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:

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 (connectivity would be maintained even if any node were removed).

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:

For each integer in the input, print 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:

Print 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 |
|  |  |
|  |  |
|  |  |

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.