-blue blue-green
10
red-yellow red-green blue-orange green-red blue-red red-blue green-
purple orange-green purple-green yellow-blue
8
red-yellow red-green green-yellow orange-orange blue-red
red-blue red-blue green-yellow
```

Example Output To Screen

```
4
1
2
10
4
```

6. Morning Person Or Not?

Program Name: Match.java Input File: match.dat

A big part of college is living in a dorm, apartment, or house with another person. Write a program to determine the compatibility score for potential roommates. Each roommate has filled out a survey with 10 questions. Each question has 5 possible answers, A through E, or a question may be skipped, indicated by an X.

The compatibility score for a question is the square of the difference bewteen answers with A assigned 1, B assigned 2, C assigned 3, D assigned 4, and E assigned 5. Thus if two people answered a question the same the compatibility score for that question is 0 (difference of 0, 0 * 0 = 0). If one answered A and the other answered B the compatibility score for that question is 1. (difference of 1, 1 * 1 = 1). If one answered A and the other answered E the compatibility score for that question is 16 (difference of 4, 4 * 4 = 16).

If one person skips a question assume their answer is whatever will yield the biggest difference. For example if one person answers D and the other skips the question assume the answer is A which gives the biggest difference and a compatibility score on that question of 9. If both people skip a question, again assume the answers will give the highest possible compatibility score of 16. (Assume one is A and the other is E.)

The overall compatibility score is the square root of the sums of the compatibility scores for each question rounded to the nearest integer.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will consist of 2 lines.
- Each line in a data set will contain 10 upper case characters, A - E or X, with no extra spaces. The characters are the answers to survey questions 1 through 10.

Output

For each data set print out the number of the data set followed by a space followed by the overall compatibility score rounded to the nearest integer.

Example Input File

```
4
AAAAAAAAAA
EEEEEEEEEE
BDBDBDBDBD
XXXXXXXXXX
ABCDEEDCBA
EDCBAABCDE
ABDEEBDBEA
CCECCEXXAA
```

Example Output To Screen

```
1 13
2 9
3 9
4 8
```

7. Howdy Neighbor

Program Name: Neighbor.java Input File: neighbor.dat

A rectangular 2D grid shows the style of houses at each location. A neighborhood is defined to be a contiguous set of houses of the same type that exceed some minimum number of houses. The program will use 4 different ways of defining neighbors:

1. 4 neighbors normal. Cells have at most 4 neighbors, the cells above, below, left, and right of a cell. Cells on the edge only have 3 neighbors. The 4 corner cells only have 2 neighbors.
2. 4 neighbors wrap. All cells have 4 neighbors. The edge and corner cells neighbors wrap around to the other edges.
3. 8 neighbors normal. Cells have at most 8 neighbors, the cells above, below, left, right, and the 4 diagonals of a cell. Cells on the edge only have 5 neighbors. The 4 corner cells only have 3 neighbors.
4. 8 neighbors wrap. All cells have 8 neighbors. The edge and corner cells neighbors wrap around to the other edges.

Consider this example of neighbors for the 8 neighbor wrap model with cells numbered for clarity.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Consider cell number 1 in the upper left corner. With wrapping its 8 neighboring cells are
16 13 14
4 2
8 5 6

Complete a program that prints the number of neighborhoods in a given data set given the minimum size of a neighborhood and the 4 ways of defining neighboring cells. A given house can belong to at most one neighborhood per data set.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- The first line in a data set will be a single integer M that indicates the minimum number of connected houses (size) required that need to be connected to form a neighborhood. M >= 1.
- The first line of a data set will be two integers R and C separated by a single space. R indicates the number of rows in the data set's matrix and C indicates the number of columns per row in the data set. R and C will both be >= 3.
- The next R lines will be the rows of the matrix in order. Each line will contain C upper case letters separated by single spaces. All letters will be between A and Z inclusive. Each letter represents the style of the house at that location in the matrix.

Output

For each data set print out a line with 4 integers separated by single spaces. The integers are the number of neighborhoods in the data set given the minimum required size based on the 4 rules for neighbors: 4 normal, 4 wrap, 8 normal, and 8 wrap.

Example Input File

```
3
3
4 3
A B C
D E F
G H I
J K L
3
5 7
A B A B A B A
A B A B A B A
A B C B A C A
A B C B A C A
A B A B A B A
5
3 7
A B B C B B A
B C C B C C B
A B B C B A A
```

Example Output To Screen

