CSE 5311 Notes 8: Disjoint Sets

(Last updated 6/28/14 3:49 PM)

CLRS, Chapter 21

(CSE 2320 Notes 1 and Sedgewick's *Algorithms in C/C++/Java* cover at a high level)

Problem: For an equivalence relation (e.g. the partition of a set into *equivalence classes*):

1. Determine if two elements are equivalent (FIND), and

2. Allows merging (UNION) of equivalence classes.

Naive implementation - indicate subset for each element

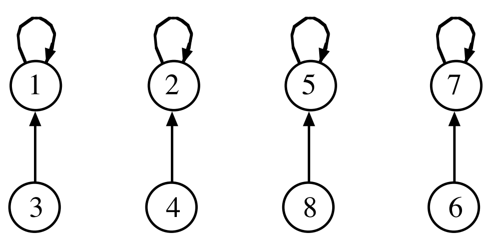
1	2	3	4	5	6	7	8
1	2	1	2	3	4	4	3

Represents equivalence relation:

$\{1,3\}$ $\{2,4\}$ $\{5,8\}$ $\{6,7\}$

UNION takes O(n) time - can do much better!!!!

Galler-Fischer Representation - Use trees (in an array) with just parent pointers



1	2	3	4	5	6	7	8
1	2	1	2	5	7	7	5

Trade-off:

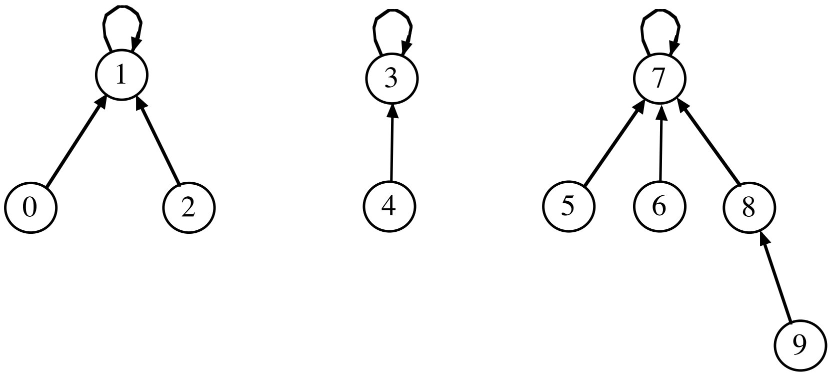
Increase in time to check equivalence (FIND)

vs.

Simplicity in merging (UNION) - redirect one root to another, then apply heuristics to reduce depth

UNION-BY-WEIGHT (size)

Keep subtree size in root (or separate array). If integer tables, then negative value for pointer indicates that the root's size (negated) is stored.



0	1	2	3	4	5	6	7	8	9
1	-3	1	-2	3	7	7	-5	7	8

Theorem: For any node x with height $h(T_x)$ in union-by-weight and size $s(T_x)$, $2^{h(T_x)} \le s(T_x)$.

Proof: By induction

Single node (as initialized):

$s(T_x) = 1$ and $h(T_x) = 0$

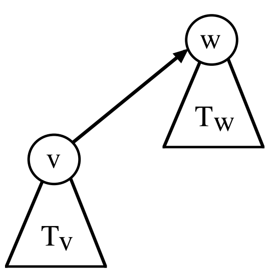
So, $2^{h(T_x)} = 2^0 \le s(T_x)$

Property holds before union and holds afterwards:

Suppose T_v and T_w are to be unioned. WOLOG, $s(T_v) \le s(T_w)$.

