Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Union-Find Structures

Prof. Dianne Foreback

Merging galaxies, NGC 2207 and IC 2163. Combined image from NASA's Spitzer Space Telescope and Hubble Space Telescope. 2006. U.S. government image. NASA/JPL-Caltech/STSci/Vassar.

Reading Material

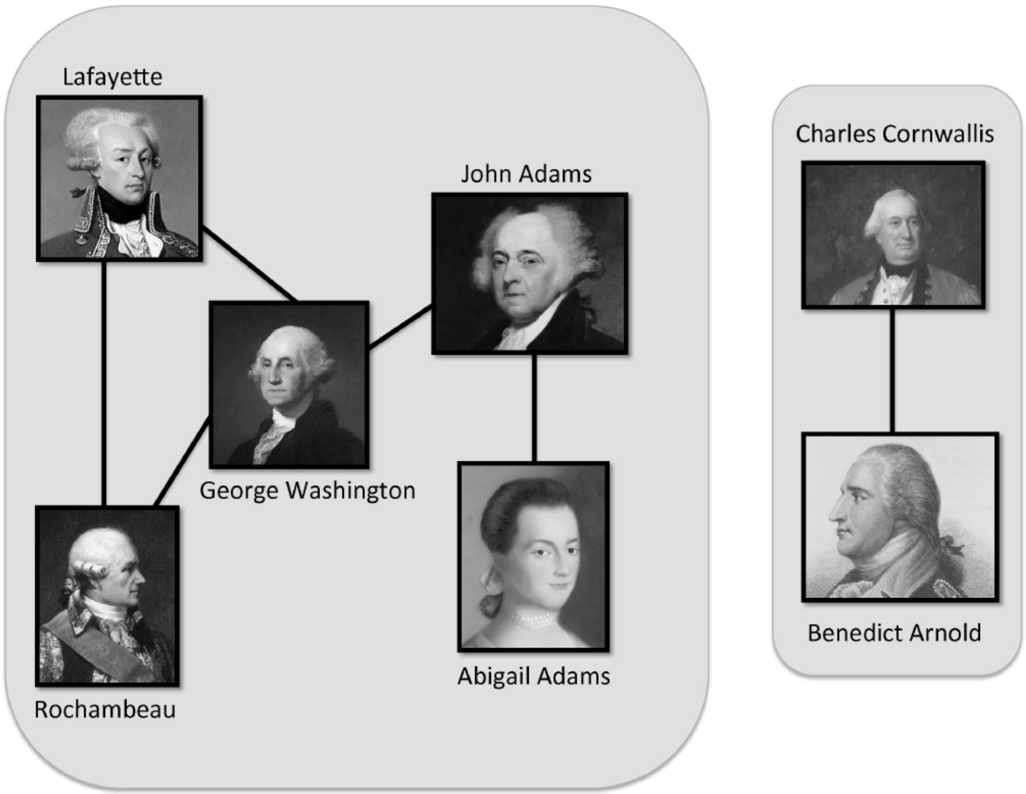
- *Algorithm Design & Applications* by Michael T. Goodrich and Roberto Tamassia
  - Chapter 7
    - Intro through and including Section 7.1.1
    - Section 7.2

Application: Connected Components in a Social Network

- Social networking research studies how relationships between various people can influence behavior.
- Given a set, S, of n people, we can define a social network for S by creating a set, E, of edges or ties between pairs of people that have a certain kind of relationship. For example, in a friendship network, like Facebook, ties would be defined by pairs of friends.
- A **connected component** in a friendship network is a subset, T, of people from S that satisfies the following:
  - Every person in T is related through friendship, that is, for any x and y in T, either x and y are friends or there is a chain of friendship, such as through a friend of a friend of a friend, that connects x and y.
  - No one in T is friends with anyone outside of T.

Example

- 2 Connected components in a friendship network of some of the key figures in the American Revolutionary War.



All images are in the public domain.

Union-Find Operations

- A **partition** or **union-find** structure is a data structure supporting a collection of disjoint sets subject to the following operations:
- makeSet(e): Create a singleton set containing the element e and return the position storing e in this set
- union(A,B): Return the set A U B, naming the result "A" or "B"
- find(e): Return the set containing the element e

Connected Components Algorithm

- The output from this algorithm is an identification, for each person x in S, of the connected component to which x belongs.

