CS 331 — Up Tree Activity

Mattox Beckman

Objective

Show how to use an up-tree to solve a practical problem.

Code Samples

```
(defprotocol UpTreeP
  (get-parent [this])
  (set-parent! [this p]))

(deftype UpTree [data
                 ^:volatile-mutable parent]
  UpTreeP
  (get-parent [this] parent)
  (set-parent! [this p] (set! parent p)))

(def u1 (UpTree. 4 nil))
(set-parent! u1 u1)

(defn up-tree
  "Create an uptree with an optional parent."
  ([data]
   (let [u (UpTree. data nil)]
     (set-parent! u u)
     u))
  ([data parent]
   (UpTree. data parent)))

(defn find! [node]
  (if (identical? (get-parent node) node) node
    (let [rep (find (get-parent node))]
       (set-parent! node rep)
       rep)))

(defn union! [a b]
  (set-parent! (find a) (find b)))
```

Question

Suppose we have a network of computers. Each computer is represented by a Clojure keyword. (e.g., "`:x`"). We have a list of pairs such as `[[:x :y] [:x :n] [:a :b]]` that represent the connections in our network.

We want, in $\mathcal{O}(n \log_* n)$ time, a function that will tell us the partitions that exists and their sizes. The partition can be given by using the up-tree representative.

```
user> (get-partitions [[:a :b] [:a :c] [:x :y]])

{:a 3, :y 1}
```