```
0 0 0 0
5 6 5 6
0 1 2 3
```

8. Moneyball

Program Name: Runs.java Input File: runs.dat

The book and movie *Moneyball* introduced Sabermetrics to a wider audience. Sabermetrics is the use of statistics to make predictions about the value and performance of baseball players and teams. One example of Sabermetrics is BaseRuns. BaseRuns attempts to predict the number of runs a team will score based on various team statistics.

The formula for BaseRuns (PR = predicted runs) is

$$PR = A * B / (B + C) + D$$

where

$$A = H + W + HBP - HR - .5 * IW$$
$$B = (1.4*TB - .6*H - 3*HR + .1*(W + HBP - IW) + .9*(SB - CS - GDP))*1.1$$
$$C = AB - H + CS + GDP$$
$$D = HR$$

H = number of hits
W = number of walks
HBP = number of times hit by pitch
HR = number of home runs
IW = number of intentional walks
TB = total number of bases
SB = number of stolen bases
CS = number of times caught stealing
GDP = number of times ground into a double play
AB = number of at bats

Write a program that calculates a team's BaseRuns and compares it to the actual number of runs a team scored. If a team score more runs than predicted they are considered lucky. If they score fewer runs than predicted, they are considered unlucky. If they score exactly the number of runs predicted they validate the model.

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 of integers separated by single spaces. The integers are
  H W HBP HR IW TB SB CS GDP AB AR
  Where all the terms are as explained above and AR represents the actual number of runs the team scored.

Output

For each data set print out the number of runs the BaseRuns formula predicts they will score rounded to the nearest integer followed by a space. Then based on the rounded predicted number of runs and the actual number of runs, print `LUCKY` if a team scored 1 or more runs than predicted, `UNLUCKY` if they scored 1 or fewer runs less than predicted, and `VALID` if they scored exactly the number of runs predicted.

Example Input File

```
4
1513 542 44 162 64 2351 57 39 169 5532 762
10 0 0 10 0 40 0 0 0 50 10
20 3 0 10 1 50 0 0 0 100 14
1439 569 30 67 66 1983 200 91 110 5455 685
```

Example Output To Screen

```
773 UNLUCKY
10 VALID
13 LUCKY
678 LUCKY
```

9. The Meta Question

Program Name: Score.java Input File: Score.dat

Write a program to determine a team's score for the programming portion of a UIL Computer Science Contest. There are 12 possible questions. A team earns 60 points if they get a question correct. If a team misses a question, they lose 5 points for each miss if they eventually get the question correct. If a team does not solve a question, they do not lose any points for incorrect attempts. A team may get negative points if they miss a question 13 or more times and eventually (stubbornly perhaps?) solve the question. If a team submits a question after they have already solved it, the submission is ignored.

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 followed by a blank line. Even the last data set will be followed by a blank line.
- The line will contain X problem submissions.
- Each submission will consist of 3 elements, T Q R, where T is a string corresponding to the team that submitted the solution, Q is an integer 1 through 12 inclusive indicating the question number, and R the result, either C for correct or I for incorrect. There will be no spaces in the team names.

Output

For each data set output the number of the data set on a line by itself followed by the results for all teams that made at least 1 submission for the data set. List each team on a line by itself with the team's name followed by a space, and followed by the team's score. List the teams in descending order based on score. In the case of ties, list the teams in ascending ASCII code order, also known as ASCIIbetical order.

Example Input File (Lines are wrapped on this page but will be a single line followed by a blank line in the input file.)

```
2
t1 1 C t5 1 C t1 2 I t2 12 C t2 10 I t2 10 I t2 10 I t2 10 I t2 10 I t2 10 I
t2 10 I t2 10 I t2 10 I t2 10 I t2 10 I t2 10 I t2 10 I t2 10 I t2 10 I t2 10
I t2 10 C t1 1 I t2 10 I t5 10 I t5 10 I t5 10 C

west 1 C west 5 C west 12 C east 2 C east 3 C east 10 C north 3 I north 5 I
north 12 I south 5 I north 12 I south 5 C south 5 C south 10 I
```

Example Output To Screen

```
1
t5 110
t1 60
t2 40
2
east 180
west 180
south 55
north 0
```

10. Square Free Integers

Program Name: Square.java **Input File: square.dat**

Write a program that determines if various integers are square free integers or not.

A square free integer is one that is divisible by no perfect square other than 1. A perfect square is an integer that is the product of some integer and itself. Thus 10 is not a perfect square, but 16 is. $16 = 4^2$.

10 is a square free integer because it is divisible by 1, 2, 5, and 10. None of those integers, other than 1 is a perfect square. 18 is divisible by 1, 2, 3, 6, 9, and 18. 9 is a perfect square ($9 = 3^2$) thus 18 is not a square free integer.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will consist of exactly 1 line with a single integer, x. $1 < x <= 2^{31} - 1$

Output

For each data set print out `FREE` if x is a square free integer or `NOT` if x is not a square free integer followed by a single space and the largest perfect square x is divisible by other than 1.

Example Input File

