TRI

The number of combinations for the operator and equals sign is small enough to check each of them separately. Two errors to avoid are printing only one solution when there are multiple possible outputs and not using integer division which truncates the decimal part (a good way to work around this is to check if A*C=B instead of A=B/C).

PASCAL

Directly implementing the given algorithm is too slow.

A more efficient approach is to realize what the given program does: it calculates the difference between N and its largest divisor. We can find the largest divisor of N by dividing N by its smallest divisor greater than 1.

If N is prime, then the solution is N-1. Otherwise, its smallest divisor is at most $\sqrt{N}$ so we can use trial division to find it.

JABUKE

We are given the formula for the area of a triangle in the problem statement so we only need to find a way to check if a point P is inside triangle ABC.

The check can be done in more than one way, one of which only calculates areas of triangles. More precisely, the sum of areas of triangles ABP, ACP and BCP is equal to the area of ABC if and only P is inside ABC.

AVOGADRO

The algorithm we use is:

1. If all three rows contain the same numbers, we are finished. If not, there is at least one number B which does not occur in every row.
2. Find all columns that contain B and delete them.
3. Return to step 1.

Although the algorithm is conceptually simple, efficiently implementing it is not.

We maintain a set of numbers that need to be removed from the table. First we put into the set all numbers not occurring in all three rows. Then we pick the elements from the set one by one and delete columns that contain the chosen elements.

To quickly locate all columns containing a number, we preprocess the table, writing for each number the columns it occurs in. We also keep track how many times a number occurs in each row.

When deleting a column, we decrease the occurrences of each number in the three rows and, when one of those counters becomes zero for any number and any row, mark that number for deletion.

BARICA

The core of the solution is a dynamic programming algorithm. To start with, sort the points in ascending order of x-coordinates, breaking ties in ascending order of y-coordinates.

Let dp(L) be the largest amount of energy Barica can have after getting from plant 1 to plant L.

Barica can reach plant L jumping to the right or upwards. As the length of the jump is insignificant, of all plants P with the same x-coordinate (and smaller y-coordinate) as plant L, we want the one we can reach with the largest amount of energy, that is for which dp(P) is largest. We apply the same reasoning to plants with the same y-coordinate.

Finally, dp(L) is calculated as the better of two cases (jumping upwards from the best plant with the same x-coordinate, or jumping to the right from the best plant with the same y-coordinate).

To reconstruct the frog's route, remember for each plant the best plant to have come from.

BAZA

Sort the words in alphabetic order. In the resulting table every word in the database can be searched for with the following algorithm.

We start the search on an interval containing all words. Using binary search, we find the interval of words starting with the same letter as the query word. Note the interval boundaries and proceed with the second letter. Binary searching on the new interval, we find the interval of words which share both the first and second letters as the query word. Note the interval boundaries again and move on.

Suppose we insert the same words from the database into a data structure, in order in which they are given in the input. Then it is possible to calculate, for every prefix of every word, how many words before it share the prefix.

Now attach a counter to every interval of consecutive equal letters in a column. When inserting words into the data structure, for every interval the word belongs to, we increase the counter. Every counter tells us exactly how many words there are with that prefix.

After preprocessing the input, for any word W and each of its prefixes P we know the numbers:

- counter(P) – how many words start with P
- before(W,P) – how many words are before word W and start with P (including W itself)
- id(W) – the identifier of word W (when it first appears in the database)

Now every query word Q can be answered:

- If Q is in the database, then we output id(Q) + before(Q, P_1) + before(Q, P_2) + … + before(Q, P_L); where L is the length of word Q, and P_k is a k-th prefix of word Q.
- Else output N + counter(P_1) + counter(P_2) + … + counter(P_L).

The time complexity of the preprocessing phase is O(L·N·log N), while answering a query is O(L·log N), where L is the length of a word, and N is the number of words in the database.

For implementation details see the accompanying source code.