

Seventh Annual Ohio Wesleyan University Programming Contest

March 23, 2013

Rules:

1. There are six questions to be completed in four hours.
2. 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. **Do not** include any prompts, other debugging information, or any other output except for exactly what is specified by the problem.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed.
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All output cases will end in a newline character. So after your program finishes output, the cursor should be on the first column of the line immediately below your last case's output. Note that some problems will require an additional blank line at the end of your output
8. All communication with the judges will be handled in the PC2 environment.
9. Judges' decisions are final. No cheating will be tolerated.

Problem A: The Automated Programming Contest Submission System

If you've participated in programming contests before, you may have been tempted to write a program that generates all possible responses, in some order, until you get the question right. The problem, of course, is that will probably take a lot of incorrect submissions. In this question, we'll look at just how many there can be.

Suppose you know that a question asks you to write a program whose output is a string that has n distinct letters in it, rearranged in some order. Suppose further that someone (on some other team, hopefully) has designed a way to automatically submit all possible rearrangements of those letters, in alphabetical order, until they get the problem correct. Your task is to determine how many total submissions the team will be making.

Input:

There will be multiple input cases. Each input case will begin with an integer n ($n \leq 11$), indicating the number of letters in the string. A value of $n=0$ will signify end of input. The next line will contain n distinct capital letters, separated by spaces. The next line will contain one permutation of the n letters, which is the correct answer to the problem.

Output:

For each input case i , starting with 1, output the line:

Case i : There will be a total of k submission(s).

.. where k is the position of the requested string in an alphabetical listing of all strings made by rearranging the given letters.

Sample Input:

```
3
A B C
BCA

5
A B D E L
BLADE

11
A B C D E F G H Z K Y
ZYKHGFEDCBA

0
```

Sample Output:

Case 1: There will be a total of 4 submission(s).

Case 2: There will be a total of 43 submission(s).

Case 3: There will be a total of 39916800 submission(s).

Problem B: MMO Crafting

Many multiplayer online games have you accumulate resources, and then provide a crafting system to combine these resources into other items. These items may then be crafted into still better items, and so on. There is usually an online marketplace to sell both components and items, and it is often the case that the price of a crafted item exceeds the price of the lesser pieces that the item is made of. This can be seen as the buyers in the marketplace paying a premium to not have to do the crafting themselves. As a result, a player who wants to make money can often do so by finding opportunities to profit from these market inequities.

Let's start by defining some terms:

- Each player has access to a collection of items. This is the universe of possible things a player can have.
- Some items can be crafted by combining other items as components according to some recipe. For our purposes, each item has at most one recipe.
- Some items can't be crafted, only found, though they may be used as components in recipes for other items. We'll call these items base components.

Your task is to write a program that figures out the amount of money that can be made from each item in a set of items, given marketplace costs and crafting recipes.

Input:

There will be several sets of input, each corresponding to a different universe of items. Each set will begin with an integer n ($n \leq 20$). A value of $n=0$ corresponds to end of input.

The next line will contain n strings, separated by whitespace. These strings are the names of the items to consider.

There will then be n lines of the form:

k c_1 c_2 ... c_k *price*

Each line will hold the information about how to make the items in the list of items (So the first line will be for the first alphabetical item, and so on)

k is the number of components needed to create this item ($k \leq 10$). A value of $k=0$ is allowed, and means that this item cannot be crafted (it is a base component). The line continues with the components c_i that can be combined to make this item. Components will be listed by name, in alphabetical order, and may have duplicates (meaning multiple copies of a component are required). The line ends with an integer price that the item will sell for in the marketplace. You should factor in the cost of buying all other crafting materials from the marketplace. (So, use the marketplace price for any other items- including extra copies of the item you have. Yes, we can have a discussion about "opportunity cost" instead, but maybe we'll do that in some future contest. Use the marketplace cost.)

There will be no “circular crafting” recipes (Where item A can be made into B, then C, then back to A).

Output:

For each input set i , starting with 1, output the line:

Set i

Then n lines of the form:

Name has a profit of X

...where “*Name*” is the name of each item (listed in alphabetical order), and X is the maximum profit you can get if you have one of that item on hand- either selling the item either directly, or by crafting it into something else, and selling that.

