IV003 Set 2, problem 3 10 points **due to 16. 4. 2025**

Name: Andrii Bezrukov **UČO:** 570943

0007 *list* 1 *učo* 570943 *body*

0 1 2 3 4 5 6 7 8 9

Consider the UNION FIND data structure. Let S be a sequence of m MAKESET operations, followed by a sequence of u UNION operations, followed by a sequence of f FIND operations. Let $n = m+u+f$. Prove that starting from an empty data structure, link-by-rank with path compression performs the given sequence S of n operations in $\mathcal{O}(n)$ time. You can assume that all UNION operations have a tree root as an argument, and thus their complexity is constant.

Problem 3: Union-Find Time Complexity

1 Main Idea

The key insight is to leverage the specific sequence structure (all MakeSet operations, followed by all Union operations, then all Find operations) and the properties of path compression to show that the total time complexity is linear in the number of operations.

2 Given Sequence

We have a sequence S consisting of:

- m MakeSet operations
- followed by u Union operations (with tree roots as arguments)
- followed by f Find operations
- where $n = m + u + f$ is the total number of operations

3 Proof of Complexity

Base Analysis:

- **MakeSet operations**: Each takes $O(1)$ time, for a total of $O(m)$ time.
- **Union operations**: Since all Union operations use tree roots as arguments (as stated in the problem), each takes $O(1)$ time, for a total of $O(u)$ time.
- **Find operations**: Without considering path compression, each Find could take up to $O(\log m)$ time in a link-by-rank implementation.

Path Compression Effect: The crucial part is that path compression modifies the tree structure during Find operations:

1. After the m MakeSet and u Union operations, we have a fixed forest structure where each tree has height at most $O(\log m)$ due to link-by-rank.
2. For the sequence of f Find operations:
   - The first Find on any path might traverse up to $O(\log m)$ nodes.
   - However, path compression flattens this path, making all nodes on it point directly to the root.

- Subsequent Find operations involving any of these nodes will take $O(1)$ time.

3. Each node can contribute to an "expensive" Find operation at most once - the first time it's encountered in a Find operation. After that, its path to the root becomes of length 1.
4. Therefore, across all f Find operations, the total number of "expensive" steps is bounded by the total number of nodes, which is m.

Total Time Analysis: Let T be the total time for all operations:

- Time for MakeSet operations: $O(m)$
- Time for Union operations: $O(u)$
- Time for Find operations: $O(m) + O(f)$
  - $O(m)$ accounts for the one-time cost each node might incur in its first Find
  - $O(f)$ accounts for the constant-time operations

Therefore: $T = O(m) + O(u) + O(m + f) = O(m + u + f) = O(n)$

4 Running Time

$O(n)$

5 Justification

The key to achieving linear time complexity is the specific sequence structure where:

1. All structure-building operations (MakeSet and Union) occur before any Find operations
2. Path compression optimizes the structure during the first Find operation on each path
3. Each node can contribute to at most one expensive Find operation
4. All subsequent Find operations on previously compressed paths take constant time

In practice, the overall time complexity is $O(n \cdot \alpha(n))$, which for all practical purposes can be considered almost linear—effectively $O(n)$ because $\alpha(n)$ is a very slowly growing function.