Union-Find Problem

Suppose we have a collection of n sets each containing some elements. Further, we suppose that the sets can be "named" using the integers 1, 2, 3, ... , n. Finally, we suppose each of these elements can be represented by an integer in the range 1 to n (inclusive). We want to implement two operations:

FIND(i) – gives the name of the set containing element **i**.

UNION(I,J,K) – Forms the set union of sets **I** and **J** and names the result **K**.

A Solution to the Union-Find Problem

The solution uses a naming scheme that uses both "**internal names**" and "**external names**" for identifying the sets. The operations FIND and UNION listed above use external names for their parameters and return values.

The Cast of Characters – Making sense of Internal and External Set Names

R[i] indicates the internal name of the set containing element **i**.

LIST[A] is a pointer to a link list of elements contained in set **A**. Note that **A** is the internal name of some set. A list cell contains the data fields `element` and `next`.

SIZE[A] is the number of elements in set **A**. Note that **A** is the internal name of some set.

EXTERNAL_NAME[A] is the external name of the set whose internal name is **A**.

INTERNAL_NAME[I] is the internal name of the set whose external name is **I**.

Implementing FIND()

```
int FIND( i )
{
    A = R[i] ;
    K = EXTERNAL_NAME[A] ;
    return K ;
}
```

Implementing UNION()

```
int UNION( I, J, K )
{
    A = INTERNAL_NAME[I]
    B = INTERNAL_NAME[J]
    // Without loss of generality, assume SIZE[A] ≤ SIZE[B]
    // Otherwise, interchange the roles of A and B
    head = LIST[A] // Pointer to the linked list representing A
    p = head ;
    while ( p != NULL ) do {
        R[ p->element ] = B
        last_p = p
        p = p->next
    }
    // Notice for Kruskal's algorithm, the sets are never empty.
    // Therefore, (in this version) there is no special case
    // checking if (head == NULL). The variable last_p is
    // always has a value at this point. We link the end of
    // list A to the beginning of list B.
    last_p->next = LIST[ B ]
    LIST[ B ] = head
    SIZE[ B ] = SIZE[A] + SIZE[ B ]
    INTERNAL_NAME[ K ] = B
    EXTERNAL_NAME[ B ] = K
}
```

Analysis:

The time for a **find()** operation is $\mathcal{O}(1)$.

In the context of Kruskal's algorithm, we begin with n sets, each of which contains only one vertex. There are exactly n vertices in the graph.

As the edge queue is processed, $n-1$ **union()** operations are performed. Each time a **union()** operations is performed, a number of elements are moved from one list onto a different list. By merging the smaller list onto the larger one, each moved element finds itself on a list at least twice as long. The number of times any list can double in size is limited to $\mathcal{O}(\log n)$, since there are only n elements.

There are $\mathcal{O}(n)$ such **union()** operations; therefore, the number of times any element is moved is bounded by $\mathcal{O}(n \log n)$.