$2^{h(T_v)} \le s(T_v)$ and $2^{h(T_w)} \le s(T_w)$

T =



Show that $2^{h(T)} \le s(T)$:

$$2^{h(T)} = 2^{\max(1+h(T_v),h(T_w))} = \max\left(2^{1+h(T_v)}, 2^{h(T_w)}\right)$$

$$\le \max(2s(T_v), s(T_w)) \qquad 2^{h(T_v)} \le s(T_v) \text{ and } 2^{h(T_w)} \le s(T_w)$$

$$\le \max(s(T), s(T_w)) \qquad s(T_v) \le s(T_w) \text{ and } s(T_v) + s(T_w) = s(T)$$

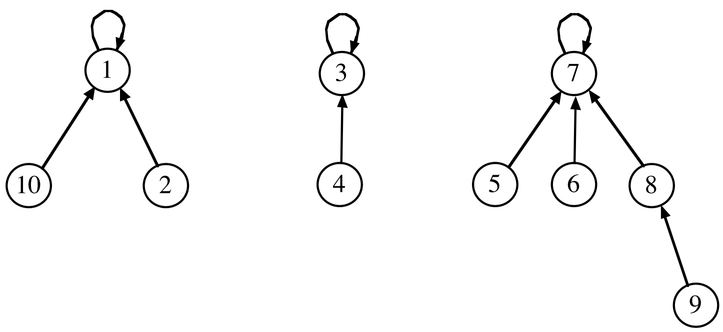
$$\le \max(s(T), s(T)) = s(T) \qquad s(T_w) \le s(T)$$

Corollary: $h(T) \le \log s(T)$

So, FINDs under union-by-weight take O(log n)

UNION-BY-RANK (height)

Keep subtree rank (height) in root (or separate array).



1	2	3	4	5	6	7	8	9	10
-1	1	-1	3	7	7	-2	7	8	1

Theorem: For any node x with rank $r(T_x)$ in union-by-rank and size $s(T_x)$, $2^{r(T_x)} \le s(T_x)$

Proof: Very similar to union-by-weight.

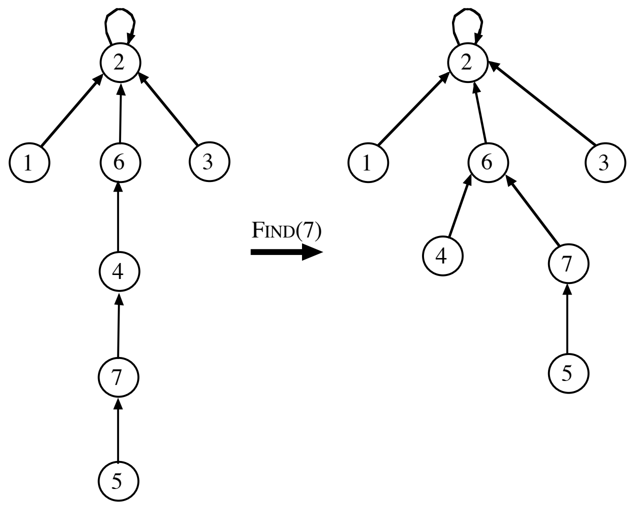
Corollary: $r(T) \le \log s(T)$

So, FINDs under union-by-rank take O(log n)

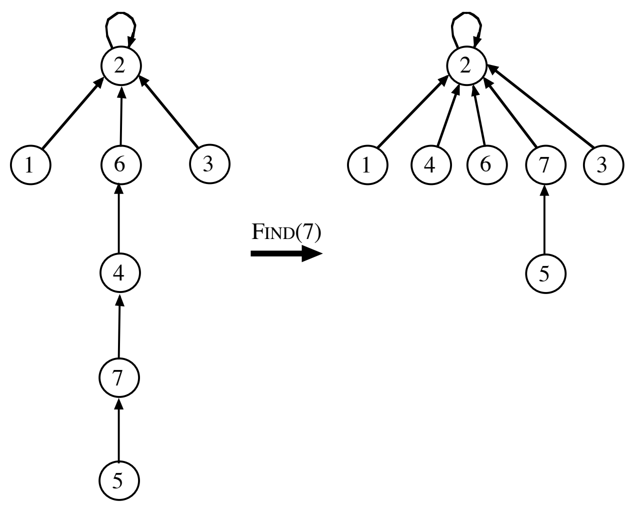
PATH COMPRESSION

Method 1: Use indirection while following the path for a FIND:

```
while (id[i]!=i)
  i=id[i]=id[id[i]];
return i;
```



Method 2 (CLRS): After a FIND reaches a tree's root, a second (backward) pass along the path makes every node point directly to the root.



