For problem **Many**, we solve each instance G by first classifying the graph**.**
If the graph is a DAG**,** we compute the maximum number of red vertices along an s, t-path in polynomial time. If the graph is undirected and acyclic (a tree)**,** we also compute the solution efficiently using a modified BFS.
However, if the graph contains any kind of cycle (directed or undirected), or if edges are mixed (→ and --)**,** then the problem becomes intractable, and our program returns **"NP-HARD"**, indicating that we cannot solve it in polynomial time.

In the DAG case, we perform a Dynamic Programming traversal over a topological ordering**.** For every vertex, we store the maximum number of red vertices collected so far on a path from s.
When processing neighbours, the red count is updated and propagated only if it improves the best value stored for that node. This ensures efficiency by avoiding unnecessary revisits.
In the undirected acyclic case (tree), we run a Breadth-First Search (BFS) starting from s, carrying forward the cumulative number of red nodes.
Since trees have a unique simple path between any two vertices, the first successful reach of t gives the optimal red count.

Building the graph and detecting DAG/tree structure both require O(V + E) time.
The DP traversal on DAGs and BFS on trees also run in O(V + E)**,** as each edge and vertex is processed at most once.

Claim: Many is NP-hard.

We prove NP-hardness by reducing from the **Longest Path** problem.
Let G = (V, E) be a directed graph and k a parameter for the Longest Path instance.
We construct an equivalent instance of **Many** as follows:

1. Use the same vertices V(G) and edges E(G).
2. Mark every vertex red.
3. Choose the same s and t as in the Longest Path instance.

Now, the Many problem asks for a path from s to t that maximizes the number of red vertices. Because all vertices are red, the number of red vertices in any path is exactly equal to its length + 1. Therefore, finding the maximum-red s, t-path in Many is identical to finding the longest path in G.

If we can solve Many in polynomial time, we could solve Longest Path as well, which is known to be NP-hard. Therefore, Many is NP-hard.

Example:

Let $V=\{s, A, B, C, t\}$ and $E=\{(s, A),(A, B),(A, C),(C, t)\}$
With all vertices red, the path $s \rightarrow A \rightarrow C \rightarrow t$ contains three red nodes, which is optimal.
This corresponds to a longest path of length two edges in the original graph.

Conclusion:

Since we reduced Longest Path to Many in polynomial time, and their solutions correspond directly, **Many is NP-hard.**