COLORADO MESA UNIVERSITY

Group Code Task 1 Due: 23:00 Sep 22nd, 2024 20 Points

CSCI 250 Introduction to Algorithms

Objectives

This group coding assessment task thoroughly evaluates your understanding of union-find, quicksort with path compression, algorithm analysis, and basic data structures like stacks and queues implemented using self-resizing arrays. It also encourages additional creativity and offers bonus points for implementing optional features.

Task Description

Part 1: Union-Find Data Structure

Implement a Union-Find data structure in C++ that supports the following operations:

- `Union(p, q)`: Merge two components represented by elements `p` and `q`.
- `Find(p)`: Find the component identifier of element `p`.
- `Connected(p, q)`: Check if elements `p` and `q` are in the same component.

Ensure to use quick union with path compression to optimize the Union and Find operations.

For a known (you provide a sequential list of at least 11 nodes and `Union(p, q)` sequence) set of nodes provide a graph showing the connected final component with path compression.

Part 2: Problem Solving

Utilize your Union-Find data structure to solve the following problem:

You are given a set of `N` elements, initially all in separate components. You will receive a sequence of `M` operations, where each operation is either of the following two types:

- `U a b`: Union operation - Merge the components containing elements `a` and `b`.
- `Q a`: Query operation - Find the size of the component containing element `a`.

Implement a function to process these operations efficiently using your Union-Find data structure.

Part 3: Algorithm Analysis

Analyze the time complexity of your solution in terms of `N` (the number of elements) and `M` (the number of operations). Discuss the efficiency of your algorithm and any potential optimizations.

Part 4: Additional Task

Implement the following data structures / features for 5 points:

- Implement a stack and a queue data structure using self-resizing arrays.
- Demonstrate how you can use these stack and queue data structures to solve simple problems such as:

  Reversing a non-trivial string using a stack.

  Simulating a queue using two stacks.

Deliverables:

- C++ code implementing the Union-Find data structure and solving the problem.
- Classes should be coded using interface and implementations in separate files i.e., ".h" and ".cpp". Templated classes should be prototyped and implemented in their interface ".h" files.
- A LaTeX based PDF document explaining your algorithm's time complexity analysis and any optional features you implemented.

Blacklist:

- C++ code using STL constructs or libraries.
- Any data structure not of your making.