Task idea: *Tomasz Idziaszek*
Solutions, tests: *Aleksejs Zajakins, Eduards Kaļiņičenko, Rihards Opmanis, Jan Kanty Milczek*
Text: *Eduards Kaļiņičenko*

Viruses

From reading the task you may think that there aren't enough constraints. This is not true as you actually have enough information.

- k. You are given that the sum of all values k does not exceed 100, so naturally, $1 \leq k \leq 100$.
- l. You are given that the sum of all values l does not exceed 50, so naturally, $1 \leq l \leq 50$.
- N. You are given that the sum of all values k does not exceed 100 and that $k \geq 1$ in every row of the mutation table. Thus, there are at most 100 rows, meaning $G - 2 \leq N \leq 100$. Since $G > 2$, $0 < N \leq 100$.
- G. You are given that every integer from 2 to $G-1$ appears in the table as a at least once. This means that $N \geq G - 2$ (which you are also conveniently given). Hence, $2 < G \leq N + 2$, or remembering constraints on N, $2 < G \leq 102$.
- M. You are given that the sum of all values l does not exceed 50 and that $l \geq 1$ for every antibody. Thus, there are at most 50 antibodies, meaning $0 \leq M \leq 50$.

Note that this means that a test with $G = 102$ is a valid test, and it may fail some solutions. We were nice though, and only put it in the first subtask, so if you're failing that one, this may be the answer why.

In all subtasks, we'll use the same approaches, which will be similar to dynamic programming and Dijkstra's algorithm. Imagine we are computing dp_i – the minimal length of a virus that we can obtain from gene i and is not detected by any antibodies. Then, similar to Dijkstra, we can take the smallest unprocessed value of dp_i and process it, that is, for every transition $a \to \langle b_1\ b_2\ \ldots\ b_k \rangle$, where for some j, $b_j = i$, we can update the value of dp_a using that transition. Since every transition is non-empty, we know that once we picked something as the smallest unprocessed value of dp_i, it cannot be updated to a better result anymore, hence it's our answer. Once you have no more unprocessed values (or they're all ∞), you're done.

Subtask 1 (No antibodies ($M = 0$))

In this subtask we're not interested in viruses themselves, just in their length as any virus is valid. So, just do what was discussed above.

Initially, $dp_0 = dp_1 = 1$. For any other i, $dp_i = \infty$. Then repeatedly pick the smallest unprocessed value of dp_i and process it — for every transition $a \to \langle b_1\ b_2\ \ldots\ b_k \rangle$, $dp_a = \min(dp_a, dp_{b_1} + dp_{b_2} + ... + dp_{b_k})$. You don't really need to even do transition selection or a queue for minimal values here, the most basic implementation will yield $O(G \cdot (G + \sum l))$.

Subtask 2 ($N = G - 2$)

In this subtask, due to the restriction that every integer from 2 to $G - 1$ appears in the table as a at least once, every gene has strictly one outgoing mutation. This means that from every gene, there can be either no or a single virus only.

However, viruses can still be quite large, so you can't just compute it. It could also be infinite. Luckily, you can use similar approach here, except this time you'd also need to store some information about the virus itself rather than only the length of it.

What information do we need about the virus then? Well, our dynamic programming state already has a condition that it's the shortest virus that is not detected by any antibodies. So, our initialization is changed slightly:

$$dp_0 = \begin{cases} 0, & \text{if 0 is an antibody} \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

$$dp_1 = \begin{cases} 0, & \text{if 1 is an antibody} \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

Now, what about transitions? Similarly, $dp_a = \min(dp_a, dp_{b_1} + dp_{b_2} + ... + dp_{b_k})$. We also know that there can't be an antibody fully inside each individual dp_{b_j}. But what about overlaps? Well, since each antibody length is at most ≤ 50, we know we are only interested in storing the first and the last 50 characters for every value of dp. Then when we are doing a transition and gluing up multiple states together, we just have to check if due to this an antibody won't appear in the result, which will make this transition invalid. It's possible that the most inefficient way of doing so will time out, but you have plenty of leeway here, so it shouldn't cause too many issues.

Subtask 3 (One antibody ($M = 1$))

Here we need to expand our DP a bit. Given that unlike in the previous subtask, there can be multiple different viruses originating from a single gene, it's no longer enough to store the information about the virus, we need to encode it inside a state. So, the general approach will be as follows:

We are now calculating $dp_{i,st,en}$, the minimal length of a virus such that we start in the state st, obtain a virus from the gene i that is not detected by any antibodies and end up in the state en. The transition then becomes where for transition $a \to \langle b_1\ b_2\ \ldots\ b_k\rangle$; $dp_{a,st,en} = \min(dp_{a,st,en}, dp_{b_1,st,x_1} + dp_{b_2,x_1,x_2} + \ldots + dp_{b_k,x_{k-1},en})$.