Output a blank line after each set (including the last)

Sample Input:

```
5
Branch GiantWeb Silk Spiderweb StickyBranch
0 1
3 Spiderweb Spiderweb Spiderweb 50
0 5
2 Silk Silk 12
2 Spiderweb Branch 10
4
FineCrystal Handle Lightsaber RoughCrystal
2 RoughCrystal RoughCrystal 10
0 3
2 FineCrystal Handle 50
0 1
0
```

Sample Output:

```
Set 1:
Branch has a profit of 1
GiantWeb has a profit of 50
Silk has a profit of 21
Spiderweb has a profit of 26
StickyBranch has a profit of 10

Set 2:
FineCrystal has a profit of 47
Handle has a profit of 40
Lightsaber has a profit of 50
RoughCrystal has a profit of 46
```

Problem C: Katie Casey and the Variable Number of Strikes

Katie Casey is the girl who asks you to “Take Me Out to the Ballgame”, in the famous song. The problem is that when singing the song, she gets a bit confused about how many strikes there are before you’re out at the old ball game.

Input:

There will be multiple input instances. Each input instance will be an integer n , $0 \leq n \leq 10$. A value of $n < 2$ will denote end of input.

Output:

For each input instance, output the following:

```
Let me root, root, root for the home team
If they don't win it's a shame
For it's <list of strikes> strikes you're out
At the old ball game!
```

Where *<list of strikes>* is a comma-separated list of the words counting from “one” to n .

There is a space after each comma, and a newline after each line above.

Output a blank line after each test case, including the last one.

Sample Input:

```
3
7
0
```

Sample Output:

```
Let me root, root, root for the home team
If they don't win it's a shame
For it's one, two, three strikes you're out
At the old ball game!
```

```
Let me root, root, root for the home team
If they don't win it's a shame
For it's one, two, three, four, five, six, seven strikes you're out
At the old ball game!
```

D: Three-Dimensional Connect-N

In a standard Connect Four game, players alternate placing discs in one of several slots arranged in a row. The discs fall down the slot until they hit another disc (or the bottom of the slot). The arrangement of discs form a grid, and one player wins if they can create a line of 4 pieces of their color either horizontally, vertically, or diagonally (pretty sneaky, sis!)

In three-dimensional connect four, the two players alternate placing torus-shaped discs onto a 4×4 two-dimensional grid of pegs. The discs slide down the pegs until they rest on another disc (or the bottom of the peg). There are more ways to win now:

- Horizontal, vertical, or diagonal along the “top plane” of the board” (what you’d see looking down vertically on the board)
- 4 in a row on one peg.

(I haven’t actually found a coherent set of rules for this game, so other ways may exist, but we’ll go with these).

What’s interesting about this game is that as the game progresses, it’s remarkably easy to accidentally make a move that wins the game for your opponent. So, on your turn, it’s useful to first see if your opponent left you a win, and secondly see which pegs are “safe”. A peg is “safe” if it is legal to place a move on the peg without giving your opponent a game-winning move on their very next turn.

Your task is to figure out these decisions for a generalized game played on an $N \times N \times N$ grid. In an $N \times N$ grid, a player wins by getting N in a row horizontally, vertically, or diagonally on the top level, or by forming a peg with N pieces of their color on it. Each peg can only hold N pieces before becoming full.

Input:

There will be several input instances. Each input instance will begin with an integer n , between 0 and 10, representing the dimensions of the grid.. A value of $n=0$ will denote end of input.

Otherwise, there will be n^2 lines, one for each peg. Each line will begin with an integer k , between 0 and n , which will be the number of pieces on that peg. Then will be k integers (either 1 or 2), showing the stack of pieces from bottom to top. A 1 means player 1’s piece, a 2 means player 2’s piece. Stacks will be given from left to right, top to bottom.

After the n lines, will be one more integer, either a 1 or a 2, saying whose turn is next.

Output:

For each input instance i , starting with 1, output either:

Board i : Winning move at peg p

..where p is the smallest numbered peg that will cause a win for the person moving

Or

Board i : Safe moves at: $p_1 p_2 \dots p_j$

..where the p_i values are peg numbers in sequential order of the safe moves , separated by commas and spaces. There is no space after the last peg.

Or

Board i : No safe moves found!

..if no save moves exist.

Position numbers start at 1, and read left to right in rows. So the position numbers of the board where $N=4$ would be:

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

Sample Input:

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

(more)

4
1 1
1 2
1 1
1 1
1 1
1 2
1 1
1 1
1 2
1 1
1 2
1 1
1 2
1 1
1 1
2 1 2
1

2
2 1 2
0
0
0
1

0

Sample Output:

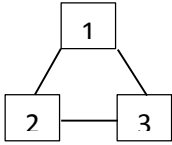
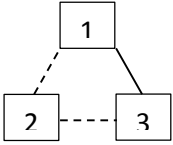
Board 1: Safe moves at: 1 2 3 4 5 7 8 9

Board 2: Winning move at peg 2

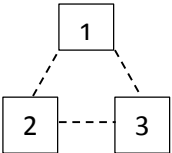
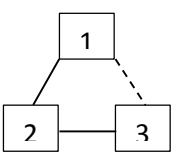
Board 3: No safe moves found!

