Coding Competitions Farewell Rounds - Round B

Analysis: Railroad Management

For this problem, let's define R_i as the number of railroad cars that the station i has already received from other stations. At the beginning, $R_i = 0$ for all i. Then, each time we make a shipment from station i to station $\mathbf{D_i}$, $R_{\mathbf{D_i}}$ increases by $\mathbf{C_i}$.

Test Set 1

The shipment order can be defined as a permutation of the network stations. In Test Set 1, the limits are small enough that we can try every possible order and output the smallest answer among all of them.

For a given order, we process the shipments one by one. For the station i, let $cost(i)$ be the number of additional railroad cars required to complete the shipment. Since we can reuse the received cars to reduce the answer, $cost(i) = \max(\mathbf{C_i} - R_i, 0)$. We can then add $cost(i)$ to the answer, and also increase $R_{\mathbf{D_i}}$ by $\mathbf{C_i}$.

The answer for a given shipment order can be calculated in $O(\mathbf{N})$ and there are $\mathbf{N}!$ different orders. Therefore, a test case can be solved in $O(\mathbf{N}! \times \mathbf{N})$.

Test Set 2

Let's create a weighted directed graph with $\mathbf{N}$ vertices (one for each station) and $\mathbf{N}$ edges (one for each shipment). In other words, for each vertex i there will be exactly one outgoing edge, from i to $\mathbf{D_i}$, with weight $\mathbf{C_i}$.

Note that for each edge e from u to v, if e is not part of a cycle, then it is optimal to complete the shipment of u before the shipment of v. The reasoning behind this is that v can reuse railroad cars sent by u, possibly reducing the answer, while it is impossible to do the opposite: no car can ever travel from v to u. Therefore, we can apply the following process:

1. Pick any vertex u with no incoming edges
2. Increase the answer by $cost(u)$ and $R_{\mathbf{D_u}}$ by $\mathbf{C_u}$
3. Erase the vertex u and its outgoing edge from the graph
4. Repeat while there are vertices with no incoming edges.

Now, since each remaining vertex has at least one incoming edge and exactly one outgoing edge, the graph is now a collection of disjoint simple cycles. Thus, we can add the cost of each cycle separately.

One way to calculate the cost of a cycle is to traverse it for each possible starting point, completing the shipments in order, and then use the smallest answer among all traversals. However, the time complexity of this is $O(\mathbf{N}^2)$, which is not enough to pass Test Set 2.

We can optimize this to a linear solution with the following observation: suppose that the starting vertex is s. The cost of s is $cost(s) = \max(\mathbf{C_s} - R_s, 0)$. For any other vertex u which is not the starting one, its cost is $cost\_in\_cycle(u) = \max(\mathbf{C_u} - R_u - \mathbf{C_{pred(u)}}, 0)$, where $pred(u)$ is the predecessor of u in the cycle (i.e. $\mathbf{D_{pred(u)}} = u$). Note that this is independent of s, hence we can precalculate sum_of_costs as the sum of $cost\_in\_cycle(u)$ for all vertices u in the cycle and compute the cost for each starting vertex s as $sum\_of\_costs - cost\_in\_cycle(s) + cost(s)$, in constant time.