Spoiler

1 stairway

If each height only occurs once, the result is 1. For each height that appears k times, there are $k!$ possible permutations. The overall result is $\prod_h \text{frequency}(h)! \mod 1\,000\,000\,007$.

The most common error was an integer overflow because the modulus was not applied often enough.

2 chase

Subtask 1

Simply run Dijkstra.

Subtask 2

First run Dijkstra from the police patrol to determine the earliest point when the police can arrive at any point.

Then run Dijkstra from Dieters position. If any point can only be reached after the police, ignore it. It is actually sufficient to check this for the goal position n: If any position on the shortest path is reached by the police before Dieter, the result is -1, because the police will also reach position n before Dieter. Otherwise, the result is the length of the shortest path without police.

Subtask 3

Running Dijkstra k times is too slow. Use the same trick as for closingtime3 in the midterm (described in the last tutorial).

3 supersequence

It is possible to calculate the result with dynamic programming on $\mathcal{O}(|s_1| \cdot |s_2|)$.

If you have solved the exercise LCS from week 10, the easiest way to implement this is $|s_1| + |s_2| - \text{lcs}(s_1, s_2)$.

4 park

Subtask 1

When calculating the minimum spanning tree, the result not only has the minimum sum of edge weights, but also a minimal maximal edge weight. Therefore you can implement Prim or Kruskal and print the maximal weight of any edge that was used.

Alternatively, it is possible to use binary search to search for the largest edge weight that is used. For any edge weight w, simply check if, starting from the entrance, the graph is connected using only edges with weight $\leq w$.

Subtask 2

If you used binary search for subtask 1, the solution is easy to adapt: Simply adapt your check function to check if at least $n - k$ nodes are connected to the entrance.

There are also solutions with both MST algorithms:

- Prim: Start from the entrance and add edges until $n - k$ nodes are connected.

- Kruskal: In union-find, remember for each component how many nodes it contains. As soon as the component containing the entrance contains $n - k$ nodes, print the largest weight used so far.

5 jumping

Subtask 1

Many people attempted dynamic programming in $\mathcal{O}(n^2)$. This is too slow even for subtask 1, since n can be $100\,000$. However, there is a dynamic programming solution in $\mathcal{O}(\max_i h_i \cdot n)$: For each height, remember the best solution so far that ends at a pillar of that height. Now iterate over all pillars from 1 to n and update position h_i:

$$dp(h_i) := \max\{dp(h_i), \max_{h_i - d \leq j \leq h_i + d} h_j + 1\}$$

Subtask 2

Instead of storing the intermediate dp values in an array, use a segment tree that stores maximums and allows for range queries and point updates.

For each position, you can now query the largest value within the correct height range and update the value at h_i in logarithmic time. The overall running time is then $\mathcal{O}(n \log h_{max})$.