ACM International Collegiate Contest
2014 East Central Regional PRACTICE Contest
Grand Valley State University
University of Cincinnati
University of Windsor
Youngstown State University
November 7, 2014

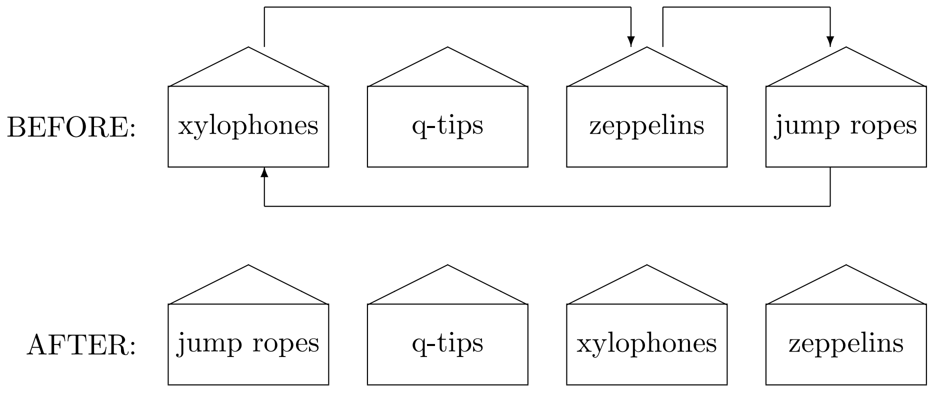
Sponsored by IBM

Rules:

1. There are **three** problems to be completed in **90 minutes**.
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. Additional input and output specifications can be found in the General Information Sheet.
3. When displaying results, follow the format in the Sample Output for each problem. Unless otherwise stated, all whitespace in the Sample Output consists of exactly one blank character.
4. The allowed programming languages are C, C++ and Java.
5. All programs will be re-compiled prior to testing with the judges' data.
6. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
7. The input to all problems will consist of multiple test cases.
8. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
9. All communication with the judges will be handled by the PC^2 environment.
10. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: The Cost of Moving

You've been put in charge of reorganizing the inventory at Amalgamated, Inc. AI has factories at various sites around the country, each site manufacturing different sets of products. Currently, each site has all of their products stored in a row of warehouses, with each warehouse storing one type of product. Your idea is to move these products around so that the warehouses store the items in alphabetical order. For example, one site which manufactures xylophones, q-tips, zeppelins and jump ropes stores them in four warehouses as shown at the top of the following figure:



The arrows show how far each product would have to be moved in order for the warehouse to store them in alphabetical order, which is shown at the bottom of the figure.

Management likes your idea but is concerned about the cost of moving all of the products. The farther a product has to be moved, the more it costs, so before committing to any re-ordering they would like to know the total length that all products have to be moved at any given site. In the site above, xylophones would have to be moved a total length of 2 (measured in warehouses), zeppelins would have to move 1 and jump ropes would have to move 3, for a total cost of 6. What you need is a program that can determine this movement cost automatically.

Input Time Limit: 3 secs, No. of Test Cases: 37, Input File Size 52.0K

There will be multiple test cases. Each test case will start with a line containing a positive integer `n` indicating the number of products at the site. Following that will be one or more lines containing the `n` names of the products in their current order. Each name will be a single string, and a single space will separate each consecutive pair of names on any line. The maximum value of `n` is 1000, and no two product names will be the same. A single zero will terminate input.

Output

For each test case, output the site number followed by the total length of movements needed to reorganize the products. Follow the format shown in the Sample Output.

Sample Input

```
4
xylophones q-tips zeppelins jumpropes
12
partridges turtledoves frenchhens callingbirds goldenrings
geese swans milkers dancers leapers pipers drummers
0
```

Sample Output

```
Site 1: 6
Site 2: 48
```

Problem B: A Jaw-dropping Problem to Vex the Quizzically Freakish

You all know what a pangram is, so I don't have to tell you that it's a phrase or sentence that uses every letter of the alphabet at least once. You also know what the most famous pangram is, so I won't waste your time reminding you that it's:

The quick brown fox jumps over a lazy dog

BUT, do you know what a double pangram is? Yes, yes, you're right, it's a phrase or sentence that uses every letter at least twice. And a triple pangram is one that uses every letter at least three times. And a ... well, we could go on, but the practice contest is only one and a half hours so we'll stop here.