Can easily combine with union-by-weight or union-by-rank.

Under union-by-rank, path compression causes each rank to be just an upper bound on the height.

In addition, the amortized cost of FIND and UNION will be *nearly* constant (inverse of extremely fast-growing function).

APPLICATIONS

1. Kruskal's Minimum Spanning Tree

    Sort edges in ascending order.

    Place each vertex in its own set.

    Process each edge {x, y} in sorted order:

    ```
    a=FIND(x)
    b=FIND(y)
    if a ≠ b
        UNION(a,b)
        Include {x, y} in MST
    ```

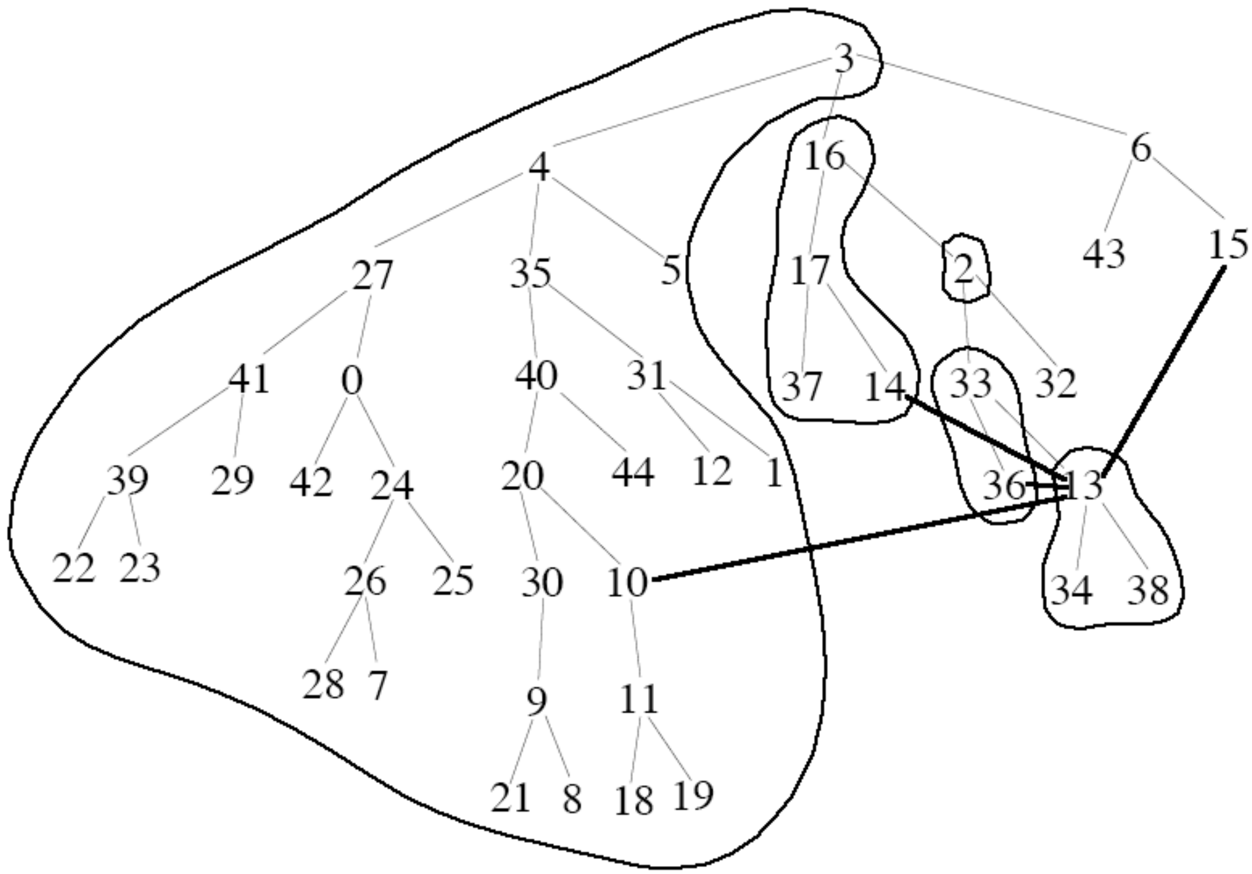
2. Many parallel algorithms use similar ideas. (See books by Ja Ja or Reif)

