

Task MARKO	Author: Dominik Gleich
-------------------	-------------------------------

The simplest solution for this task is to transform each word from the dictionary into a sequence of keypresses that result in the given word. Therefore, we transform each word into that sequence of keypresses, i.e. "ana" -> "262". Finally, we simply count how many "transformed" words from the input are equal to the input sequence of keypresses.

Necessary skills: arrays, strings

Category: ad-hoc

Task GEPPETTO	Author: Dominik Gleich
----------------------	-------------------------------

In this task, we need to count how many combinations, out of the 2^N possible, meet the requirements for combining the ingredients.

To begin with, for each ingredient **i**, we can create an array of ingredients **j** that cannot be combined with that ingredient.

After this, we solve the task by recursively looping through the ingredients and trying to do the following:

- a. Choose the ingredient - only if we haven't already chosen an ingredient that cannot be combined with the current one.
- b. Do not choose the ingredient.

For implementation details, consult the official solution.

Necessary skills: arrays

Category: ad-hoc, recursion

Task ARTUR	Author: Ivan Paljak
-------------------	----------------------------

Let us first notice that there is always going to be at least one stick that can be removed from the table in a manner that follows the rules of the game. In other words, a solution always exists.

The first step towards solving the task is answering the question: "Is stick **B** in the way of moving stick **A**?". We observe the **projections** of the corresponding line segments on the x-axis. A projection of a line segment with ending points (x_1, y_1) and (x_2, y_2) corresponds to the line segment with ending points $(x_1, 0)$ and $(x_2, 0)$. If the observed projections don't have points in common, then the removal of sticks **A** and **B** is mutually independent. Otherwise, let $(x', 0)$ denote one of the mutual points of two projections. Let T_A denote the intersection of line $x = x'$ with stick **A**, and T_B denote the intersection of that line with stick **B**. Stick **B** is in the way of moving stick **A** if T_A is located above T_B . Therefore, the answer to the given question is obtained in $O(1)$.

Let us imagine that each line segment corresponds to a node in a directed graph and that there is an edge from node **A** to node **B** if and only if stick **B** is in the way of moving stick **A**. Now we need to find a sequence of nodes so that it holds for each directed path between **a** and **b** that node **a** is located before node **b** in that sequence. This order of nodes is called **topological sort** and it is found using a slight modification of depth-first search (DFS). More precisely, after we have expanded from one node to all its neighbours, we add that node to the end of the sequence. This way, we have made sure that all line segments in the way of moving the current one are removed before we remove the current line segment.

The time complexity of this algorithm is $O(N^2)$ because of building the graph. The topological sort itself is as efficient as a DFS traversal of the graph. For implementation details, consult the official solution.

Necessary skills: geometry, topological sort

Category: graph theory

Task SAVEZ	Author: Zvonimir Medić
-------------------	-------------------------------

In order to solve this task, two things need to be noticed:

1. If a string x_i is a prefix of string x_j , and x_j prefix of string x_k , then string x_i is a prefix of string x_k ($i < j < k$).
2. If a string x is a suffix of string y , then string x^R is a prefix of string y^R , where a^R denotes string a written backwards.

Now it is sufficient to build a prefix tree in a slightly modified form:

1. The alphabet doesn't consist of 26 letters, but 26^2 , because each pair of letters represents a single "letter".
2. When inserting a word in the tree, the first letter in the pair is taken from the beginning, the other from the end.
3. Each node stores the best solution for that "prefix" ($1 + \max(\text{node_children})$).

If we start from the end of input data, we've changed the task from expanding to contracting one of the existing strings. By doing this, we ensure that the solution takes into account the order of words from the input data. The time complexity of this solution is $O(\text{total_length_of_words})$.

Necessary skills: prefix tree, dynamic programming

Category: ad-hoc

Task VUDU	Author: Dominik Gleich
------------------	-------------------------------

In this task, we need to find the number of consecutive subsequences such that the average value is $\geq P$.

To begin with, it is important to notice that, if we subtract the value P from each number, we have reduced the task to finding the number of consecutive subsequences such that their sum is non-negative.

Therefore, now the task is to find the number of consecutive subsequences that have the sum of values non-negative in the sequence of numbers. We can do this in two ways, one of which will be described here, while the official solution implements the other.

Let $\text{pref}(x)$ be the sum of the first x elements in the transformed sequence. It is easy to see that the sum of elements in the interval $[L, R] = \text{pref}(R) - \text{pref}(L - 1)$. Because we are searching for a number in the interval $[L, R]$, we can calculate how many positions L there are so that the sum in the interval $[L, R]$ is non-negative for each R . Therefore, for each $\text{pref}(R)$ we ask how many $\text{pref}(L - 1)$ there are such that $L \leq R$ and $\text{pref}(R) \geq \text{pref}(L - 1)$. If we imagine that we are moving from left to right in the array, for each R we need to be able to answer how many $\text{pref}(L - 1) \leq \text{pref}(R)$, $L \leq R$ there are.

