http://algs4.cs.princeton.edu

1.5 UNION-FIND

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Subtext of today's lecture (and this course)

Steps to developing a usable algorithm.

- Model the problem.
- Find an algorithm to solve it.
- Fast enough? Fits in memory?
- If not, figure out why not.
- Find a way to address the problem.
- Iterate until satisfied.

The scientific method.

Mathematical analysis.

http://algs4.cs.princeton.edu

1.5 Union-Find

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Dynamic connectivity problem

Given a set of N objects, support two operation:

- Connect two objects.
- Is there a path connecting the two objects?

connect 4 and 3

connect 3 and 8

connect 6 and 5

connect 9 and 4

connect 2 and 1

are 0 and 7 connected? ✗

are 8 and 9 connected? ✔

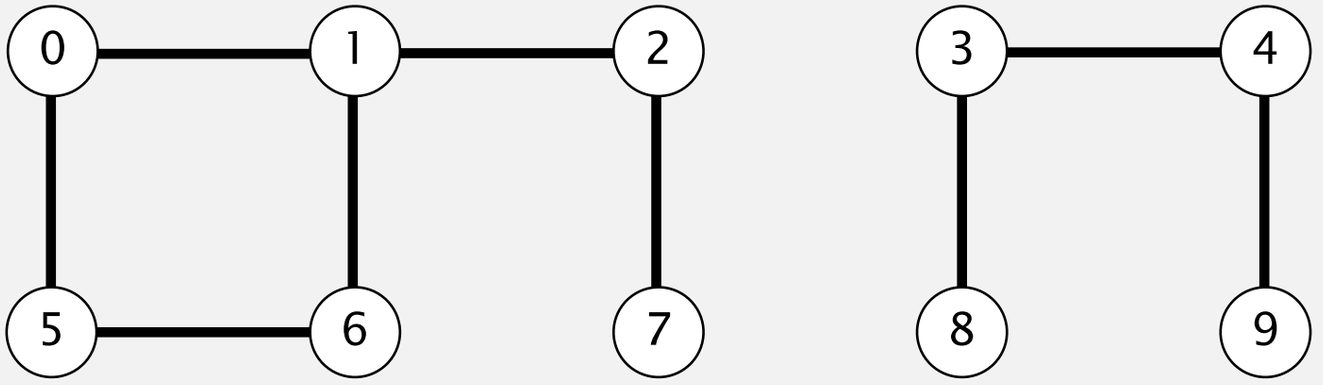
connect 5 and 0

connect 7 and 2

connect 6 and 1

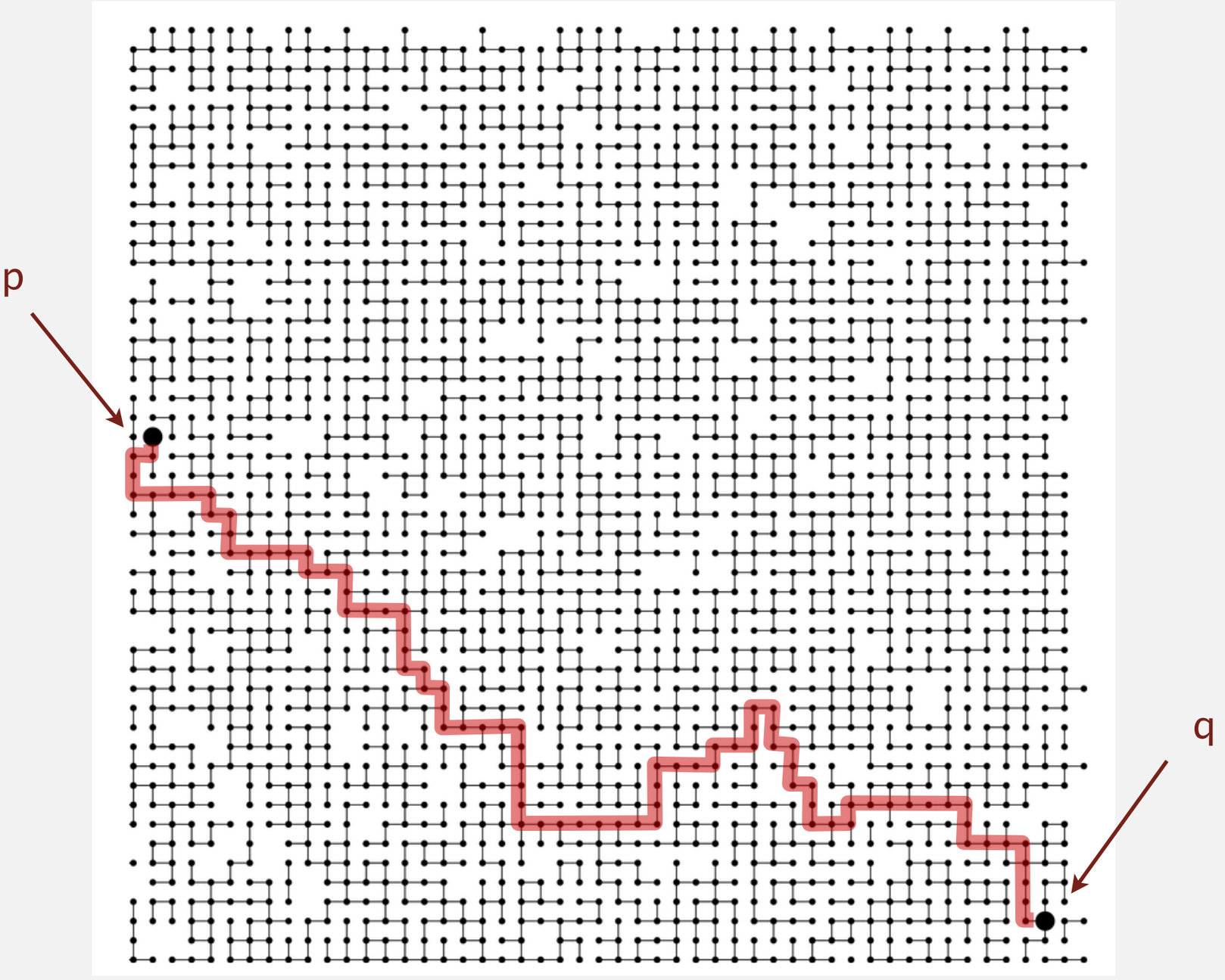
connect 1 and 0

are 0 and 7 connected? ✔



A larger connectivity example

Q. Is there a path connecting p and q ?



A. Yes.

Modeling the objects

Applications involve manipulating objects of all types.

- 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 a Fortran program.
- Metallic sites in a composite system.

When programming, convenient to name objects 0 to N – 1.

- Use integers as array index.
- Suppress details not relevant to union-find.

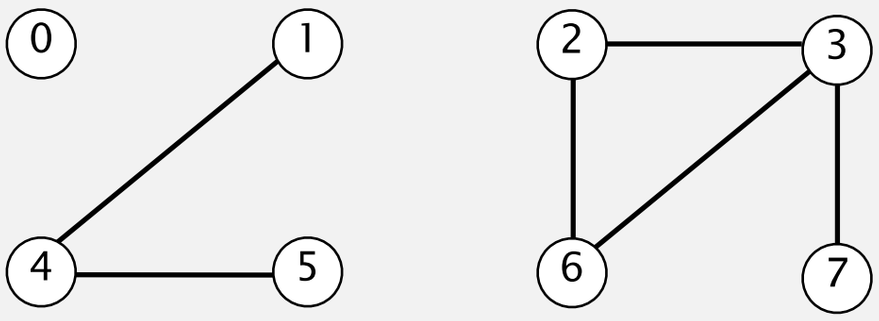
can use symbol table to translate from site names to integers: stay tuned (Chapter 3)

Modeling the connections

We assume "is connected to" is an equivalence relation:

- Reflexive: p is connected to p.
- Symmetric: if p is connected to q, then q is connected to p.
