19th Annual Denison Spring Programming Contest
23 February 2008

Rules:

1. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
2. All programs will be re-compiled prior to testing with the judges' data.
3. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed.
4. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
5. All communication with the judges will be handled by the PC^2 environment.
6. The allowed programming languages are C, C++ and Java.
7. Judges' decisions are to be considered final.
8. There are **six** questions to be completed in **four hours**.
9. **Important:** No extraneous whitespace should appear in your output. Specifically, there should be no blank lines, unless specifically called for. A line should *never* end with whitespace. Do not use tabs in your output, only spaces. Data fields in an output line should be separated by a single whitespace, unless specified otherwise. Any deviations to these guidelines will result in a "Format Error" from the judges.

Problem A: Teetering Towers

Towers of varying heights are arranged in a long row. You may dynamite one so that it falls either right or left. Like dominoes, if the falling tower hits another one, it will cause it to fall over, too. (But if the top of a tower just touches the bottom of another tower, that tower stays standing.) You have just enough dynamite to cause one tower to fall either right or left. You wish to knock over as many towers as possible.

Input

There will be multiple problem sets. Each set will start with a line containing a single positive integer n indicating the number of towers. The next line will contain n pairs of integers. Each pair, x h, will indicate the position x of a tower of height h. ($x \geq 0$, $h > 0$ and all numbers will be no larger than 100.) You must find the tower to initially knock over (and in what direction) so as to knock over as many towers as possible. Input will terminate with a 0 on a line.

Output

For each input set, output a line of the form:

`Case` k `can have` m `towers fall.`

where k indicates the number of the input set (starting at 1) and m is the maximum number of towers that can be knocked over.

Sample Input

```
6
0 2 3 2 4 2 5 1 7 4 8 1
3
1 3 8 2 6 1
0
```

Sample Output

```
Case 1 can have 4 towers fall.
Case 2 can have 1 towers fall.
```

Problem B: Relay Team

You are the coach of a youth swimming club. One of the big events is the 100 yard medley relay which has four 25 yard legs, each with a different stroke — backstroke, butterfly, breaststroke, and freestyle (in that order) — each leg by a different swimmer. In order to encourage learning all the strokes, you allow no one to swim the medley relay who cannot swim all four strokes under a minute each. League championships are coming up and you're putting together your best team. It's not just a matter of picking the swimmer with the best time in each stroke because your strongest swimmers typically are strong in more than one stroke. You decide to write a program to pick the best relay team.

Soon you realize that there may be more than one arrangement of swimmers that get you the best total time. Now you've numbered each swimmer 1, 2, 3, etc. in the order they appear in the file. Each line in the file will contain one swimmer's times in the backstroke, butterfly, breaststroke, and freestyle (in that order), in seconds. To break ties between teams with the best total time, you'll take the team with the lowest numbered backstroke swimmer . If still a tie, then lowest numbered butterflyer. If still a tie, then lowest numbered breaststroker. Finally, if still a tie, then lowest numbered freestyler.

For example, consider the following listing of times:

```
35 28 30 30
49 25 37 38
39 28 30 30
37 29 33 35
32 30 35 40
```

The best team would be swimmers 5, 2, 1, 3 swimming the backstroke, butterfly, breaststroke, and freestyle, respectively, for a total time of 1:57.

Input

Input for each test case will start with n ($4 \leq n < 100$) indicating the number of swimmers' data to follow. The data for each swimmer follow, one swimmer per line, as integers k b t f, giving the time (in seconds) for their backstroke, butterfly, breaststroke, and freestyle, respectively. All times will be less than 60. The first swimmer is swimmer number 1, followed by swimmer number 2, etc. $n = 0$ indicates end of input.

Output

For each test case output one line of the form:

`Team` k`'s best relay team is swimmers` a b c d `for a total time of` m:ss.

where k is the test case (starting at 1), a b c d are the swimmers' numbers swimming the backstroke, butterfly, breaststroke, and freestyle, respectively, and m:ss is the total time in minutes and seconds of this team.

(next page)

Sample Input

```
5
35 28 30 30
49 25 37 38
39 28 30 30
37 29 33 35
32 30 35 40
6
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 29
29 30 30 30
0
```

Sample Output

```
Team 1's best relay team is swimmers 5 2 1 3 for a total time of 1:57.
Team 2's best relay team is swimmers 6 1 2 5 for a total time of 1:58.
```

Problem C: Mirrored Palindromes

As you know, a palindrome is a string of characters that is the same both forwards and backwards, like the word `MADAM`. There is another form of palindrome call a *mirrored palindrome*. Some characters are mirrors of themselves (such as `A` or `M`) and some characters are mirror images of other characters (such as `E` and `3` — you sometimes have to stretch your imagination a bit). An example of a mirrored palindrome is `AEIUI3A`. `MADAM` is not a mirrored palindrome. We'll restrict our characters to `A-Z` and `1-9`. (Note zero is not included, being too close to `O`.)

