Task 1: Time Complexity Analysis

Let $G = (V, E)$ be a directed graph with $n = |V|$ vertices. We aim to find, for each vertex u, the smallest labeled vertex v reachable from u.

Method 1: Modified DFS from Each Vertex

Adjacency List Representation

- **Time to perform DFS from a single vertex u:**
  - The DFS explores all vertices reachable from u. In the worst case, this could be all n vertices.
  - The time to explore each vertex is proportional to its out-degree.
  - Total time for one DFS: $O(E_u)$, where E_u is the number of edges explored starting from u.
- **Total Time Complexity:**
  - Sum over all vertices: $T(n) = \sum_{u=1}^{n} O(E_u)$.
  - Since $\sum_{u=1}^{n} E_u = |E|$, the total time is $T(n) = O(n + |E|)$.
  - Performing DFS from each vertex, total time becomes $T(n) = n \cdot O(n + |E|) = O(n^2 + n|E|)$.

Adjacency Matrix Representation

- **Time to perform DFS from a single vertex u:**
  - In an adjacency matrix, checking for adjacent vertices involves scanning an entire row of the matrix.
  - For each vertex, this requires $O(n)$ time to identify all neighbors.
  - During DFS, each vertex may lead to scanning all n possible edges, resulting in $O(n)$ operations per vertex visited.
  - Therefore, the total time for one DFS is $O(n^2)$, as each of the n vertices reachable from u could involve scanning n entries in the adjacency matrix.
- **Total Time Complexity:**
  - Since we perform DFS from each of the n vertices, the total time becomes $T(n) = n \cdot O(n^2) = O(n^3)$.

Method 2: Transpose Graph and Modified DFS

Adjacency List Representation

- **Time to construct G^T:**
  - For each edge (u, v) in G, add an edge (v, u) in G^T.
  - Time: $O(|E| + n)$.
- **Time for DFS Traversals:**
  - Each edge and vertex is explored at most once during all DFS traversals.
  - Total time: $O(|E| + n)$.
- **Total Time Complexity:**
  - $T(n) = O(2|E| + 2n)$.

Adjacency Matrix Representation

- **Time to construct G^T:**
  - Transposing the adjacency matrix involves swapping rows and columns.
  - Time: $O(n^2)$.
- **Time for DFS Traversals:**
  - For each vertex, scanning the adjacency matrix row to find neighbors takes $O(n)$ time.
  - Since each edge is processed once, the total time for DFS traversals is $O(n^2)$.
- **Total Time Complexity:**
  - $T(n) = O(2n^2)$.

Conclusion

After analyzing both methods under different graph representations, we observe the following:

- **Method 1: Modified DFS from Each Vertex**
  - **Adjacency List:** $O(n^2 + n|E|)$
  - **Adjacency Matrix:** $O(n^3)$
- **Method 2: Transpose Graph and Modified DFS**
  - **Adjacency List:** $O(|E| + n)$

- **Adjacency Matrix:** $O(n^2)$

Best Algorithm: Method 2 using an adjacency list representation is the most efficient approach, with a time complexity of $O(|E| + n)$. This method outperforms Method 1, especially for sparse graphs where $|E|$ is much less than n^2. Additionally, it avoids the higher computational costs associated with using an adjacency matrix.

Task 2: Algorithms

Method 1: Modified DFS from Each Vertex (Adjacency List)

Algorithm 1 FindSmallestReachableVertices(G)

```
for each vertex u in V do
    visited ← ∅
    min_label ← u.label
    DFS(u, visited, min_label)
    result[u] ← min_label
end for
DFS(u, visited, min_label)
visited.add(u)
if u.label < min_label then
    min_label ← u.label
end if
for each neighbor v of u do
    if v ∉ visited then
        DFS(v, visited, min_label)
    end if
end for
```

Method 2: Transpose Graph and Modified DFS (Adjacency List)

Algorithm 2 FindSmallestReachableVerticesTransposed(G)

$G_T \leftarrow$ TransposeGraph(G)
$marked \leftarrow \emptyset$
$result \leftarrow$ empty map
for each vertex v in V in increasing order **do**
    if $v \notin marked$ **then**
        $visited \leftarrow \emptyset$
        DFS(G_T, v, $visited$)
        for each u in $visited$ **do**
            $result[u] \leftarrow v$
        end for
        $marked \leftarrow marked \cup visited$
    end if
end for

DFS(G_T, u, $visited$)
$visited$.add(u)
for each neighbor v of u in G_T **do**
    if $v \notin visited$ **then**
        DFS(G_T, v, $visited$)
    end if
end for

TransposeGraph(G)
$G_T \leftarrow$ new graph
for each vertex u in G **do**
    for each neighbor v of u **do**
        G_T.addEdge(v, u)
    end for
end for
return G_T

Task 3: Description of Significant Edge Cases Tested

To ensure the robustness and correctness of the algorithm, several significant edge cases were tested. These edge cases cover a wide range of scenarios that the algorithm might encounter in practical applications.

Edge Cases

- **Disconnected Graph:**

  - A graph where some vertices have no edges connecting them to other vertices.
  - Ensures that the algorithm correctly identifies each vertex as its own smallest reachable vertex when there are no connections.

- **Graph with Self-Loops:**
  - Vertices that have edges pointing to themselves.
  - Tests the algorithm's ability to handle cycles of length one without getting stuck in infinite loops.

- **Complete Graph:**
  - Every vertex has an edge to every other vertex.
  - Ensures that the algorithm efficiently handles graphs with the maximum number of edges and correctly identifies the smallest vertex reachable from any vertex.

- **Graph with Cycles:**
  - Graphs that contain cycles of length greater than one.
  - Tests the algorithm's ability to navigate through cycles without redundant processing and correctly determine the smallest reachable vertex.

- **Single Vertex:**
  - The simplest possible graph containing only one vertex and no edges.
  - Ensures that the algorithm can handle minimal input gracefully.