- Transitive: if p is connected to q and q is connected to r, then p is connected to r.

Connected component. Maximal set of objects that are mutually connected.



{ 0 } { 1 4 5 } { 2 3 6 7 }

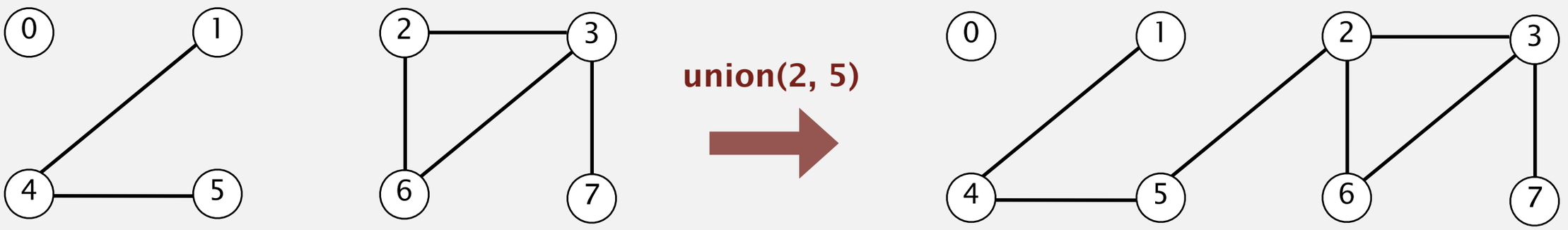
3 connected components

Implementing the operations

Find. In which component is object p ?

Connected. Are objects p and q in the same component?

Union. Replace components containing objects p and q with their union.



{ 0 } { 1 4 5 } { 2 3 6 7 }

3 connected components

{ 0 } { 1 2 3 4 5 6 7 }

2 connected components

Union-find data type (API)

Goal. Design efficient data structure for union-find.

- Number of objects N can be huge.
- Number of operations M can be huge.
- Union and find operations may be intermixed.

public class UF	
UF(int N)	*initialize union-find data structure with N singleton objects (0 to N – 1)*
void union(int p, int q)	*add connection between p and q*
int find(int p)	*component identifier for p (0 to N – 1)*
boolean connected(int p, int q)	*are p and q in the same component?*

```
public boolean connected(int p, int q)
{  return find(p) == find(q);  }
```

1-line implementation of connected()

Dynamic-connectivity client

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

```
public static void main(String[] args)
{
   int N = StdIn.readInt();
   UF uf = new UF(N);
   while (!StdIn.isEmpty())
   {
      int p = StdIn.readInt();
      int q = StdIn.readInt();
      if (!uf.connected(p, q))
      {
         uf.union(p, q);
         StdOut.println(p + " " + q);
      }
   }
}
```

```
% more tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
```

already connected