Problem E: Stable relationship triangles.

Suppose we define relationships between two people as either “friends” or “enemies”. Then we can look at groups of three people, which will fall into one of four patterns. If a solid line means “friend” and a dashed line means “enemy”, we can say that two of the patterns are “stable”:

	
The “mutual admiration society”	The “enemy of my enemy is my friend”

And two of the patterns are “unstable” in that they tend to quickly devolve into one of the two “stable” patterns.

	
The “mutual antagonism society” (Soon two enemies will find common cause against the third person)	The “two of my friends hate each other!” (Either 2 will persuade 1 and 3 to be friends, or 1 or 3 will poison 2 against the other one)

Your task is to design a program that takes a population of several people, with their mutual relationships defined, and determining how many unstable relationships there are both now, and after one change.

Input:

There will be several input instances. Each input instance will begin with a non-negative integer $n \leq 20$, representing the number of people. A value of $n=0$ signifies end of input.

The input instance will continue with n lines. Each line will begin with an integer m ($0 \leq m < n$), and then m integers $f_1..f_m$. These values contain the friends of each person. The first line will be the friends of person 1, the second line will be the friends of person 2, and so on. Person numbers will range from 1 to n . Any relationship between two people will be symmetric (so if A is marked as a friend of B, then B will be marked as a friend of A). If a friendship is not defined between two people, they are assumed to be enemies.

Each input set will always have at least one unstable triangle.

Output:

For each input instance i , output the lines:

Set i :

There are currently j unstable triangles.

Making # A and # B friends/enemies results in k unstable triangles.

Where:

i is the current input instance

j is the number of unstable triangles in the input list

A and B are the numbers of people whose relationship can be flipped to create the minimum number of unstable triangles, out of all possible flips. You should report whether they become friends or enemies.

If there are multiple ways to find the minimum number of triangles, break ties by choosing the lowest numbered A . Break further ties by choosing the lowest numbered B .

k is the new number of unstable triangles after your flip.

Output a blank line after each set, including the last one.

Sample input:

4

2 2 3

3 1 3 4

3 1 2 4

2 2 3

5

4 2 3 4 5

2 1 4

2 1 5

3 1 2 5

3 1 3 4

0

Sample Output:

Set 1:

There are currently 2 unstable triangles.

Making #1 and #4 friends results in 0 unstable triangles.

Set 2:

There are currently 5 unstable triangles.

Making #1 and #2 enemies results in 4 unstable triangles.

Problem F: OCD Easter Bunny

The Date: Easter morning, 2013

The Time: 5:27 AM

The Place: The local rectilinear grid-based park

The Problem: The Easter Bunny has been to town, dropping eggs beneath bushes willy-nilly. Some bushes have a bunch of eggs underneath them, some bushes have none. The children are coming soon to look for eggs, and you don't want them looking under the wrong bushes, do you?

Your Mission (if you choose to accept it): Create a catalog of all bushes and egg counts

Input:

There will be several input instances. Each input instance will begin with two integers, r and c indicating the number of rows and columns in the park. Both r and c will be ≤ 20 . A value of $r=c=0$ will signify end of input.

Otherwise, there will be r lines of c characters, separated by spaces, representing the spaces in the park. Each space will either be a '-' for an empty space, or a 'B' for a bush. Two B squares are considered to be part of the same bush if they are adjacent horizontally or vertically.

Input will continue with a positive integer e , denoting the number of eggs. Then there will be e lines of two integers r_i and c_i , denoting the row and column of each egg. The upper left corner of the park is row 0, column 0. Each egg will be in a space marked 'B' in the grid.

Output:

For each input instance i , output a table with the following lines:

Park i :

J_1 bush(es) have K_1 egg(s) underneath them.

J_2 bush(es) have K_2 egg(s) underneath them.

...

Where the various J and K values are the number of bushes and the number of eggs, respectively. You should sort your output by K (so increasing number of eggs), and you should omit numbers of eggs that have 0 bushes.

Output a blank line after each park's information, including the last one.

Sample Input:

```
5 5
B B - - B
- B - - B
B - - - -
- - B B B
- - - - B
4
0 0
0 1
3 2
0 4

7 8
B B B B B B B -
B - - - - B -
B - B B B - B -
B - B - B - B -
B - B B B - B -
B - - - - B -
B B B B B B B -

5
0 0
0 1
2 2
4 4
6 6

0 0
```

Sample Output:

```
Park 1:
1 bush(es) have 0 egg(s) underneath them.
2 bush(es) have 1 egg(s) underneath them.
1 bush(es) have 2 egg(s) underneath them.

Park 2:
1 bush(es) have 2 egg(s) underneath them.
1 bush(es) have 3 egg(s) underneath them.
```