Because we only want to know about the relative order between the prefix sums, we can hash the value of each prefix sum and convert it to a single number from the interval $[0, N]$, depending their relative order. The hashing needs to be implemented in order to efficiently answer to queries using a data structure.

The simplest data structure to implement that enables inserting number **X** and querying how many numbers smaller than **Y** there are is the Fenwick tree data structure. For details about this data structure, consult http://wiki.xfer.hr/logaritamska_tutorial.

Necessary skills: data structures

Category: ad-hoc, sweep

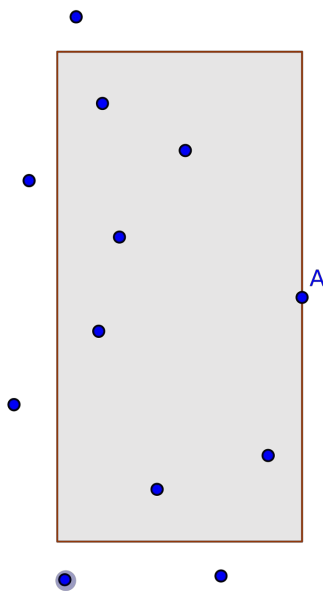
Task DRŽAVA	Author: Mislav Balunović
--------------------	---------------------------------

There are two conceptually different solutions that are both based on the following observation:

*If there is a county with at least **K** cities in it, the prime minister will be happy.* The proof of this relatively simple claim is obtained by using Dirichlet's principle and is left as an exercise to the reader.

First solution:

Let us notice that we can use binary search by distance **D** if for a fixed **D** we know how to check if there is a layout of roads such that the prime minister is happy.



For each city, let us define a rectangle (so-called "bounding box") that contains all points with the smaller x coordinate, which x and y coordinates differ for at most **D** from the city we are currently observing.

It is sufficient to notice that, if there are more **8K** cities in that rectangle, then there exists a county with at least **K** cities. The reason for this is that a rectangle can be split into **8** smaller squares of side length **D/2**. All cities inside the smaller square are in the same county and, given the aforementioned lemma, we know that we can make the prime minister happy. Additionally, all cities with distance to the current city smaller than **D** are located inside the rectangle.

Let us sort the cities by their x coordinate and process them in this order using the sweep line technique (http://wiki.xfer.hr/sweep_line/).

When processing a city, we insert it into a binary tree where the cities are sorted by their y coordinate. We remove from the tree all cities which x coordinate differs from the x coordinate of the current city for more than **D**.

Now we can efficiently iterate over all cities inside the rectangle and connect with roads the cities that are distant for at most **D** from the current city. If at some point we notice that a rectangle contains more than **8K** points, we can conclude that we can make the prime minister happy and we don't need to check any further.

After that, we perform a DFS algorithm in order to form counties and, in the end, use dynamic programming to check if there is a subset of cities in a county that has the sum divisible by **K**.

The time complexity of this algorithm is $O(N * K * \lg D_{MAX})$ where **D_MAX** is the maximal possible value of number **D**.

Second solution:

This solution takes advantage of the fact that, for each city, it is enough to know of its **K** closest cities.

(That fact is obtained directly from the first lemma.)

The solution consists of two parts:

- 1.) Finding the **K** nearest neighbours for each point.
- 2.) Forming counties and checking if the newly made county contains the required subset of cities.

We process the cities from left to right and use **P** to denote the minimal **P** such that we have found a city with at least **K** neighbours so far that are distant for at most **P**. For the current city we calculated the same rectangle from the first solution, go over all points in the rectangle and find the **Kth** nearest point from the rectangle and change the value **P**. Notice that the rectangle consists of at most **8K** points using a similar argument to the one from the first solution.

This part of the solution is of complexity $O(N * K + N \lg N)$.

When we have found $N * K$ potential roads for each city, we sort these roads by length and connect them respectively. The counties are maintained using the union find data structure (http://wiki.xfer.hr/union_find/). When connecting two cities from different counties, we go over all the cities from the smaller county, insert them into the larger county and calculate the remainders **mod K** from the subsets of the newly formed county. With careful implementation, this part of the solution is of complexity $O(N * K * \lg N)$ if we use the algorithm that joins the smaller component to the larger one in union find.

For implementation details, consult the official solution.

Necessary skills: sweep line, dynamic programming, Dirichlet's principle

Category: sweep line