Here's a list of all pairs of mirrored characters: `E`↔`3`, `J`↔`L`, `S`↔`2`, `Z`↔`5`.

The 'self-mirrored' characters are: `A`, `H`, `I`, `M`, `O`, `T`, `U`, `V`, `W`, `X`, `Y`, `1`, `8`.

The characters without mirrors are: `B`, `C`, `D`, `F`, `G`, `K`, `N`, `P`, `Q`, `R`, `4`, `6`, `7`, `9`.

Input

Each input set will consist of a string of length no longer than 50 characters from the characters `A-Z`, `1-9`. The last line, which should not be processed, will be the word `END`.

Output

For each input string, output a line that is either `NO` or `YES` according to if the string is a mirrored palindrome.

Sample Input

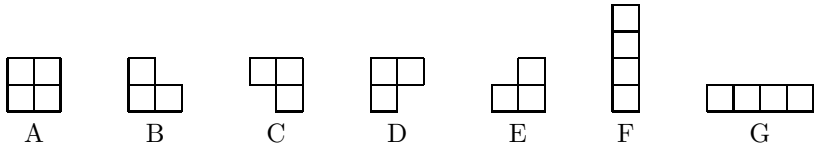
```
TOTEM
MADAM
AEIUI3A
MOTOM
END
```

Sample Output

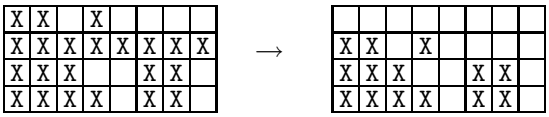
```
NO
NO
YES
YES
```

Problem D: Tetris

At least once in your life, you've probably played Tetris. Here, we'll be interested in a simplified version where the shapes that are dropped into the box can not be re-oriented. There will be 7 different shapes, labeled A through G as indicated below.



Our game will be played on a 'board' that is eight squares wide. A shape is dropped in a certain position and falls until it 'bumps up against' a previous piece, or hits the bottom. You'll be given a series of drops and asked to show the resulting board. Recall that in Tetris, if an entire row of squares is filled, the row 'disappears' and the rows above this eliminated row all drop down one row. For example, the diagram below indicates this process, where an `X` indicates a filled square.



Note that when a row disappears the *rows* above drop, not the individual squares. Thus there may be 'floating' squares, as we see in the example.

Input

There will be multiple test cases. Each test case will begin with an integer n ($n < 100$) on a line indicating the number of pieces to drop onto the board, ($n = 0$ indicates end of input.) The next line(s) will consist of n pairs, s c, separated by a single space, indicating that shape s will drop with its left-most square in column c. The columns are numbered $0 \ldots 7$, left-to-right, and $s \in \{A, B, C, D, E, F, G\}$, as given above. No shape will be illegally dropped 'outside' the board. (For example, `G 5` would be an illegal drop.) Furthermore, assume that at no time during the game will the pieces stack up higher than 10 rows.

Output

For each input set you will output a line indicating how many rows have squares filled and then print out the board. The format of the first line should match the example in the sample output. There should be no empty lines between output of test cases. The board should be printed as rows of `X` (indicating a filled square) and `.` (indicating an empty square). The top non-empty row is printed first. The first sample test case shows the board given above.

(next page)

Sample Input

```
6
B 2 A 0 A 0 E 2 A 5 G 4
2
G 4 G 0
8
F 0 F 1 F 2 F 3 F 4 F 5 F 6 F 7
3
D 0 B 1 D 2
0
```

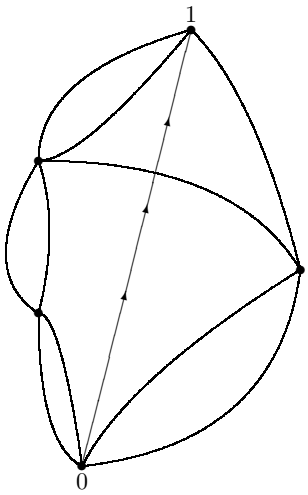
Sample Output

```
Case 1 results in 3 rows.
XX.X....
XXX..XX.
XXXX.XX.
Case 2 results in 0 rows.
Case 3 results in 0 rows.
Case 4 results in 5 rows.
..XX....
.XX.....
.XX.....
XX......
X.......
```

Problem E: Ski Slope

Bill and Peg own a little mom and pop ski slope. Their little mountain has only one lift but a few slopes. The visiting skier has limited options, but Bill and Peg want to advertise the number of different runs down the mountain that are possible — even though many runs overlap a bit.

Below we've shown Bill and Peg's slope. The base of the lift is labeled 0 and the top 1. The other curves are the various paths down the mountain. Curves between dots are called *legs* and a *run* is a path from the top to the bottom (which consists of one or more legs). The dots indicate *junctions* where the various legs of the runs meet and you then have a choice of the next leg you might take, thus giving you different runs. In the example below, there are 14 different runs from the top to the base, depending on which combination of legs you take.



