Question A: The Name Game

The classic song "The Name Game" shows how you can take any name and make a cute-sounding song about it:

For example, Jack:

> Jack, Jack, bo-back
> Banana-fana fo-fack
> Fee-fi-mo-mack
> Jack!

This process can be used for any name, the general rule for a name X is:

> (*X*), (*X*), bo-b(*X*−1)
> Banana-fana fo-f(*X*−1)
> Fee-fi-mo-m(*X*−1)
> *X*!

(X-1) here means the name X without the first letter. (The actual song removes the entire first consonant sound, but our rule will be to only remove the first letter)

The only exception comes when the name itself begins with a B, F, or M. In that case, the letter is not repeated in the line that adds that letter. So, for Billy:

> Billy, Billy, bo-illy
> Banana-fana fo-filly
> Fee-fi-mo-milly
> Billy!

Input:
The input for this problem will contain a number, *n*, and then *n* names, one on each line. Each name will be started with an upper-case letter, and will be a single string of letters, with no whitespace.

Output:
For each name print the entire name game for that name. Separate data sets with a blank line, but do not put an extra blank line at the end of the output.

Sample Input:
2
Jack
Billy

Sample Output:
Jack, Jack, bo-back
Banana-fana fo-fack
Fee-fi-mo-mack
Jack!

Billy, Billy, bo-illy
Banana-fana fo-filly
Fee-fi-mo-milly
Billy!

Problem B: High Jump

The High Jump is a popular Olympic event. A bar is placed at some initial height, and contestants jump over it. Each contestant has three chances to clear the bar at the given height. If they can't clear it in any of their three chances, they are eliminated. After all contestants finish at a certain height, the bar is raised, and contestants try to jump over the higher bar. Misses from previous heights don't carry over, so all contestants will be able to take three attempts at all heights.

Contestants are also allowed to skip an entire height, and thus will not have an entry at that height. Contestants that do this are still alive in the competition, but only receive credit for jumps that are successfully made.

When all (or all except for one) of the contestants are eliminated, the contestant who cleared the highest distance successfully is the winner. In the case of a tie, the contestant with the least misses at the highest distance is the winner. If there is still a tie, the contestant with the least total misses overall wins. If there is still a tie, then the result is a tie. (In the Olympics, this is resolved by having a "jump-off" between the two competitors).

Your task is to process the High Jump scores for an event and determine the winner (or winners).

Input:
The first input will be a number, n, $n \geq 1$, denoting the number of events to score for. Each of the n events will consist of a number, m, containing the number of entrants ($1 \leq m \leq 10$). Then there will be a series of heights in the following form:
Height
k
Contestant number$_1$ Attempts
Contestant number$_2$ Attempts
…
Contestant number$_k$ Attempts

"*Height*" is a decimal value with exactly one digit before the decimal point and two digits after the decimal point. (for example, 1.97), which corresponds to the height of the bar. *k* is the number of contestants who attempted the jump at this height. Then there will be *k* lines, each denoting a contestant by number (contestant numbers start at 1) and their attempts. The attempts will consist of three characters, each character will be either 'Y', signifying they made the jump on that attempt, 'N', signifying they missed the jump on that attempt. Attempts after a success are designated by 'S', signifying they skipped that attempt (since they made it successfully).

The heights will be in increasing order, but there is no special ordering for the contestants within a height. The height entries will conclude with a height of 0.00. There will be at most 10 distinct heights.

Output:
For each input case, output one line stating:
Event *i*: The winner is contestant *n*, with a height of *h.*

where *i* is the event number (starting at 1) n is the contestant number of the winner, and h is their last successful jump.

If there is a tie and no way to determine the winner, output the line:
There is a tie between the following contestants: n_1 n_2…n_n
(There is a single space after the colon)

Where the various n_i are the contestant numbers of the tied contestants, listed in increasing numerical order and separated by spaces.

Sample Input:
3
3
1.00
3
1 N Y S
2 S S S
3 N N N
1.10
2
1 Y S S
2 N N N
0.00

2
1.00
2
1 N N Y
2 Y S S
0.00

2
1.00
2
1 N N Y
2 N N Y
0.00

Sample Output:
Event 1: The winner is contestant 1, with a height of 1.10
Event 2: The winner is contestant 2, with a height of 1.00
Event 3: There is a tie between the following contestants: 1 2

Problem C: Pollution Simulation

In modeling oil spills and other polluted environments, scientists often want to know how the flow of the pollutant spreads through the environment and around obstacles. In this problem, you are to simulate the flow of a pollutant through a sample environment.

Input:
The first input value is a number, n, corresponding to a number of input instances. For each input instance, there will be two parameters, r and c, specifying the rows and columns of the environment to test (both r and c will be between 1 and 100, inclusive). Then there will be an rxc grid of characters, where each cell in the grid is either:

- a - which signifies an empty space
- a * which signifies a barrier that the pollution cannot get through
- a P which signifies a pollutant.

Output:
For each input instance, output a grid showing the time at which each cell in the grid becomes polluted. If a cell never gets polluted, leave it in its initial state. Some assumptions to make:

- All P spots start polluted at time 0
- Each time step, all empty squares rectilinearly adjacent (i.e. not diagonally) to a polluted square become polluted themselves.
- Each cell is either polluted or not. You can't be "doubly polluted"

Your output grid should show the timestep (starting at 0) when the cell becomes polluted. If a cell never becomes polluted, leave it in its initial state.

