UCF Local Contest — September 5, 2015

Medal Ranking

filename: `medal`
(*Difficulty Level:* `Easy`)

When different countries compete against each other (e.g., in the Olympics), they receive gold/silver/bronze medals. The countries can then be ranked in one of two ways: by "count" which is based on the total number of medals (regardless of the medal colors) or by "color" which is based on the number of gold medals (and silver medals if tied in gold medals, and bronze medals if tied in gold and silver).

The Problem:

Given the gold/silver/bronze medal counts for USA and Russia, you are to determine if USA wins in these two ranking methods.

The Input:

The first input line contains a positive integer, *n*, indicating the number of data sets to check. The sets are on the following *n* input lines, one set per line. Each set consists of 6 integers (each integer between 0 and 500 inclusive); the first three integers represent (respectively) the gold, silver, and bronze medal counts for USA; the last three integers provide this info for Russia (in same order).

The Output:

Print each input set. Then, on the next output line, print one of four messages (`count, color, both, none`), indicating how USA can win. USA will win by `count` if its total medal count is higher than the total for Russia. USA will win by `color` if it has more gold medals than Russia (if tied in gold, then USA must have more silver; if tied in gold and silver, then USA must have more bronze). Leave a blank line after the output for each test case.

Sample Input:

```
5
10 5 15 10 1 0
10 5 15 10 6 10
12 5 10 5 20 30
10 0 15 10 5 30
10 5 15 10 5 15
```

Sample Output:

```
10 5 15 10 1 0
both

10 5 15 10 6 10
count

12 5 10 5 20 30
color

10 0 15 10 5 30
none

10 5 15 10 5 15
none
```

UCF Local Contest — September 5, 2015

Brownies vs. Candies vs. Cookies

filename: `brownie`
(*Difficulty Level:* `Easy`)

Everyone is welcome to the UCF Programming Team practices, and many students take advantage of this opportunity. The main benefit is that these students improve their problem solving and programming skills. Another benefit is that the students enjoy the refreshments Dr. Orooji brings every week! Dr. O usually brings candies but sometimes he brings cookies or brownies. Brownies are very popular and don't usually last long, so Dr. O has to come up with some clever trick to make the brownies last longer (so that the students stay for the entire practice!). Well, the easiest solution is to cut the brownies in half; that will double the number of brownies.

The Problem:

Given the original number of brownies and the students wanting brownies, you are to keep track of the brownie count as Dr. O cuts them in half.

The Input:

The first input line contains a positive integer, *n*, indicating the number of programming team practices. This is followed by the data for these practices. The first input line for each practice contains two integers (separated by a space): the number of students (between 1 and 30 inclusive) in the practice and the number of brownies (between 60 and 600 inclusive) Dr. O has brought that day. The next input line for the practice contains a positive integer, *m*, indicating how many groups of students approach the refreshment table to take brownies. This is followed by the number of students in each group, one number per line. Assume that the input values are valid, e.g., the number of students in a group will be at least 1 and it will not be greater than the number of students in the practice.

If a group of students is approaching the refreshment table and Dr. O notices that the number of remaining brownies is less than or equal to the number of students in the group, Dr. O cuts the brownies in half to be sure they won't be all gone after each student in the group grabs one brownie. Note that, if needed, Dr. O will cut the brownies more than once (as many times as needed). For example, if there are 3 brownies left and 24 students are approaching the table, Dr. O has to cut the brownies four times ($3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48$) to be sure the brownies won't be all gone after each student in the group grabs one.

The Output:

At the beginning of each practice, output "`Practice #`*p*`:` *s* *b*" where *p* is the practice number (starting with 1), *s* is the number of students in this practice, and *b* is the number of brownies. Then, on each of the following output lines, print the number of students in a group approaching the refreshment table and the number of brownies left after each of these students

has grabbed one brownie (note that cutting in halves may occur before grabbing). Leave a blank line after the output for each practice.

Sample Input:

```
2
20 60
8
15
10
20
18
9
12
2
10
15 100
4
1
2
3
5
```

Sample Output:

```
Practice #1: 20 60
15 45
10 35
20 15
18 12
9 3
12 12
2 10
10 10

Practice #2: 15 100
1 99
2 97
3 94
5 89
```

UCF Local Contest — September 5, 2015

Lemonade Stand

filename: `lemonade`
(*Difficulty Level:* `Medium`)

You are running a lemonade stand and have the good fortune of knowing exactly how many cups of lemonade customers are going to want to buy on each day that you run the lemonade stand. You hate to turn any customer away, so you would like to make sure that you always have enough lemons and sugar to make the appropriate number of cups of lemonade on each day. Unfortunately, the cost of lemons and sugar change daily, so you have to choose on which days you buy each, and how much of each to buy. You can buy individual lemons and five pound bags of sugar. (Note that there are 16 ounces in one pound.) On the days you choose to buy ingredients, you buy them in the morning, before any sales are made. (You're an early riser, so you can always get to the store and back before any customers would come.) Note that you can buy as little or as much as you wish on any day to minimize your overall cost, i.e., you have enough startup money (capital) to buy as much as you wish on any day.

The Problem:

Given that you always want to have enough lemons and sugar to serve each customer, determine the minimum cost of buying those lemons and sugar.

The Input:

The first input line will have a single integer, n $(1 \leq n \leq 100)$, the number of cases to process. The first line of each test case will have three space-separated positive integers: d $(1 \leq d \leq 1000)$, the number of days you'll run the lemonade stand, x $(1 \leq x \leq 10)$, the number of lemons needed to make a single cup of lemonade, and s $(1 \leq s \leq 10)$, the number of ounces of sugar needed to make a single cup of lemonade. The following d lines will contain data for days 1 through d, respectively. Each of these lines will have three integers separated by spaces: c $(1 \leq c \leq 1000)$, the number of cups sold for that day, p_l $(1 \leq p_l \leq 50)$, the price of a single lemon in cents for that day, and p_s $(1 \leq p_s \leq 500)$, the price of a five pound bag of sugar in cents for that day. Note that the extra sugar and lemon from each day carry over to the next day.

