



Editorial

Tasks, test data and solutions were prepared by: Marin Kišić, Pavel Kliska, Vedran Kurdija, Daniel Paleka, Stjepan Požgaj and Paula Vidas.

Implementation examples are given in attached source code files.

Task Bold

Prepared by: Daniel Paleka and Paula Vidas

Necessary skills: 2D array

We make a new $n \times m$ matrix and fill it with ' '. Then we go through the original matrix, and when we find '#' on some position (i, j) , we put '#' on positions (i, j) , $(i, j + 1)$, $(i + 1, j)$ and $(i + 1, j + 1)$ in the new matrix.

Task Alias

Prepared by: Marin Kišić and Vedran Kurdija

Necessary skills: Dijkstra's algorithm

We can think of Rafael's database as a directed weighted graph. Since vertices are words, we need to first assign a unique integer between 1 and n to each word to make the implementation easier.

Notice that the task asks for the length of the shortest path from vertex a to vertex b . Therefore, 20 points can be scored using brute force in $O(n!)$ complexity. For additional 20 points, we can use the [Floyd-Warshall algorithm](#) to calculate distances between every two points in $O(n^3)$ complexity, and then answer the questions in $O(1)$.

For all points we can use [Dijkstra's algorithm](#) for each question, which has $O(m \log n)$ complexity. The total complexity of our solution is then $O(qm \log n)$.

Task Anagramistica

Prepared by: Marin Kišić and Pavel Kliska

Necessary skills: dynamic programming, combinatorics

The first subtask can be solved by iterating over all possible subsets and counting the number of similar pairs in each one.

Since each element is either in the chosen subset or not, that leads us to a dynamic programming solution, where $dp[n][k]$ is the number of ways to choose among the first n words a subset with exactly k similar pairs. The transition would be:

$$dp[n][k] = dp[n - 1][k] + dp[n - 1][k - x],$$

where x is the number of new similar pairs that appear when we add the n -th word to the subset. Unfortunately, since we don't know which words are in the subset, we can't know x .

But, we can use a similar approach if we notice that if we sort the letters in each word, then the words are similar if and only if they are equal.

After sorting, we get a multiset of words, with m distinct elements, and the i -th element appears a_i times. Let $dp[m][k]$ be the number of ways to choose among the first m different words a subset with exactly k equal words. The transition is:

$$dp[m][k] = \sum_{i=0}^{i \leq a_i, \frac{i(i-1)}{2} \leq k} \binom{a_i}{i} dp[m-1][k - i(i-1)/2].$$



The initial values are $dp[0][k] = 1$, for $k = 0$, or 0 otherwise.

Since the numbers in the binomial coefficient are at most n , we can precompute them using Pascal's triangle. The complexity for each k is $O(n)$, since the sum of all a_i is equal to n , so the total complexity of the solution is $O(n^2 + nk)$.

Task Geometrija

Prepared by: Paula Vidas

Necessary skills: ccw, triangulation, ad hoc

The first subtask can be solved in $O(n^4)$ complexity by simply checking for each segment if it crosses with other segments.

To check if two segments cross, we will use the function

$$ccw(A, B, C) = x_A \cdot (y_B - y_C) + x_B \cdot (y_C - y_A) + x_C \cdot (y_A - y_B).$$

This is equal to twice the signed area of the triangle ABC (which is not important in this task), and is positive when A , B and C are in counterclockwise order (*ccw order* for short).

Segments \overline{AB} and \overline{CD} cross if and only if

$$ccw(A, B, C) \cdot ccw(A, B, D) < 0 \quad \text{and} \quad ccw(C, D, A) \cdot ccw(C, D, B) < 0.$$

(since the points are in general position in this task, we don't have to deal with special cases that would arise if some three were collinear). The proof is left as an exercise to the reader.

We say a segment is *good* if it doesn't cross with any other segment. For the rest of the solution, we use the observation that every good segment necessarily belongs to every triangulation of the given set of points. *Triangulation* of some set of points on the plane is a division of the convex hull into triangles with vertices in given points, such that every point is a vertex of at least one triangle. Equivalently, it is a maximal set of non-crossing segments between the given points. A triangulation can have at most $3n - 6$ segments, so if we find some triangulation, we get just $O(n)$ candidates for good segments. The proof of the claims above is left as an exercise to the reader.

Therefore, the second subtask can be solved in $O(n^3)$ complexity. We can build a triangulation in $O(n^3)$ by going through all the segments, checking if it crosses with the already added segments, and if not, we add it to the triangulation. We then check for every segment we added, in $O(n^2)$ complexity, if it crosses with each possible segment.