```
6
10
18
2
2500
731324180
100
```

Example Output To Screen

```
FREE
NOT 9
FREE
NOT 2500
NOT 146264836
NOT 100
```

11. Romancing the Stones

Program Name: Stones.java **Input File: stones.dat**

A family has inherited a set of valuable necklaces made of up of various jewels. Each jewel in the necklace has a certain value. The family wishes to split the necklace up into segments so that each segment is same total value. Write a program to determine if it is possible to cut a given necklace into the desired number of segments so that the value of the segments, based on the value of the jewels in a segment, are all equal. The number of cuts must equal the target number of segments.

For example, consider the following necklace. The values of the jewels are shown as numbers. The '-' and '|' characters are links in the necklace. There is a single link between the last jewel and the first jewel. The links have 0 value.

```
--------------------
|-1-1-3-2-2-2-4-2-1-|
```

Suppose the family wishes to split this necklace up into 3 segments. We must cut the necklace up into 3 segments. Recall the value of the resulting segments must all be equal.

One solution for the necklace shown is to split into these segments:

```
-------------------------------
|-1-1-3 CUT 2-2-2 CUT 4-2 CUT 1-|
```

By making the 3 cuts shown we create a segment with 2, 2, 2 worth 6, a segment with 4, 2 worth 6 and a segment with 1, 1, 1, 3 worth 6. It is possible to split the necklace shown into 3 segments with equal value. Cuts must be made between jewels. Jewels cannot be cut into smaller pieces.

Input

- The first line will contain a single integer N that indicates the number of data sets.
- Each data set will consist of 2 lines.
- The first line of each data set will be two integers separated by a single space, `s j`. The first integer, `s`, indicates the number of segments the necklace for the data set is to be cut into. `s > 0`. The second integer, `j`, indicates the number of jewels in the necklace for this data set. `j > 0`.
- The next line will contain `j` integers separated by spaces representing the value of the jewels. Each integer will be greater than or equal to 0. (Recall there is a link between the last jewel and the first jewel on a line.)

Output

For each data set print out the number of the data set followed by a space and then `CUT` if it is possible to cut the necklace into the desired number of segments so that all segments are of equal value. Print out `WHOLE` if it is not possible to cut the necklace into the desired number of segments so that all segments are of equal value.

Example Input File

```
8
3 9
1 1 3 2 2 2 4 2 1
3 8
2 2 1 3 3 4 1 2
2 10
2 1 3 1 4 1 2 2 1 3
3 13
2 2 1 5 10 2 5 3 2 1 1 1 1
3 11
5 3 1 6 9 7 10 5 3 5 3
4 9
1 1 3 2 2 2 4 2 1
5 11
1 1 1 2 1 1 1 2 1 3 1
3 12
4 1 5 1 3 2 1 4 3 1 1 2
```

Example Output To Screen

```
1 CUT
2 WHOLE
3 CUT
4 CUT
5 WHOLE
6 WHOLE
7 CUT
8 WHOLE
```

12. Fight Song

Program Name: Texas.java Input File: none

Written by Colonel Walter S. Hunnicutt in collaboration with James E. Kin, *Texas Fight*, is the official fight song of the University of Texas at Austin. The lyrics to the first verse of the song are:

Texas Fight, Texas Fight,
And it's goodbye to A&M.
Texas Fight, Texas Fight,
And we'll put over one more win.
Texas Fight, Texas Fight,
For it's Texas that we love best.
Hail, Hail, The gang's all here,
And it's good-bye to all the rest!

Write a program that prints out the first verse of the song, building it up one line at a time. Before each build print out the number of the build and the number of 't's and 'T's that appear in that build.

Example Output To Screen

```
1 4
Texas Fight, Texas Fight,
2 6
Texas Fight, Texas Fight,
And it's goodbye to A&M.
3 10
Texas Fight, Texas Fight,
And it's goodbye to A&M.
Texas Fight, Texas Fight,
```

**** BUILDS 4 through 7 not shown. Your output must show them correctly.**

```
8 25
Texas Fight, Texas Fight,
And it's goodbye to A&M.
Texas Fight, Texas Fight,
And we'll put over one more win.
Texas Fight, Texas Fight,
For it's Texas that we love best.
Hail, Hail, The gang's all here,
And it's good-bye to all the rest!
```