Bear in mind that gravity being what it is, you can only ski in one direction (downhill, of course) between adjacent junctions. In this problem, you'll be given the layout of a ski slope serviced by a single lift and asked to figure out how many runs there are

Input

Input for each test case starts with a line of the form n l indicating there are n junctions (in addition to the base and top) and l legs. ($l = 0$ indicates end of input and $l < 30$.) These junctions will be numbered $2, 3, \ldots, n+1$ (if $n > 0$) with 0 indicating the base and 1 indicating the top. On the following line there follows l pairs, a b, indicating a leg for a down to b. All junctions will be connected to at least one other junction. The data will be consistent (no loops up the mountain) and every leg will be on at least one run.

Output

Each test case should output a line of the form

`Slope` m `has` r `runs.`

where you determine the value for r and m is the number of the test case (starting at 1.)

(next page)

Sample Input

```
3 10
1 2 1 2 1 3 2 3 2 4 2 4 4 0 4 0 3 0 3 0
3 10
1 2 1 2 1 2 2 4 4 0 2 3 4 0 2 3 3 0 3 0
0 0
```

Sample Output

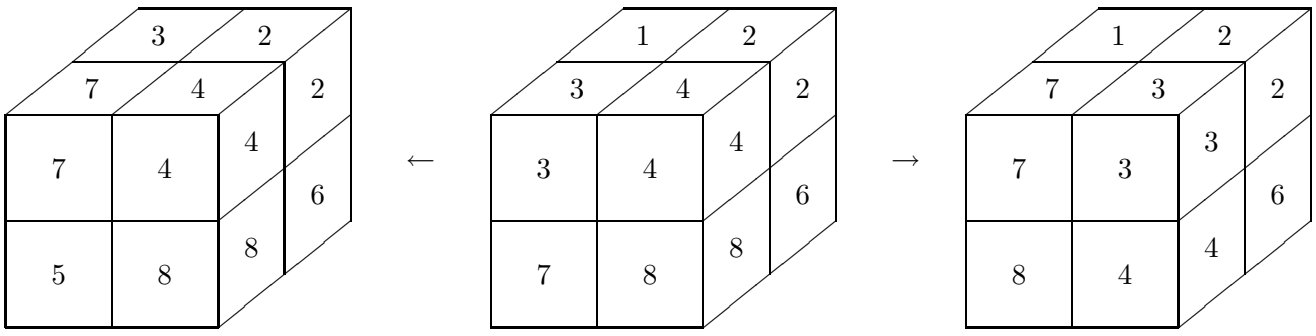
```
Slope 1 has 14 runs.
Slope 2 has 18 runs.
```

Problem F: Rubic's Cube — alpha version

While everyone knows Rubic's Cube, few people know that it went through a number of refinements until it reached it's final form. We'll be looking at the first, or alpha, version. In the alpha version, the cube consisted of only 8 blocks arranged in a $2 \times 2 \times 2$ array. For reference purposes, we'll number the blocks as in the middle picture below and call this the *standard position*. Beyond fewer blocks in the cube, each block was a solid color, either red, green, blue, white, yellow or orange. (We'll denote the colors by their single lower-case first letter.) So, the individual faces of a given block were not colored separately — that refinement came later.

As in the final version, all planes of blocks could be rotated in either direction. We'll denote these planes as top, bottom, right, left, front and back. (Denoted `T, B, R, L, F` and `K`.) each plane can be rotated either clockwise (`c`) or counter-clockwise (`t`) relative to looking at the face in question. For example a move that moves the left face counter-clockwise (`L t`) would result in the standard positioned cube to be oriented as in the left cube below. And the move `F c` would result in the standard positioned cube being as in the right cube below.

As you've probably guessed by now, you'll be given the initial coloring of a cube and a series of moves and asked to produce the end colorings.



move `L t` standard position move `F c`

Input

Each test case will start with a line containing the the integer n (≤ 20) indicating the number of moves followed by a string of length eight indicating the colors of the blocks, in order of their initial position. (A value of $n = 0$ indicates end of input, in which case there will be no colors following on the line.) The following line(s) will contain n pairs, x y, indicating that face x will be moved in direction y.

Output

Each test case should produce a single output line of the form:

`Cube` m `has coloring` $c_1c_2c_3c_4c_5c_6c_7c_8$`.`

where c_i is the color of the block that sits in standard position i after all n of the moves have been made and m is the test case number.

(next page)

Sample Input

```
5 rwbgoyrg
F c B c R t R t K c
3 rrrwyyyb
L c T c K t
0
```

Sample Output

```
Cube 1 has coloring gbrgryow.
Cube 2 has coloring yrwryyrb.
```