Problem 1

Let us partition the set of nodes $V$ into disjoint subsets according to their distance from the node $s$: $V = D_0 \cup D_1 \cup D_2 \cup \ldots$, where $D_i$ consists of nodes the shortest distance to which from $s$ equals to $i$. Trivially, $D_0 = \{s\}$. Let $D_m$ be the subset containing the node $t$. By the choice of $s$ and $t$ in the graph, $m > \frac{n}{2}$. Consequently, among $D_1, D_2, \ldots, D_{m-1}$ there exists $D_k$ consisting of a single node. Otherwise, if each of these subsets has at least two nodes, the total number of nodes in the graph could not be less than $2(m+1) + 2 = 2m > n$, which contradicts definition of $G$.

Consider the subset $D_k = \{v\}$. Since for any pair of consecutive nodes in a path, the distance from $s$ can increase at most by one, to leave the subset of nodes $D_0 \cup D_1 \cup \ldots \cup D_{k-1}$ a path from $s$ to $t$ must pass through $D_k$. Thus, removing $v$ from $G$ deletes all paths connecting $s$ and $t$.

In order to build the subsets $D_i$, we can use the breadth-first search algorithm, which discovers nodes in the order of non-decreasing distance from the starting node. Time complexity of BFS is $O(|V| + |E|)$.

Problem 2

As in the previous problem, let us consider the partition of the set of nodes $V$ into disjoint subsets $D_0 \cup D_1 \cup D_2 \cup \ldots = V$ according to their distance from the node $v$. For each node $u \in D_k$ we define $N_v(u)$ as the number of shortest paths from $v$ to $u$. By definition of the subsets $D_i$, any shortest path from $v$ to $u$ passes through a node belonging to $D_{k-1}$.

Consider a node $t \in D_{k-1}$ adjacent to $u$. Each of the $N_v(t)$ shortest paths from $v$ to $t$ (if there are any) can be extended with the edge $(t, u)$. Therefore,
the following recurrence holds.

\[ N_v(u) = \sum_{t \in D_{k-1}} N_v(t), \quad (1) \]

for any \( u \in D_k, \ k > 0 \). For \( v \) itself, we set \( N_v(v) = 1 \).

Again, partition \( D_0, D_1, D_2, \ldots \) naturally comes from the breadth-first search in \( O(|V| + |E|) \) time. For a node \( u \), calculating the value of sum (1) takes \( O(\text{deg } u) \) time. Since the sum of degrees of all nodes in the graph equals to \( 2|E| \), sum (1) can be computed for every node in total time of \( O(|E|) \). Therefore, the overall time complexity is \( O(|V| + |E|) \).

**Problem 3**

Let us build a directed graph \( G \) of possible virus expansion. Each node in this graph corresponds to a computer at a certain moment in time. The set of edges is constructed so that two nodes \((c_i, t_k)\) and \((c_j, t_l)\) are connected by a directed path iff a virus contaminated computer \( c_i \) at time \( t_k \) can infect computer \( c_j \) at time \( t_l \). Specifically, for each triple \((c_i, c_j, t_k)\) in trace data, there are two nodes \((c_i, t_k)\) and \((c_j, t_k)\) connected with each other in both directions by two edges \((c_i, t_k) \rightarrow (c_j, t_k)\) and \((c_j, t_k) \rightarrow (c_i, t_k)\). These edges implement the infection rule described in the problem statement. Additionally, all of the nodes \((c_i, t_{k_1}), (c_i, t_{k_2}), \ldots, (c_i, t_{k_m})\) corresponding to the same computer \( c_i \) are connected by edges in chronological order:

\[(c_i, t_{k_1}) \rightarrow (c_i, t_{k_2}) \rightarrow \cdots \rightarrow (c_i, t_{k_m})\],

where \( t_{k_1} < t_{k_2} < \cdots < t_{k_m} \). These edges are due to the fact that, once infected, a computer stays infected to the end of observing period.

Each directed path in \( G \) corresponds to a sequence of infections \((\cdots \rightarrow (c_i, t_k) \rightarrow (c_j, t_l) \rightarrow \cdots)\) and “dormancy” intervals \((\cdots \rightarrow (c_i, t_{k_1}) \rightarrow (c_i, t_{k_2}) \rightarrow \cdots, t_k < t_l)\) of a virus, which are the only two actions it can undertake according to the given model. By the construction of \( G \), the opposite is also true. If a computer \( c_a \) contaminated at time \( x \) can infect a computer \( c_b \) by time \( y \), there is a directed path in \( G \) from \((c_a, t_1)\) to \((c_b, t_2)\), where \( t_1 \geq x \) and \( t_2 \leq y \). Therefore, the problem reduces to locating these nodes in the graph, and testing whether there is a path connecting them.
The complete algorithm for solving this problem consists of three steps: building the graph $G$, locating nodes $(c_a, t_1), (c_b, t_2)$ by given $c_a, x, c_b, y$, and finding a path connecting those nodes. Since $|V(G)| \leq 2m$ and $|E(G)| \leq 3m$, where $m$ is the number of contacts $(c_i, c_j, t_k)$ in trace data, building the graph takes $O(m)$ time. Locating nodes can be trivially done in $O(|V(G)|) \in O(m)$ time. Connectivity between two nodes in $G$ can be tested using the depth-first or the breadth-first search algorithms, both having $O(|V(G)|+|E(G)|) \in O(m)$ time complexity, which is also the time complexity of the overall algorithm.