

DISJOINT-SET

A disjoint-set is a data structure for maintaining dynamic partitions of a set.

I. Language Agnostic Overview

A partition of some set X is a collection of non-empty subsets whose union is X , but any two of the subsets are disjoint. For example, let $X = \{1,2,3,4,5\}$. Then, $\{\{1,3,5\},\{2,4\}\}$ is a partition of X .

The source code for the disjoint-set class is taken from the book “Data Structure and Algorithm Analysis in Java” by Mark Allen Weiss, first edition, 1998.

There are two operations that must be supported by a disjoint set implementation:

- Find – This takes an element from the set X and returns the subset that contains the element. In the above example, Find(3) would return the subset $\{1,3,5\}$. This algorithm is also commonly referred to as a “union-find” algorithm.
- Union – This takes two different subsets and merges them into one subset.

Any disjoint-set implementation must satisfy the following properties:

- Tree nodes must be the elements of the set X .
- Every node (except the root) must have a pointer to the parent.
- Every tree forms a set, and the root of a tree is representative of the entire tree.

The find operation takes a tree node as input and returns the root of the tree which contains the node. It does this by following the parent pointers until a root node is reached. Thus, the time complexity of the find operation is proportional to the maximum height of the trees in the forest, and the complexity is linear when the forest degenerates into a list. Conversely, the union operation takes the roots of two trees and merges them into one tree. The time complexity of the union operation is constant since the operation simply sets one root's parent pointer to be the other input root node.

Further, an optimized union-find algorithm incorporates two improvements over the original algorithm described above:

- Union by rank – In the union operation, the root of the shorter tree is set as a child of the root of the higher tree. This guarantees that the maximal tree height is increased at most by one after the union operation.
- Path compression – After the find operation, the input element is set as a child of the return value (The root of the tree which contains the input element).

II. Java Specific Overview

The disjoint set tree is implemented on top of an integer array, which makes the forest have the following properties:

- The array indexes are the elements of the set.
- The value of an array index i ($s[i]$) is the parent of index i .
- A root is denoted by having a value of -1, that is, node r is a root node if and only if $s[r] = -1$.

There are two implementations of the disjoint-set class included, one optimized and one unoptimized. They are both subject to the same invariants, however, and so only the invariants and dependent methods are discussed here.

The disjoint-set invariant has three requirements:

- The underlying array must not be null.
 - Enforced by verifying that $s \neq \text{null}$.
- Every value must be between -1 and one less than the value of the array.
 - Enforced by verifying that the return value from the method `goodValues` is `boolean true`.
- The data structure must be acyclic, that is, it must be a forest.
 - Enforced by verifying that the return value from the method `acyclic` is `boolean true`.

The pre and postconditions for the Find operation, `Find(int x)`, are:

- Precondition: $x \geq 0 \ \&\& \ x < s.length$
 - This asserts that the input element is in the set.
- Postcondition: $s[\$ret] < 0$
 - This verifies that the returned value is a root of a tree. Note that the postcondition is rather weak. Ideally, we want to assert that the input element is in the tree that has the return as its root.

The pre and postconditions for the Union operation, `union(int root1, int root2)`, are:

- Precondition: $root1 \geq 0 \ \&\& \ root1 < s.length \ \&\& \ root2 \geq 0 \ \&\& \ root2 < s.length \ \&\& \ root1 \neq root2 \ \&\& \ s[root1] == -1 \ \&\& \ s[root2] == -1$
 - This checks that the two input elements are roots of different trees.
- Postcondition: $find(root1) == find(root2)$
 - This makes sure that the two trees are merged together.

Metrics:

Original:

Lines of code for implementation: 35
 Methods for implementation: 5
 Lines of code for invariants: 39
 Methods for invariants: 3

Fast:

Lines of code for implementation: 38
 Methods for implementation: 4
 Lines of code for invariants: 39
 Methods for invariants: 3

References:

Wikipedia's entry on disjoint sets: http://en.wikipedia.org/wiki/Disjoint-set_data_structure

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 21: Data Structures for Disjoint Sets, pp.498-524