Algorithm UFConnectedComponents(S, E):

Input: A set, S, of n people and a set, E, of m pairs of people from S defining pairwise relationships

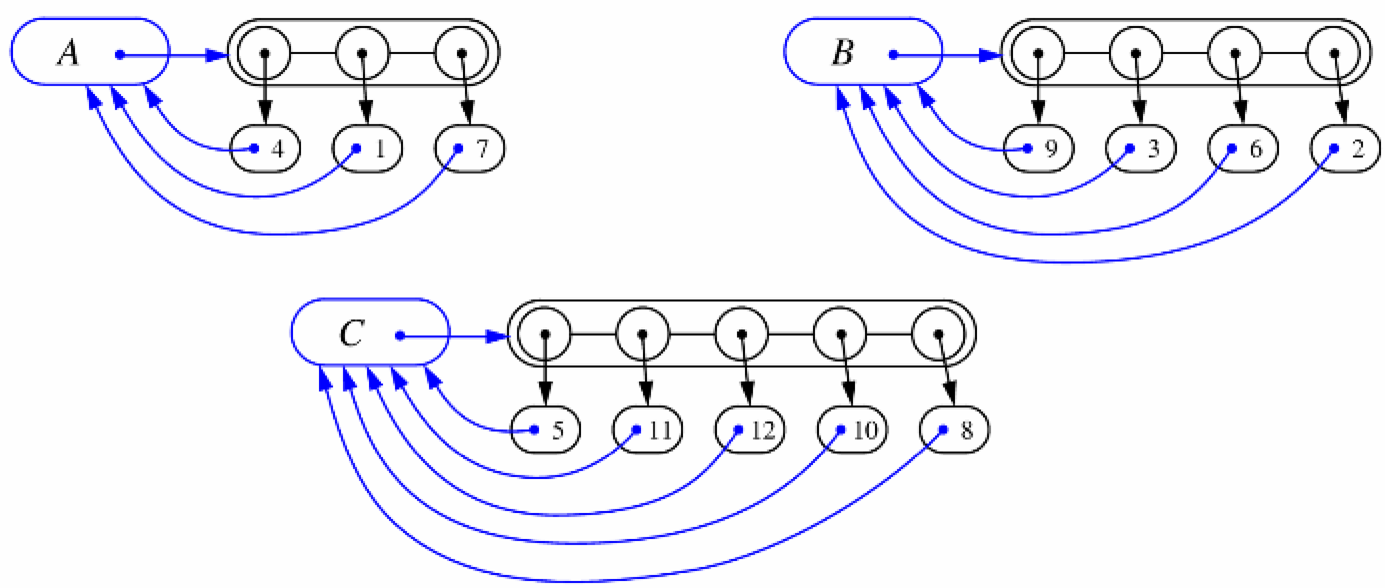
Output: An identification, for each x in S, of the connected component containing x

for each x in S **do**
    makeSet(x)
for each (x, y) in E **do**
    if find(x) $\neq$ find(y) **then**
        union(find(x), find(y))
for each x in S **do**
    Output "Person x belongs to connected component" find(x)

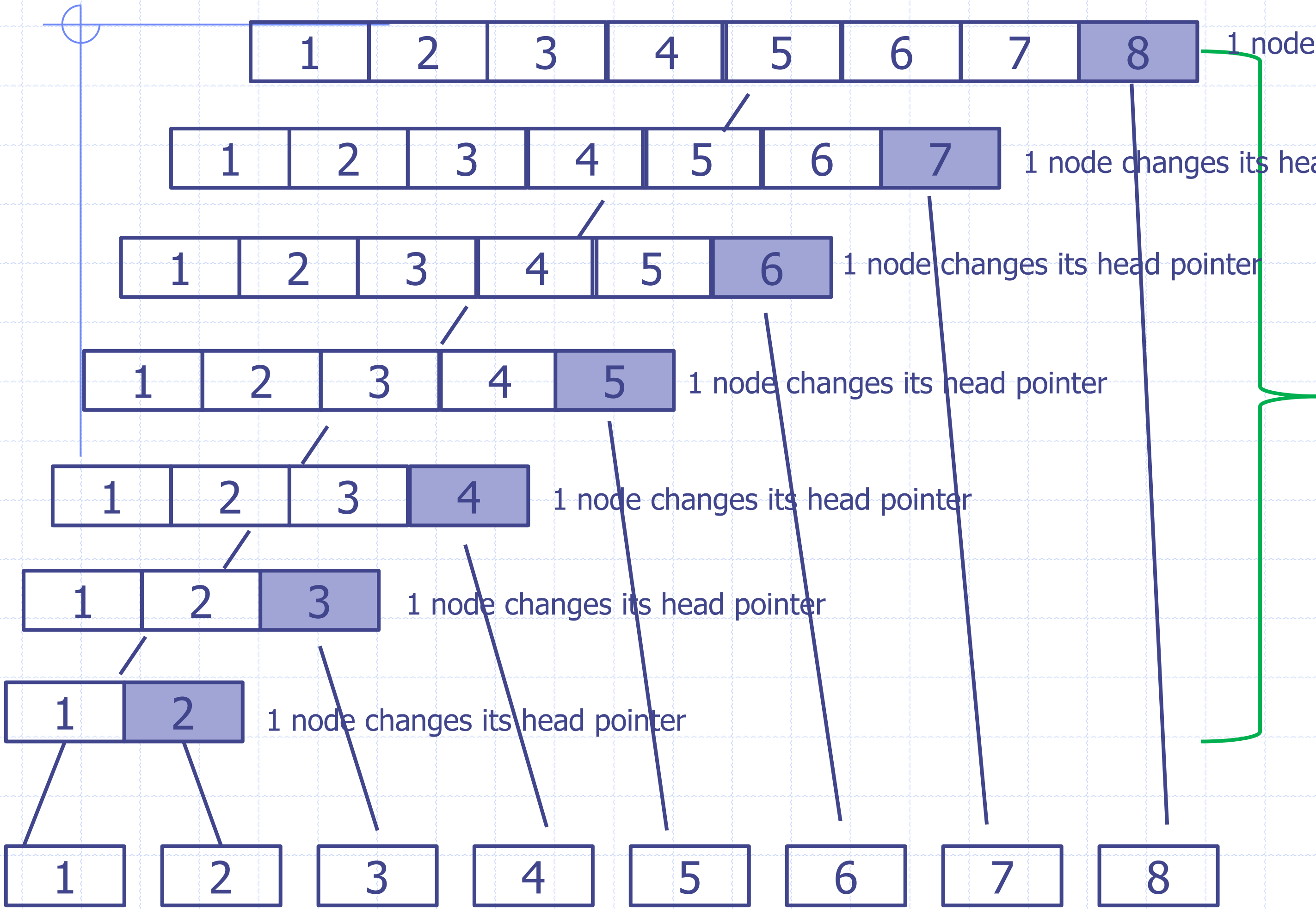
- The running time of this algorithm is O(t(n,n+m)), where t(j,k) is the time for k union-find operations starting from j singleton sets.

