**Algorithms – Part I – Lecture Notes**

**Week 1**

**Union-Find**

**Dynamic Connectivity**

Steps to Developing a Usable Algorithm

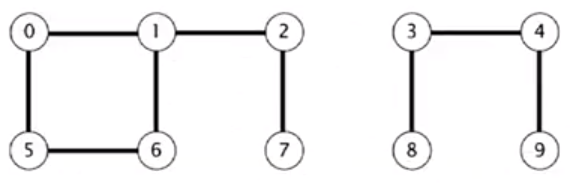
* Model the problem
* Find an algorithm to solve it
* Fast enough? Fits in memory?
* If not, address the issue
* Iterate until satisfied

Dynamic Connectivity

* Given a set of objects, we must support the following commands
  + Union command: connect two objects
  + Find/connected query: is there a path connecting the two objects



* After we call union(5, 0), union(7, 2), union(6, 1), union(1, 0), we get the following graph.



* Then, given the command connected(0, 7), this returns true.

Modeling the Objects

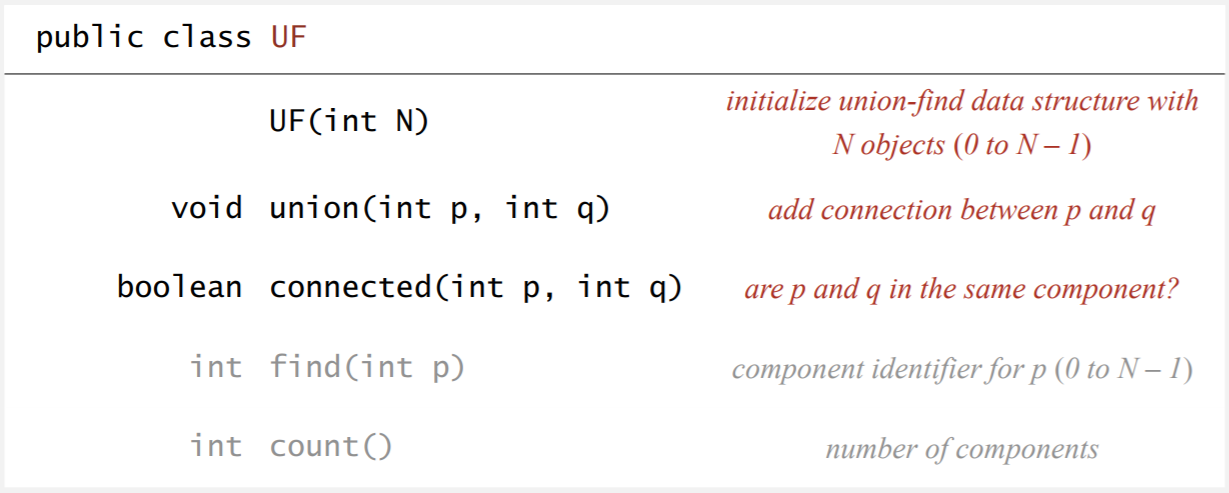
* Pixels in a digital photo
* Computers in a network
* Friends in a social network
* Transistors in a computer chip
* Elements in a mathematical set
* Variable names in FORTRAN
* Metallic sites in a composite system
* When programming, it is convenient to name objects from 0 to .

Modeling the Connections

* We assume “is connected to” is an equivalence relation:
  + Reflexive: is connected to
  + Symmetric: if is connected to , then is connected to
  + Transitive: if is connected to and is connected to , then is connected to
* **Connected components:** Maximal *set* of objects that are mutually connected.

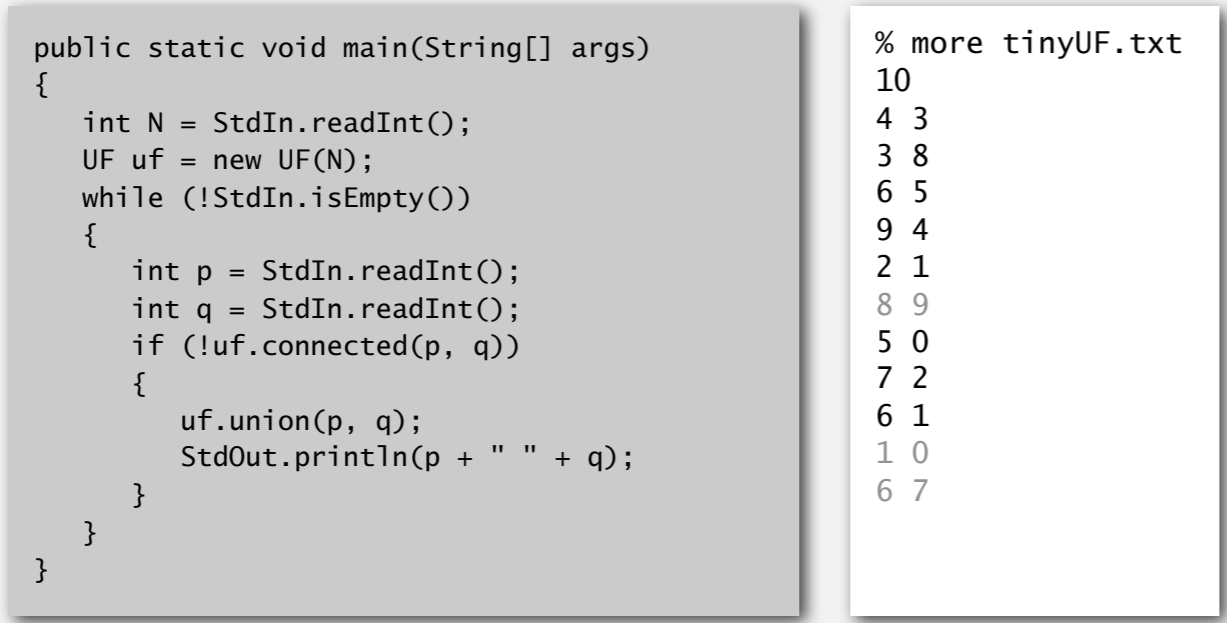
Implementing the Operations

* *Find query.* Check if two objects are in the same component.
* *Union command.* Replace components containing two objects with their union.



Dynamic-Connectivity Client

* Read in number of objects from standard input.
* Repeat:
  + read in pair of integers from standard input
  + if they are not yet connected, connect them and print out a pair



**Quick Find**

[Eager Approach]

* Data structure used:
  + Integer array id[] of size N
  + Interpretation: and are connected iff they have the same id



* *Find.* Check if and have the same
* *Modify the Quick-Union Too:* Link root of smaller tree to smarter tree
  + *Proposition.* Delph said constant time means

Weighted Quick-Union Analysis

* Running time of all the different parts of the algorithm.
  + *Find:* takes time proportional to and proportional to the depth of and . We see that really trying to work.

Nolaysis. Prestionzzzz