But wait! We have to now compute for every value of $k-1$ variables of x, and $k \leq 100$! Well, we can notice that we can do a second dynamic programming, somewhat reducing the running cost, but this will only be enough for Subtask 4. Instead we can transform the transitions themselves.

What we want to do is to make sure that in every transition $a \to \langle b_1\ b_2\ \ldots\ b_k\rangle$, $k \leq 2$. We can do that, since

$$a \to \langle b_1\ b_2\ \ldots\ b_k\rangle = \begin{cases} a \to \langle b_1\ z\rangle, \text{where } z \text{ is a new gene} \\ z \to \langle b_2\ \ldots\ b_k\rangle \end{cases} \tag{3}$$

reduces the length of a transition. Note that by doing so we would have at most $\sum k$ transitions still and the sum of all values k would still be $O(\sum k)$, since it could only double worst-case. We are also creating new genes, but similarly, we'll still have $O(G)$ of them.

However, now transition is a lot smoother. For a transition $a \to \langle b_1\rangle$, we have $dp_{a,st,en} = \min(dp_{a,st,en}, dp_{b_1,st,en})$. For a transition $a \to \langle b_1\ b_2\rangle$, we have $dp_{a,st,en} = \min(dp_{a,st,en}, dp_{b_1,st,x} + dp_{b_2,x,en})$

Now, remember that from $dp_{i,st,en}$ we only need to consider transitions that contain i as the right hand side. Now, imagine we fix st and en and iterate freely over i. It's clear that we will consider every transition at

most twice. This means that to compute all $O(G \cdot S^2)$ states we need to process $O(G \cdot S^2)$ transitions. As such, if we assume that the amount of states is S, then if we're using fast structure to find minimal states, the complexity would be $O(G \cdot S^2 \cdot (S \cdot \log(G \cdot S^2)))$. The last logarithm comes from the necessity to update the queue of the smallest values every time we update a value successfully.

So, the final question we need to answer is what is the state here? We can observe that since we have one antibody, it's enough to have a state encoding at what length of prefix for the antibody you are. For an antibody of length l, this gives you l states (because you can't visit prefix of length l as that just means that you can detect this virus). So, you are now looking for $dp_{i,st,en}$, the minimal length of a virus such that we start by already having part of the virus ending with a prefix of antibody of length st, obtain a virus from gene i that is not detected by any antibodies and end up with a virus ending with a prefix of antibody length en.

Now, your answer for a gene i is $\min(dp_{i,0,x})$ for any $0 \leq x < l$. The transition doesn't really have any additional work either as the fact that we are not having an antibody in the virus is encoded in the states themselves. The initialization is where all of the work happens now. What we can do is to iterate over terminal genes i (0 and 1) and starting states st. Then, let's observe the virus obtained from virus corresponding to the state st (e.g. prefix of antibody of length st) and the gene i appended to it. Well, we get a string and need to check what is the maximum its suffix that is also a prefix for the antibody. Doing so naively is enough, but you can also observe that it's exactly what KMP algorithm is looking for. Let's call this new length en. Then we've found that for this i and st, as long as $en < l$, $dp_{i,st,en} = 1$. Everything that's not initialized after we've iterated over every i and st has a value of ∞ as it's impossible.

So, our $S = \sum l$ and as such our total complexity is $O(G \cdot (\sum l)^3 \cdot \log(G \cdot (\sum l)^2))$.

Subtask 5 (No further constraints)

We can use the same solution as Subtask 3, but the state needs to change. Now that we have multiple antibodies, we need to compute a set of prefixes for all antibodies, where some of them would be invalid. Please note that the definition of a bad prefix isn't if it's equal to one antibody or not. The prefix is bad if and only if it has a suffix that is an antibody, since we may have that a prefix of one antibody already ends with a different antibody.

Now a similar initialization can occur, where we would iterate over every i and st, obtain a string from st and i and find its largest suffix that is in the prefix set to make the transition. Doing so naively is enough, but once again, you can also observe that it's exactly what the Aho-Corasick algorithm is doing.

We can also see that we'll still have at most $O(\sum l)$ states, and as such the rest of the algorithm and the total complexity is the same as in Subtask 3.

Subtask 4 (The sum of all values l does not exceed 10)

This was a subtask designed to award points to conceptually right solutions but inefficiently written. Such as, for example, not implementing fast enough structure for picking the minimal values of dp or not transforming the transitions and as such having to write a separate dynamic programming for executing the transitions.