LADUE INVITATIONAL COMPUTER SCIENCE TOURNAMENT

Problem 1: Network Vulnerability

Description:

You are in charge of maintaining a network for your company to maximize connectivity between computers while avoiding redundancy and vulnerability. While you have successfully connected all computers in the network, there are a few choke points (referred to as “cut nodes”) that make the system fragile. These cut nodes are nodes that, once removed, prevent one part of the network from communicating with another. Your job is to compute the minimum number of additional connections required to eliminate all cut nodes.

|  |  |
| --- | --- |
| Image result for cut node graph theory | C:\Users\Patrick\AppData\Local\Microsoft\Windows\INetCacheContent.Word\separated.png |
| c and e are both cut nodes | Removing either creates two separate networks |

Input: network.in

The first line will contain an integer N (1<N<100), the number of datasets. Each dataset will begin with a number E (1<E<1000), the number of edges in the graph. All graphs will be connected. Each edge is represented by two space separated strings, both representing a node. The end of the input is signified by the end of the file.

Output: network.out

For each dataset, print a single integer, followed by a newline, representing the minimum number of additional edges required to make the network invulnerable (i.e., make it so that there are no longer any “cut nodes” within the graph). For example, in the graph above, the answer would be 1, because an edge from b to f would make the graph invulnerable.

Samples:

|  |  |
| --- | --- |
| Input | Output |
| 3  4  1 2  3 1  1 4  3 4  5  1 2  2 6  6 4  4 1  4 5  5  1 2  1 3  3 2  3 4  1 4 | 1  1  0 |
| 1  19  1 2  1 4  2 4  2 3  3 4  3 5  5 6  6 7  6 11  7 8  7 9  7 10  8 9  8 10  11 12  11 14  12 13  12 14  13 14 | 2 |
| 4  1  1 1  2  1 2  2 3  5  1 2  1 3  1 4  1 5  1 6  4  a b  b c  c d  d e | 0  1  4  1 |

Problem 2: Factorization

Description:

Everyone knows how to factorize, but your math department is interested in a specific type of factorization. They want to know what prime numbers will divide a specified integer evenly. However, they don’t need the exact prime factorization, rather, they simply need to know the number of distinct prime factors a number has. For example, 8 has a prime factorization of 23, but has just one distinct prime factor (2). 15 has the prime factorization 3\*5, and so has 2 distinct prime factors. Your task is to write a program that identifies the number of distinct prime factors a given integer has.

Input: factors.in

The input will begin with a number N (1<N<1000), the number of integers to be analyzed. N integers will follow, each of which is between 2 and 1,000,000 (inclusive).

Output: factors.out

For each integer in the input, write out a number representing the number of distinct prime factors the integer has, followed by a newline.

Samples:

|  |  |
| --- | --- |
| Input | Output |
| 1  15 | 2 |
| 5  15  8  10124  101243  602910 | 2  1  2  2  6 |
| 100  886823  567038  823628  987797  248079  630107  378836  963768  722887  802134  656071  619683  937598  691368  156667  559099  763899  994047  277947  218159  770312  769306  853168  551208  57266  11388  918924  917582  950207  746476  370523  722446  463218  150761  327939  421663  469995  610258  854924  463368  528311  187171  250595  413962  576577  100371  61147  149724  693225  657355  512183  64535  30299  566582  305730  599727  16370  124231  48065  409289  815121  250109  849643  497077  82522  894716  462121  808524  562652  762782  744915  412276  154090  659103  139992  191531  153265  697639  512455  216790  198948  116530  726216  329871  465480  202126  738870  743824  842078  755944  601743  761297  253997  786984  606678  874059  221020  601028  856568  115033 | 3  2  4  1  3  1  2  4  2  3  3  3  3  3  2  1  3  2  3  2  2  3  2  5  4  4  4  2  1  2  2  2  5  2  2  2  3  3  4  4  3  1  2  3  1  2  2  3  4  3  3  2  2  4  5  2  3  1  2  1  3  1  4  2  3  2  3  4  2  3  4  2  4  3  4  1  4  2  3  5  4  4  3  3  4  2  5  2  3  3  3  1  3  4  3  3  4  4  2  2 |

Problem 3: Heights

Description:

You and your classmates want to line up the entire school by height, tallest people at the front. If two students are of the same height, then seniority determines who goes first (seniors at the front, first-years at the back). Your job is to write a program that, given the students’ heights and grades, determine a suitable ordering. Be careful though – your school is quite large and contains 1,000,000 students!

