v.6.8

Literate Programming Illustration

In this article, I illustrate true enhanced literate programming where code chunks can be both displayed and evaluated. In this document, I will demonstrate the implementation of a union find algorithm, and the use of the algorithm directly in this document.

Here we define a union find structure:

```
(struct uf-node (parent rank) #:mutable)

(define (make-uf-node)
  (define n (uf-node #f 0))
  (set-uf-node-parent! n n)
  n)
```

Here's the `find-set` function:

```
(define (uf-find node)
  (cond
    [(equal? node (uf-node-parent node)) node]
    [else
     (set-uf-node-parent! node (uf-find (uf-node-parent node)))
     (uf-node-parent node)]))
```

And the `union` function:

```
(define (uf-union x y)
  (let ([x (uf-find x)]
        [y (uf-find y)])
    (cond
      [(equal? x y) #f]
      [(> (uf-node-rank x) (uf-node-rank y)) (set-uf-node-parent! y x) #t]
      [(< (uf-node-rank x) (uf-node-rank y)) (set-uf-node-parent! x y) #t]
      [else
       (set-uf-node-parent! x y)
```

```
        (set-uf-node-rank! y (+ 1 (uf-node-rank y)))
        #t])))
```

Now let's create two sets:

```
(define A (make-uf-node))
(define B (make-uf-node))
```

Now let's union the two sets. And result should be `#t`, since they were successfully unioned together.

```
(uf-union A B)
```

⇒ `#t`

Let's test to make sure they are in the same set. The set representative of `A` and `B` should be the same. So this code should return `#t`:

```
(equal? (uf-find A) (uf-find B))
```

⇒ `#t`