We will solve the third subtask in complexity $O(n^2 \log n)$. Finding a triangulation is the easy part. There is a lot of ways to do it, fastest in complexity $O(n \log n)$, but we will describe an easier algorithm with $O(n^2)$ complexity. First, we find the convex hull, and triangulate that polygon in some way (for example by taking all diagonals from some point). Then, we go through all interior points, find the triangle that contains the current point, and divide it into three smaller triangles.

Now we describe how to check in $O(n)$ complexity if some segment is good. Consider a segment \overline{PQ} . To make it easier to describe the solution, imagine without loss of generality that the line PQ is vertical and P is above Q , as in the figure below. Let $A_1, \dots, A_s, B_1, \dots, B_t$ be all other points, in ccw order around P , so that for all A_i it holds $ccw(P, Q, A_i) < 0$, and for all B_j it holds $ccw(P, Q, B_j) > 0$ (see the figure).

Consider some point A_i . Let $f(i)$ be the maximum j such that $ccw(A_i, P, B_j) < 0$ (if it exists). Points $B_1, \dots, B_{f(i)}$ are exactly those points on the right side that are below the line A_iP . It's easy to see that f is increasing, so it can be calculated using the two pointers method.

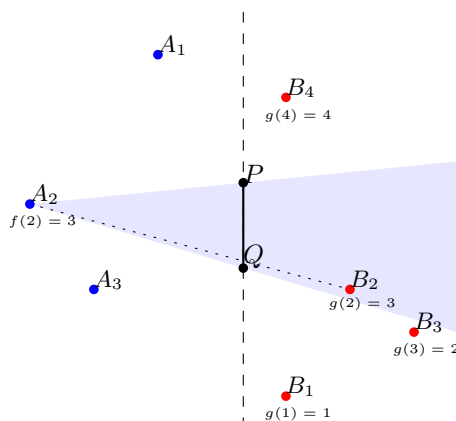
Consider the ccw order of points B_1, \dots, B_t around Q (where we take the first point to be the one with maximum angle $\angle PQB_j$). Let $g(j)$ be the position of B_j in this order.



Let $j' = \arg \max_{1 \leq j \leq f(i)} g(j)$ (the j among $1, \dots, f(i)$ for which $g(j)$ is maximal). We claim that it's enough to check if segment $\overline{A_i B_{j'}}$ crosses with segment \overline{PQ} , i.e. if it doesn't, then the segment $\overline{A_i B_j}$ for any other j also doesn't cross with \overline{PQ} .

It's clear that $\overline{A_i B_j}$ and \overline{PQ} cross if and only if point B_j is below the line $A_i P$ and above the line $A_i Q$. Among the right points that are below $A_i P$, point $B_{j'}$ is "furthest around Q ", so if it's below $A_i Q$, then all other points are too.

So, for each point A_i we need to find its j' (it can easily be updated when we calculate f) and check whether segments $\overline{A_i B_{j'}}$ and \overline{PQ} cross. If there is no such i , then \overline{PQ} is a good segment.



The ccw order of points around some center point can be found directly by sorting the points in $O(n \log n)$ complexity. It's worth noting that there exists a (much more complicated) algorithm with $O(n^2)$ complexity that determines this for all points together (so the total complexity of the solution of this task would be $O(n^2)$). It uses [duality](#) to transform it into the [line arrangement problem](#).

Task Index

Prepared by: Marin Kišić

Necessary skills: binary search, persistent segment tree

First subtask can be solved by brute force.

The idea for the rest of the solution is to use binary search to determine the h-index. We need to be able to answer questions of the form "How many numbers on positions l through r are greater than or equal to h ?" efficiently. The difference between the second and third subtask will be in the data structure we use to answer such queries.

One possible approach is to build a segment tree, where the node corresponding to the interval $[a, b]$ holds the numbers p_a, p_{a+1}, \dots, p_b sorted in increasing order. When we query, we use binary search in the appropriate nodes to find how many numbers are greater than h in the corresponding interval of the node. Therefore we can answer each question from the problem in $O(\log^3 n)$ complexity, which is fast enough for the second subtask.

For all points, there are multiple approaches. One is to use parallel binary search (most solutions on HONI, local version of COCI, were of this type). We will describe a different solution that uses a persistent segment tree. Times will be the prefixes of the array, that is, we will have a "separate" segment tree for each prefix. A node in the tree corresponding to the interval $[a, b]$ will hold the value $cnt_a + cnt_{a+1} + \dots + cnt_b$, where cnt_i is the number of elements in the prefix that are equal to i .

Now we can answer the question "How many numbers on positions l through r are greater than or equal to h ?" by querying the sum of values in the interval $[h, 200\,000]$ in moments r and $l - 1$, and subtracting



them. The complexity per question from the problem is $O(\log^2 n)$.

For those who want to know more: there is a solution with $O(\log^2 n)$ complexity per question, that uses the same data structure as described for the second subtask. It can be speed up using a technique known as *fractional cascading*. You can read more about it [here](#).