Input: heights.in

Each line will contain an integer H (1<H<1000) representing the height of the student in centimeters. This integer will be followed by a space and then one of the following strings: “SE”, “J”, “SO”, or “F”, representing the grade of that student (SE = senior, J = junior, SO = sophomore, F = first-year). For example, an entry of “170 SO” would represent a sophomore who is 170 cm tall. There will be 1,000,000 such lines in the file.

Output: heights.out

Write out the sorted list of heights and grades, with each entry separated by a newline.

Samples:

These samples are abbreviated for clarity – the actual test file will contain 1,000,000 entries.

|  |  |
| --- | --- |
| Input | Output |
| 170 SO  170 SE  1000 F | 1000 F  170 SE  170 SO |
| 678 F  560 J  407 SE  784 F  795 SO  28 J  739 SO  785 SO  707 F  672 SE  26 F  6 SO | 795 SO  785 SO  784 F  739 SO  707 F  678 F  672 SE  560 J  407 SE  28 J  26 F  6 SO |
| 100 SE  100 SE  100 F  100 F  100 F  100 SO  100 J  100 SE  100 J  100 SO  100 F | 100 SE  100 SE  100 SE  100 J  100 J  100 SO  100 SO  100 F  100 F  100 F  100 F |

Problem 4: Encrypt

Description:

The Caesar cipher is a well-known cipher in which each letter is replaced with one a predetermined distance away from it in the alphabet. For example, if Caesar wanted to send the message, “HELP,” to another battalion, and they agreed that the “shift” was 3, he would send the message “KHOS.” Your job is to figure out what messages to send given a shift value and a message.

Input: encrypt.in

The first line of the input file will contain an integer N (1<N<100), representing the number of messages you must encode. Each message will be two lines. The first line will be another integer, S (-26<S<26), representing the number of letters to shift forwards (negative numbers indicate backwards shift). The next line will be the message to encrypt, and will be between 1 and 200 characters long.

Output: encrypt.out

Write out N lines, each containing the corresponding shifted message from the input file. Spaces and punctuation receive no shift. Be sure to preserve the case of the original message (lower case letters should remain lower case, uppercase letters should remain upper case).

Samples:

|  |  |
| --- | --- |
| Input | Output |
| 1  3  Help! | Khos! |
| 5  26  mEet at MidNighT.  14  Where are you?  -5  Is this the real life?  -26  Is this just fantasy?  7  Caught 1n a landslide. | mEet at MidNighT.  Kvsfs ofs mci?  Dn ocdn ocz mzvg gdaz?  Is this just fantasy?  Jhbnoa 1u h shukzspkl. |
| 2  1  What's 1+1?  -1  Is 9+10 really 21? I thought it was -5. | Xibu't 1+1?  Hr 9+10 qdzkkx 21? H sgntfgs hs vzr -5. |

Problem 5: Grammar

Description:

Υοur English teacher is fed up with students not proof-reading their papers before they turn them in. She has tasked you with writing a program to determine if there are any grammatical errors in the students’ papers, and if there are, to determine the number of such errors so that she can make appropriate deductions to their grades. She has given you a predetermined set of rules to check each essay against.

1. Apostrophes: Some students forget to add apostrophes where they belong or accidentally type the key where they don’t. Either mistake should be counted as one error.
2. Capitalization: Students often forget to capitalize proper nouns or capitalize random letters in other words. Any word that has its first letter capitalized will be considered a proper noun which should maintain its capitalization. However, if a letter other than the first letter of a word is capitalized, that should be counted as an error. Additionally, if the first word in a sentence is not capitalized, it should be counted as an error.
3. Spelling: If a word is not in the dictionary, the first letter is not capitalized (indicating it is a proper noun), and it is not a contraction, it should be counted as misspelled. This error applies to the entire word so if a student tries to spell “hamburger” but writes “laasdfd” or “hambirger,” both count as one error. Additionally, do not count a spelling error if a student uses a contraction correctly. So “would’ve” is a valid word, but “wouldve” would count as an error (either spelling, or apostrophe). “It’s” will never count as a spelling error but could count as an apostrophe error if used incorrectly.

These are simply some sanity checks so that your teacher can spend more time teaching and less time grading.

