**Code Jam 2020 - Round 2**

# Analysis: Security Update

Let $R_i$ be the number of computers that receive the update before computer i, and $T_i$ be the time between computer 1 and computer i receiving the update. For each i, the input gives us exactly one of these numbers. We can set $R_1 = T_1 = 0$ for convenience.

## A simplified problem

Let us assume for now that we have all the $T_i$s. If computers i and j share a direct connection and $T_i$=$T_j$, then any path that gets to computer i in time $T_i$ does not go through computer j, and vice versa, because all latencies are positive. Therefore, we can assign any positive latency to all those connections. If computer i has a given $T_i$, it means that any connection coming from computer j with $T_j < T_i$ needs to have a latency of at least $T_i$-$T_j$, or otherwise the update could get to computer i in less than $T_i$ time by getting to computer j in $T_j$ time and then using that connection. In addition, for at least one j, the latency of the connection between i and j has to be exactly $T_i$-$T_j$, or otherwise the time for the update to reach computer i would be larger than $T_i$. One simple way to solve this problem is to make all connections between computers with different T values have a latency of exactly $|T_i-T_j|$; this takes O(**D**) time.

Notice that the algorithm above finds a valid assignment for any set of $T_i$s. To solve the actual test sets, we are left with the problem: given some $T_i$s and some other $R_i$s, assign all of the non-given values in a way such that sorting the computers by $T_i$ leaves them sorted by $R_i$, and vice versa. In particular, computers with equal T values should have equal R values, and vice versa.

## Test Set 1

In this test set, we can solve the subproblem from the previous section by setting $T_i := R_i$.

## Test Set 2

For Test Set 2, we again focus on solving the subproblem. We do that by first ordering the computers by what is going to be their final $T_i$ value (or equivalently, by their final $R_i$ value). We can partition the set of computers other than the source computer into two: those for which we know $R_i$ (part R) and those for which we know $T_i$ (part T). We can sort each of those in non-decreasing order of the known value. We now have 2 sets that are in the right relative order, and need to merge them as in the last step of [Merge Sort](). We assign the source computer first. Then we iterate through the remaining **C**-1 slots in order. Suppose we have already merged N computers, and let computer k be the last one of those. Let i and j be the first computers remaining in parts R and T, respectively. If $R_i \le N$, we take computer i next and assign $T_i := T_k$ if $R_i = R_k$, and $T_i := T_k+1$ otherwise. If $R_i > N$, we take computer j next and assign $R_j := R_k$ if $T_j = T_k$ and $R_j := N$ otherwise.

We can prove that if the original set of values is consistent with at least one latency assignment (which the statement guarantees), this generates a valid order and assignment of missing values, and moreover, it generates one in which the T value of the last computer in the order is minimal. We do that by induction on the number of computers. For a single computer, this is trivially true. Suppose we have **C** > 1 computers. By our inductive hypothesis, the first **C**-1 computers in the order were ordered and assigned values in a consistent way, with a minimal T value for the last computer among all options. Let us say the last computer in the full order is computer i, and the next-to-last computer is computer j. By definition of how we assign missing values, $R_i = R_j$ if and only if $T_i = T_j$. If indeed $R_i = R_j$ and $T_i = T_j$, then the condition for the final assignment is equivalent to the inductive hypothesis. If computers i and j come from the same part, then the ordering choice between them was fixed, and the assignment of T values if needed is clearly minimal. So consider further the case in which computer i comes from a different part than computer j, and their R and T values are different. We have two cases: either computer i was in part R, or in part T.

If computer i was in part R, then its assigned T value is by definition the largest among all computers, and it's the smallest possible for it to go after computer j, whose value is minimal by the inductive hypothesis. As for the order, $R_i \leq \mathbf{C}-1$ per the limits. Since computer j comes from part T and was chosen for position $\mathbf{C}-1$ (when N was $\mathbf{C} - 2$), that means $R_i > \mathbf{C} - 2$. Therefore, $R_i=\mathbf{C}-1$, and the chosen position is correct.

If, on the other hand, computer i was in part T, then its T value is minimal because $T_i$ is fixed. As for the order, notice that all computers have either a T value strictly less than $T_i$ or an R value strictly less than $\mathbf{C}-1$, so none of them could have been last. By the inductive hypothesis, $T_j$ is minimal among all possible orders, which means, by the existence of a full assignment, it has to be $T_j < T_i$, which implies the consistency of the final order and value assignment.