Input Time Limit: 3 secs, No. of Test Cases: 113, Input File Size 13.0K

There will be multiple test cases. The input file starts with an integer `n` indicating the number of cases. Each test case will be a single line containing upper and lower case letters, as well as other non-alphabetic characters such as digits, punctuation and spaces.

Output

For each test case, output the case number followed by one of the following phrases:

```
Not a pangram
Pangram!
Double pangram!!
Triple pangram!!!
```

Select the phrase that best describes the test case. For example, even though a triple pangram is also a double (and single) pangram, the phrase `Triple pangram!!!` best describes it. There will be no test cases which use every letter of the alphabet more than three times.

Sample Input

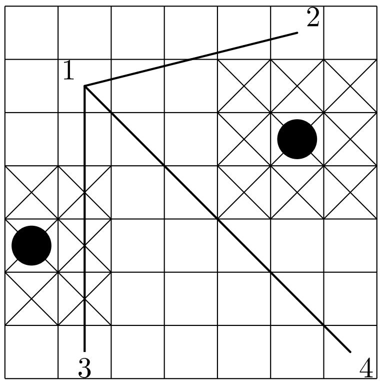
```
3
The quick brown fox jumps over a lazy dog.
The quick brown fox jumps over a laconic dog.
abcdefghijklmNOPQRSTUVWXYZ-zyxwvutsrqpon   2013/2014      MLKJIHGFEDCBA
```

Sample Output

```
Case 1: Pangram!
Case 2: Not a pangram
Case 3: Double pangram!!
```

Problem C: This Too Shall Pass

Tessa is a high school senior who plays on the soccer team. Her coaches are trying to put an emphasis on passing, and they want to help players learn how to recognize when a teammate is open for a pass. They've decided to model the field as a square grid like the one shown below. Each player (for both the offense and the defense) takes up one grid square - the offensive players are indicated with numbers and the defensive players with black circles. Each defensive player also has the ability to move to any neighboring square in order to intercept a pass; the squares that each defender can guard are shown with an X in them. Assume player 1 has the ball. Another player is considered open if the line drawn from the center of that person's square to the center of player 1's square does not touch any of the squares that can be reached by a defender. If the line touches a defenders area even at a single point, then the pass could be intercepted. Offensive players never block a pass to other offensive players.



In the above example, Player 1 can pass the ball to Player 2, but not to Players 3 and 4.

Having set up this great model, the coaches suddenly realize that they don't have any ability to write code to determine who is open and who is isn't. One of the players has given them your name as a computer whiz, so it's time to get your game face on, put it all on the line, give 110% and code one for the Gipper.

Input Time Limit: 3 secs, No. of Test Cases: 36, Input File Size 63.5K

The input file will consist of multiple test cases. Each case starts with four positive integers `r c o d` indicating the number of rows (`r`) and columns (`c`) in the grid and the number of offensive and defensive players (`o` and `d`, respectively). Both `r` and `c` will be ≤ 50. Following this will be `o` lines containing two integers giving the row and column location of an offensive player (row and column numbering start at 0). Following this will be `d` analogous lines for the defenders. The offensive players are numbered $1, 2, 3, \ldots$ in the order that they appear in the input, and offensive player 1 is the one with the ball. No two players will ever be in the same grid square. A line with four zeros will terminate input.

Output

For each test case, output the case number followed by a list of all the players that Player 1 can pass the ball to. Output the numbers in increasing order, separated by a single space. Label each test case as shown below.

Sample Input

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

Sample Output

```
Case 1: 2
Case 2:
Case 3: 2 3 4
```