Input: grammar.in

The first 370101 lines of the input file will be a new-line delimited list of words to be used as the dictionary (already alphabetized). After this list will be an integer N (1<N<50), representing the number of essays to follow. Each essay will be one line long (no need to worry about indents or newlines). The ends of sentences will always be marked with some kind of punctuation (either a period, question mark, or exclamation point) followed by a space.

Output: grammar.out

For each essay, write the number of grammatical errors to the output file, according to the above rules.

Samples:

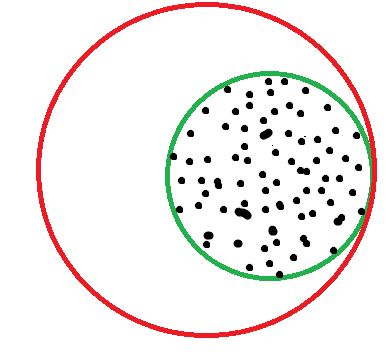
The dictionary has been omitted from the samples for clarity. The actual test file will contain the words at the top. Errors have been colored red.

|  |  |
| --- | --- |
| Input | Output |
| 1  Its a nice day outside. i wonder if it will rain. | 2 |
|  |  |
|  |  |

Problem 6: Cell Tower

Description:

You are working for a cell-phone carrier company that is looking into placing a new cell tower in the middle of a rural area. They want to ensure that they need only place one tower (as building each tower is quite expensive). As a result, they have tasked you with finding the optimal location to place this tower, the optimal location being defined as the location that minimizes the maximum distance between the tower and any town.



For example, here the center of the green circle is the optimal tower location because that location minimizes the maximum distance between the tower and the farthest town. The center of the red circle is not an optimal location, because it creates a larger distance between the tower and the farthest town.

Your job is to find the optimal (x,y) location of the tower, and to find the distance from the farthest town to the tower.

Input: tower.in

The first line of the input file will be an integer N (1<N<?), the number of test cases. Each test case will begin with an integer T (1<T<100), the number of towns to be covered by the cell tower. T lines will follow, each containing two space separated doubles (-1000<D<1000), representing the (x,y) coordinates of each town.

Output: tower.out

For each test case, write the (x,y) coordinates of the optimal tower location (separated by spaces) and the distance from the tower to the farthest town (also separated by a space) to the output file. Round all numbers to three decimal places. Answers for each entry will be accepted if they are within 5% of the correct answer.

Samples:

|  |  |
| --- | --- |
| Input | Output |
| 2  3  0 0  1 1  4 5  5  0 0  1.1 2.3  5.3 -6.1  -5 -7  1.8 -1 | 2.000 2.500 3.202  -0.110 -3.572 5.995 |
| 3  2  0 0  1 1  3  0 0  1 1  2 2  4  0 0  1 1  2 2  3 3 | 0.500 0.500 0.707  1.000 1.000 1.414  1.500 1.500 2.121 |
| 1  1  1 1 | 1.000 1.000 0.000 |

Problem 7: Perm

Description:

Your English teacher has enlisted your help again. They have various books that they want arranged on their shelves, and want to know the number of ways to do so. Actually, more than that, they would like to know every possible arrangement of books. They have therefore tasked you with creating a program to output every possible permutation of books they have.

Input: perm.in

The input file will contain a space separated list of book titles followed by a newline. No book title will contain a space. There will be a maximum of 9 distinct books, but an absolute maximum of 15 books (there can be repeats). There will be at least one book in the sample file.

Output: perm.out

Write each possible ordering of the books on the shelves to the output file. Identical orders should be omitted, that is, if you have already stated a possible order, you should not repeat it later on. For example, if your teacher had two copies of “Faust,” you should only print out “Faust, Faust,” not “Faust, Faust\nFaust, Faust.”

Samples:

|  |  |
| --- | --- |
| Input | Output |
| Mockingbird Mockingbird Mockingbird | Mockingbird, Mockingbird, Mockingbird |
| Faust Mockingbird Potter | Faust, Mockingbird, Potter  Faust, Potter, Mockingbird  Mockingbird, Faust, Potter  Mockingbird, Potter, Faust  Potter, Faust, Mockingbird  Potter, Mockingbird, Faust |
| Faust Potter Faust | Faust, Potter, Faust  Faust, Faust, Potter  Potter, Faust, Faust |