List-based Implementation

- Each set is stored in a sequence represented with a linked-list
- Each node should store an object containing the element and a reference to the set name



Largest Number of Calls to Union Until Only 1 Component Exists

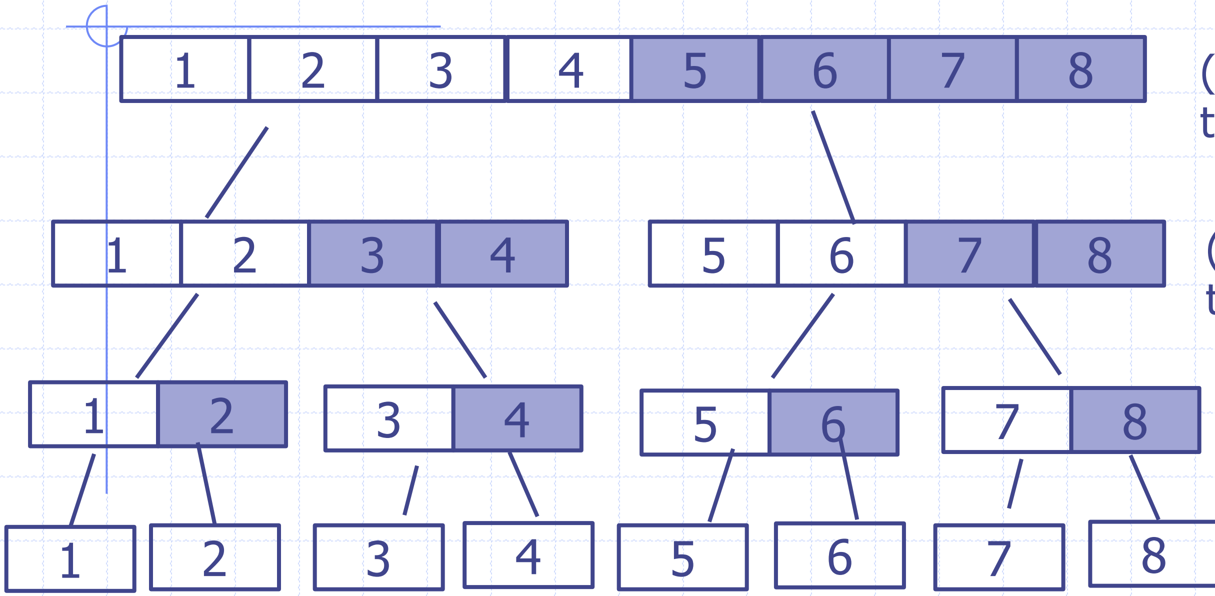


(n-1) calls to union; the height of the tree with only one node changing its header pointer.

Thus, 1*(n-1) times.

A node that belongs to the bigger list does not change its head pointer.

Worst Case Number of Times a Node Changes it Head Pointer Until Only One Component Exists



(½ n) change their head pointer

(½ n) change their head pointer

(½ n) change their head pointer

At most, (½ n log n) = O(n log n) changes of the head pointer

Analysis of List-based Representation

- When doing a union, always move elements from the smaller set to the larger set
  - Each time an element is moved it goes to a set of size at least double its old set
  - Thus, an element can be moved at most $O(\log n)$ times
- Total time needed to do n unions and m finds is $O(n \log n + m)$.

Thank You !

Questions ?