The Output:

For each test case, print the minimum cost of supplies (in cents) necessary to make sure that no customer who wants a cup of lemonade gets turned away.

Sample Input:

```
2
3 3 2
200 10 399
300 8 499
400 12 499
2 5 10
9 10 199
8 20 99
```

Sample Output:

```
31977
1347
```

UCF Local Contest — September 5, 2015

Balanced Strings

filename: `balance`
(*Difficulty Level:* `Medium`)

Being an individual obsessed with balancing and trying to find strings that meet that criteria, you have begun a new endeavor to study the curious structure of what we will call balanced strings.

Balanced strings are strings that maintain an equal ratio of consonants to vowels in all of their substrings. What? You don't know the difference between a consonant and a vowel? Vowels are the characters 'a', 'e', 'i', 'o', 'u' and sometimes 'y'. Actually, you don't like the character 'y' because this makes the problem much harder. What was the difficulty level of this problem again? Oh yeah Medium! We can't have a problem about consonants and vowels that makes 'y' sometimes a vowel! That would make the problem a Hard and too many hard problems is a very bad thing. Being obsessed with balance, you have decided that 'y' will be included with the vowels. That way, there are 6 vowels and 20 consonants. A nice balanced even number of both! Oh! I almost forgot! A consonant is any letter that is not a vowel. Also to simplify things (this is a medium problem after all!), we will consider strings composed of lowercase letters only.

Now you are ready to understand balanced strings! A balanced string is a string that has an equal number of consonants and vowels in all of its substrings. Well… not all of its substrings. Just the substrings of even length! For example, the string "orooji" has the following set of even-length substrings: {"or", "ro", "oo", "oj", "ji", "oroo", "rooj", "ooji", "orooji"}. In that set the following substrings are unbalanced: {"oo", "oroo", "ooji", "orooji"}. That is, the substrings do not contain an equal number of vowels and consonants. So, the string "orooji" is not balanced. But a string like "arup" is balanced. The requisite even-length substrings are: {"ar", "ru", "up", "arup"} and all these substrings are balanced (have the same number of vowels and consonants), thus the string "arup" is balanced. Note that a balanced string may contain an odd number of characters, e.g., the string "ali" is balanced since all its even-length substrings ({"al", "li"}) are balanced.

Now you want to take words and erase some of the letters and replace them with new letters to form balanced strings. Since this is a medium problem, we've taken the liberty of erasing some of the letters for you and replaced them with '?' characters. See! The problem is already halfway solved!

The Problem:

Given a string of lowercase letters and '?' characters, count the number of ways to replace all the '?' with lowercase letters such that the string results in a balanced string. Two ways are considered different if there exists some i such that the i^{th} character in one string differs from the i^{th} character of the other string. Note that all the '?' do not have to be replaced by the same letter.

The Input:

The first input line contains a positive integer, *n*, indicating the number of strings to balance. The strings are on the following *n* input lines, one string per line. Each string contains only lowercase letters and/or '?' characters. Assume each input string has at least one character and at most 100 characters.

The Output:

For each string, output "`String #d: v`" where *v* is the number of ways to replace the questions marks with lower case letters. It is guaranteed that *v* will fit in a signed 64-bit integer for the strings we provide. Leave a blank line after the output for each string.

Sample Input:

```
7
orooji
al?
a?i
g?ha
a?u?
??????????????
arup
```

Sample Output:

```
String #1: 0

String #2: 6

String #3: 20

String #4: 6

String #5: 400

String #6: 11179524096000000000

String #7: 1
```

Note: The first and last strings in Sample Input do not have any '?'. The first input is not balanced so there is no way of replacing all '?' to make it balanced, thus the output is zero. However, the last input is balanced so there is one way of replacing all '?' to make it balanced, thus the output is 1.

UCF Local Contest — September 5, 2015

Longest Path

filename: `longpath`
(*Difficulty Level:* `Hard`)

You are visiting a lovely garden, with several points of interest. Every pair of points of interest have a single, one way path (edge) connecting them. (The paths are one way so that guests don't get upset having to deal with other guests walking in the opposite direction.) Luckily, you get to decide which point of interest to begin from. Naturally, to get your money's worth, you'd like to visit as many of the points of interest as possible without revisiting any point of interest, since that would be a waste of time. Determine a walking path that allows you to visit the maximal number of points of interest without repeating any of them.

The Problem:

Given a directed graph containing exactly one edge (path) between every pair of vertices (points of interest), determine a walking path of maximal length that doesn't revisit any vertex.

The Input:

The first input line contains a positive integer, n $(n \leq 100)$, indicating the number of gardens to check. The input for each garden follows. Each set starts with a line with a single integer, v $(1 \leq v \leq 500)$, the number of points of interest for that case. The following v lines will contain v space separated integers each. The j^{th} $(1 \leq j \leq v)$ of these integers on the i^{th} $(1 \leq i \leq v)$ line is 1 if there is a path from point i to point j, or 0 if there is a directed path from point j to point i instead. The i^{th} integer on the i^{th} line will always be 0. It is guaranteed that the input information is consistent - if the j^{th} integer on the i^{th} line is 1 then the i^{th} integer on the j^{th} line is 0, and vice versa.

The Output:

For each garden, on a line by itself, output a space separated list of any maximal length sequence of unique points of interest to visit.

Sample Input:

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

Sample Output:

```
1 2 4 3
1 2 3
```