Sample Input:

```
2
5 5
*****
*---*
*-P-*
*---*
*****
10 10
**********
*----P---*
*--------*
*------***
****-*****
*--------*
----------
----------
------P---
----------
```

Sample Output:

```
*****
*212*
*101*
*212*
*****

**********
*43210123*
*54321234*
*654323***
****4*****
*87654345*
8765432345
7654321234
6543210123
7654321234
```

Problem D: Koosbanian Calculations

Aliens on the planet Koosbain have 6 fingers on their left hand, and 7 fingers on their right. As a result, their number system is base 13, instead of the base 10 system that we use. This almost led to a galactic war when the Koosbanian envoy said that we should exchange 12 gifts. We showed up with twelve, and they showed up with fifteen, leading to some unhappy shortchanged Koosbanians.

To avoid another interplanetary incident, the government has asked you to create a conversion program that takes Koosbanian mathematical expressions and convert them to the normal base-10 expressions used by most humans.

The Koosbanian number system has 13 "digits": 0-9, and the letters A B and C, representing 10, 11, and 12.

Koosbanian mathematical expressions use the same order of operations that we use, and the same associativity, but have no parentheses.

Input:
The first input is a number, n, denoting the number of expressions to follow. Each expression will be on one line, and will consist of a Koosbanian mathematical expression, using just Koosbanian numbers, and the operators +, -, *, and / (integer division, which rounds down). All numbers and symbols will be separated by spaces, and the end of the line will be marked with a # symbol. There will be no spaces within a number.

All numbers in the expression are non-negative, but a negative number may result from the calculation. There will be no division by 0.

Output:
The base 10 number equivalent to the result of the Koosbanian expression. Each answer should go on its own line, with no separating blank lines.

Sample Input
```
2
10 + 2 #
3 + 4 * 1A - B #
```

Sample Output
```
15
84
```

Question E: Speedy Delivery

Irene Matilda Speedy runs a delivery company for packages in downtown Columbus, Ohio. Irene's company owns two delivery trucks. Each day, there is a sequence of deliveries that must be made, and Irene needs to decide how to schedule the trips of each of her two trucks. Your job is to determine the best allocation of trips for the trucks, to minimize the total distance both trucks travel.

Some details to consider:

- Delivery locations will be denoted as points in the plane (corresponding to the number of blocks in the North/South and East/West direction to travel.) The corporate headquarters will be located at (0,0). Since this is city driving, all distances will be Manhattan Distance (the distance between (x_1, y_1) and (x_2 , y_2) is $| (x_1-x_2) + (y_1-y_2)|$. (In other words, the trucks can only drive directly North/South or East/West between blocks).

- Each customer knows about their place in the list of deliveries, and is very touchy about being served in order. Thus, if a truck makes more than one delivery, each delivery that the truck makes must be in order of the deliveries received. (There can be gaps in the sequence, but no re-orderings). In other words, if two deliveries, d_i and d_j are made by the same truck, and d_i comes before d_j in the list of deliveries, then d_i must be delivered before d_j.

- Each truck can start the day fully loaded, so there is no need to return back to the headquarters to restock.

Input:
The first input will be a number *n*, followed by *n* problem instances. Each problem instance will begin with a number *k*, followed by *k* locations for deliveries to be shipped to ($1 \leq k \leq 50$). Each location will be on its own line, and will consist of two integers- the x and y coordinates of the delivery destination.

Output:
For each input instance, output the line:

Day N: K total blocks traveled.

N is the current problem instance (starting at 1), and K is the minimum number of blocks traveled by both trucks combined to make all deliveries. Do not factor in the time it takes to make it back to (0,0) from the last delivery at the end of the day.

Sample Input:

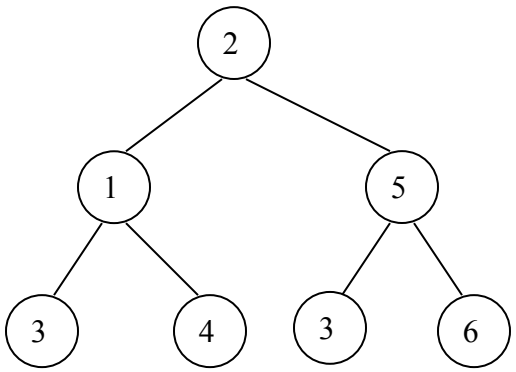
```
2
5
1 0
0 1
2 0
0 2
2 2
3
0 5
0 2
2 0
```

Sample Output:

```
Day 1: 6 blocks traveled.
Day 2: 10 blocks traveled.
```

Problem F: Tree Routing

Suppose we have a binary tree that has duplicate elements (this is not necessarily a binary search tree). For example:



There are two copies of the value "3" in this tree. We can define a "route" between the two threes:
Starting with the leftmost one, move Up, Up, Right, Left

Given a complete, full binary tree (i.e. a tree in which every non-leaf node has two children, and all leaves are on the same level of the tree), and a value that exists exactly twice, describe the route between both instances of that value.

Input:
The first input will be a number, *n*, consisting of the number of problem instances. Each of the *n* instances will begin with a number *k*, denoting the number of nodes in the tree ($k < 128$). The next line will contain *k* integers, denoting the values in the tree. The values will be listed breadth first, so the first value will be the root, then the left child of the root, then the right child, then the level below from left to right, and so on. The final line of the instance is the number to route between. This number will appear in the tree exactly twice.

Output:
For each instance, output the line:
Tree *N*: Path: <*path*>

Where *N* is the current problem instance (starting at 1), and <*path*> is the path to take to get from the leftmost instance of the requested number to the rightmost. The path should be a sequence of one of the letters U,R,L (meaning to go Up, Right, or Left), and should have no spaces between them.

Sample Input:

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

3
1 2 2
2
```

Sample Output:

```
Tree 1: Path: UURL
Tree 2: Path: UR
```