3. Connected components for undirected graphs.

4. First-order unification / logic programming / tree matching (*ACM Computing Surveys 21:1*, March 1989, fig. 4, http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=62029.62030)

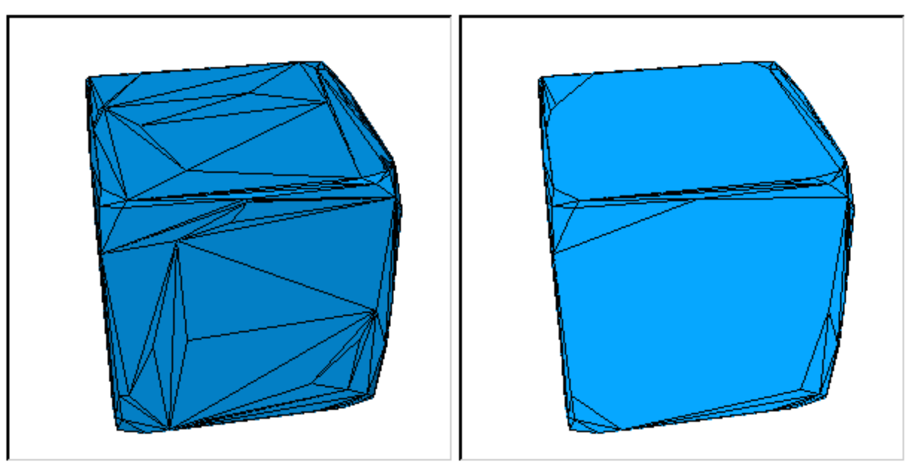
5. Off-line least common ancestors (CLRS, p. 521 http://ranger.uta.edu/~weems/NOTES5311/LCAoffline.c)

    - Union-find structure maintains subsets of nodes that have been processed.
    - Separate array ("ancestor") maintains the dominant node for each subset.
    - Processed by depth-first traversal (left to right)
        - When going down to a node, initialize its subset.
        - When going up to node X from node Y, union for X and Y subsets, make X dominant.
        - Before going up to node X from node Y, process any input query pair {Y, Z} where Z has already been processed completely.



(Aside: M.A. Bender and M. Farach-Colton, "The LCA Problem Revisited", May 2000, http://www.cs.sunysb.edu/~bender/newpub/BenderFa00-lca.pdf connects LCA to the range minimum query problem and cartesian trees.)

6. Find pairs of co-planar polygons sharing an edge in 3-d convex hull (Spring 2005 CSE 5392, http://ranger.uta.edu/~weems/NOTES5319/LAB2/dcel.html)



7. Maximum cardinality k-coloring of a set of intervals - a scheduling problem

M.C. Carlisle and E.L. Lloyd, "On the k-coloring of intervals", ***Discrete Applied Mathematics*** 59, 1995, 225-235,
http://www.sciencedirect.com.ezproxy.uta.edu/science/article/pii/0166218X9580003M

Sort intervals in ascending right-end ordering
Generate k + 1 dummy intervals (0 indicates "could not color")
Determine the (rightmost left-) adjacent interval for each interval
Initialize a union-find tree for each interval (simplified to linked lists here)
for each non-dummy interval i, according to the sorted order
    j = Number of the interval at the end of the list for the interval adjacent to i (FIND)
    if color of j is 0
        Color i with 0
        // If a later interval reaches i from its adjacent interval, then i - 1 might lead to a back-up
        Link i to i - 1 (FIND & UNION)
    else
        Color i with color of j
        // If a later interval reaches j from its adjacent interval, then j - 1 might lead to a back-up
        Link j to j - 1 (FIND & UNION)

3-coloring instance: