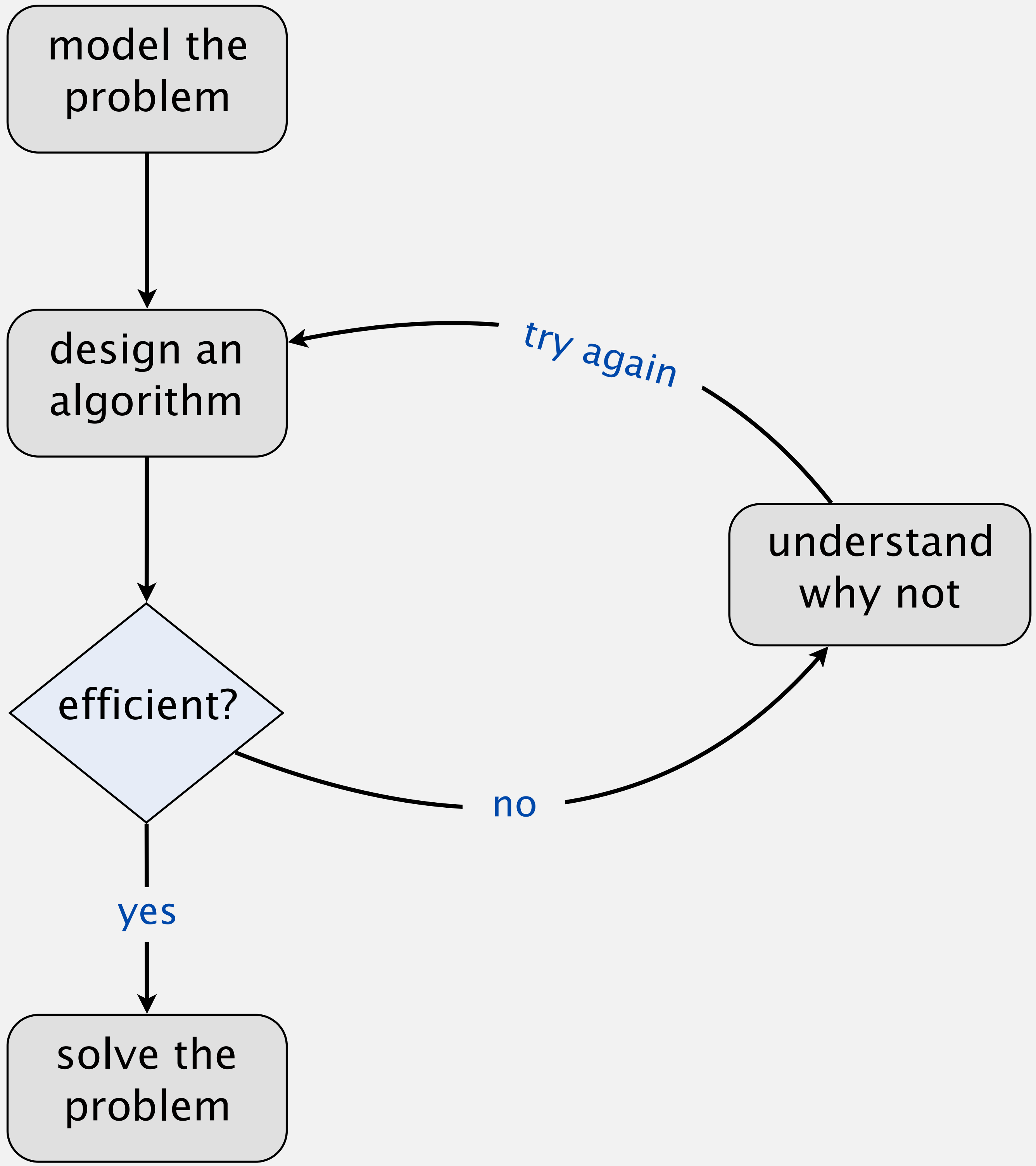
http://algs4.cs.princeton.edu

1.5 Union–Find

- *union–find data type*
- *quick-find*
- *quick-union*
- *improvements*
- *applications*

Subtext of today's lecture (and this course)

Steps to developing a usable algorithm to solve a computational problem.



http://algs4.cs.princeton.edu

1.5 Union–find

- *union–find data type*
- *quick-find*
- *quick-union*
- *improvements*
- *applications*

Union–find data type

Disjoint sets. A collection of sets; each element in exactly one set.

Find. Return a "canonical" element in the set containing p ?

Union. Merge the set containing p with the set containing q.

find(1) = find(4) = find(5) = 4

union(2, 5)

`{ 0 } { 1, 4, 5 } { 2, 3, 6, 7 }` → `{ 0 } { 1, 2, 3, 4, 5, 6, 7 }`

8 elements, 3 disjoint sets — **2 disjoint sets**

Simplifying assumption. The n elements are named 0, 1, ..., $n - 1$.

Union–find data type (API)

Goal. Design an efficient union–find data type.

- Number of elements n can be huge.
- Number of operations m can be huge.
- Union and find operations can be intermixed.

public class UF	
`UF(int n)`	*initialize union–find data structure with n singleton sets (0 to $n - 1$)*
`void union(int p, int q)`	*merge sets containing elements p and q*
`int find(int p)`	*canonical element in set containing p (0 to $n - 1$)*

An application: dynamic connectivity

Given n vertices, support two operations:

- Add edge: directly connect two vertices with an edge.
- Connection query: is there a path connecting two vertices?

add edge 4–3

add edge 3–8

add edge 6–5

add edge 9–4

add edge 2–1

are 8 *and* 9 *connected?* ✔

are 5 *and* 7 *connected?* ✕

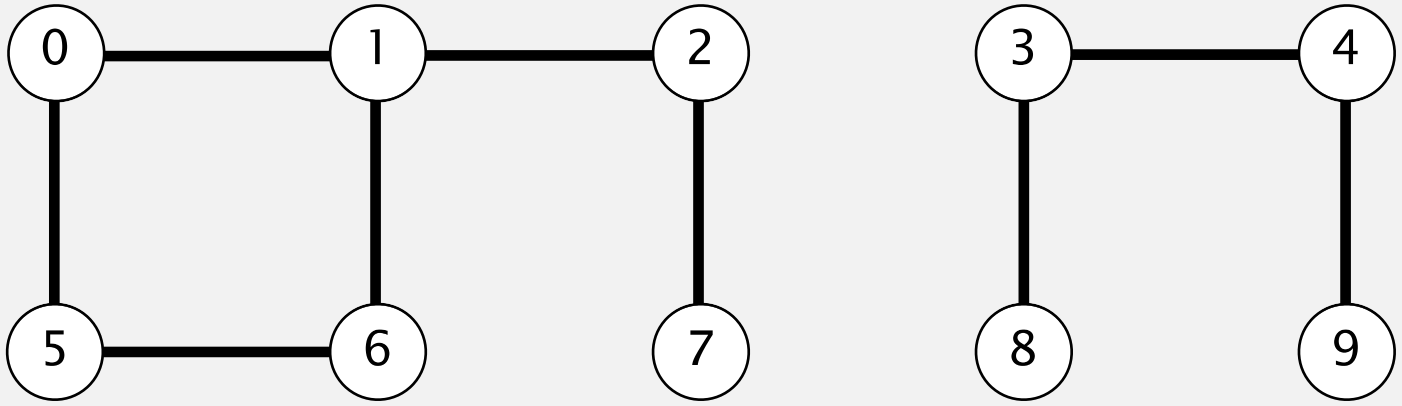
add edge 5–0

add edge 7–2

add edge 6–1

add edge 1–0

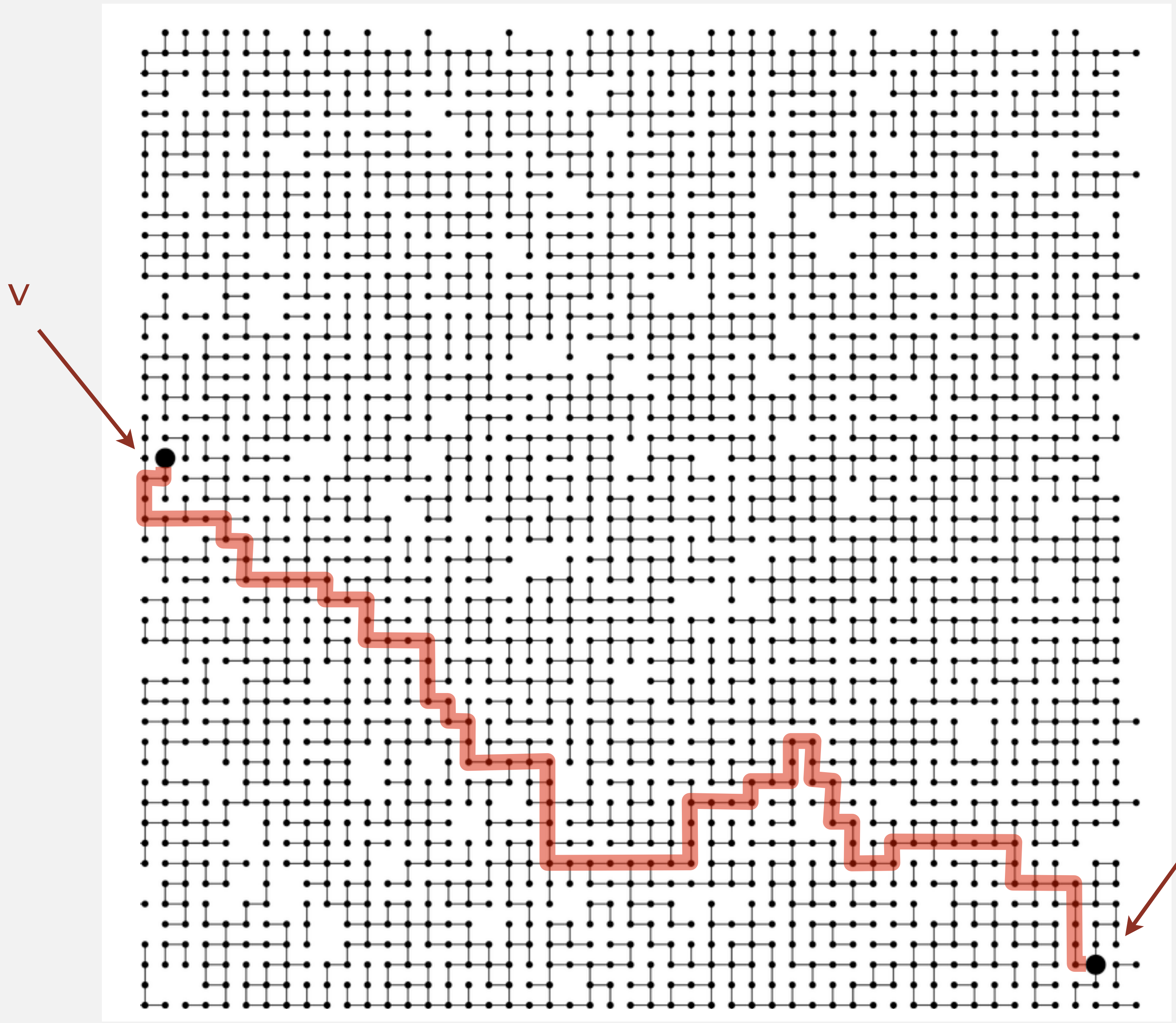
are 5 *and* 7 *connected?* ✔



A larger connectivity example

Q. Is there a path connecting vertices v and w ?

finding a path is a slightly harder problem
(stay tuned for graph algorithms in Chapter 4)



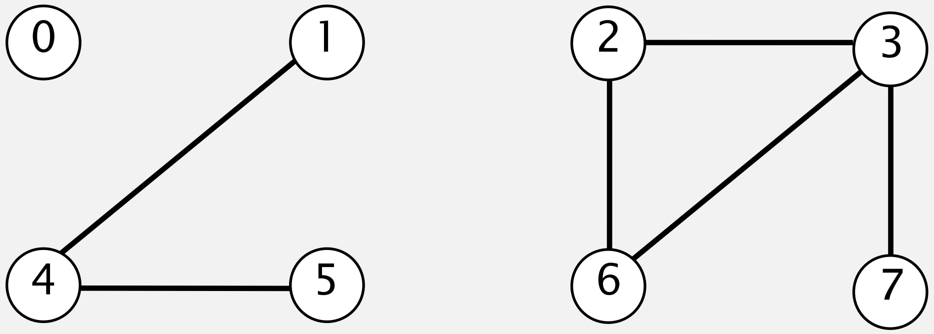
A. Yes.

Modeling the dynamic-connectivity problem

Q. How to model the dynamic-connectivity problem using union–find?

A. Maintain disjoint sets that correspond to connected components.

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



3 connected components

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

3 disjoint sets

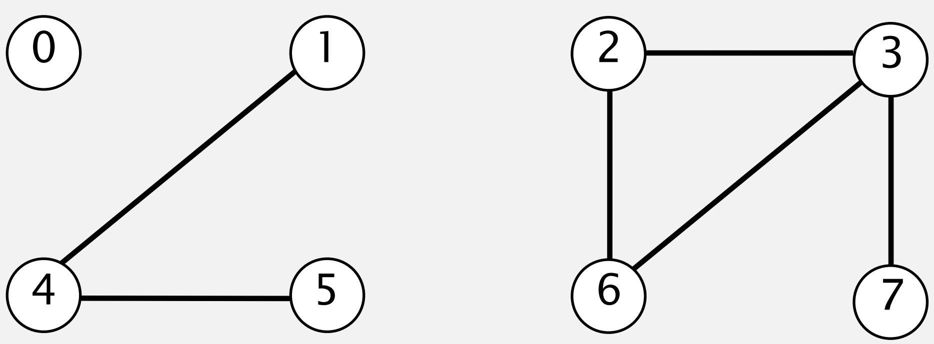
Modeling the dynamic-connectivity problem

Q. How to model the dynamic-connectivity problem using union–find?

A. Maintain disjoint sets that correspond to connected components.

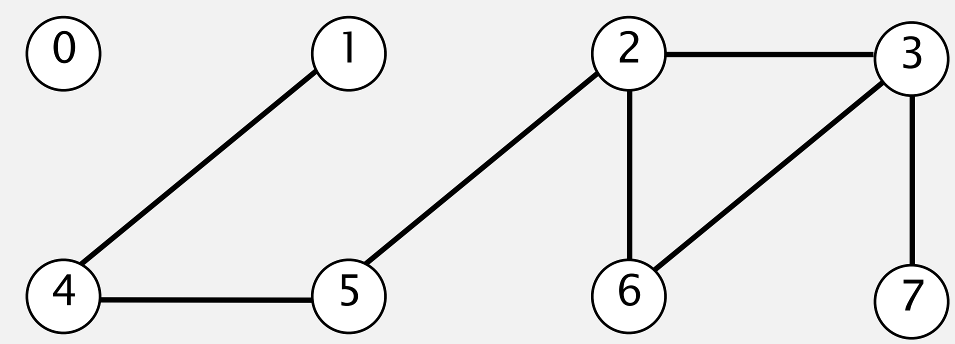
- Add edge between vertices v and w.
- Are vertices v and w connected?

add edge 2–5



3 connected components

are vertices 5 and 6 connected?



2 connected components

union(2, 5)

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

3 disjoint sets

find(5) == find(6) ✔

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

2 disjoint sets

http://algs4.cs.princeton.edu

1.5 Union–find

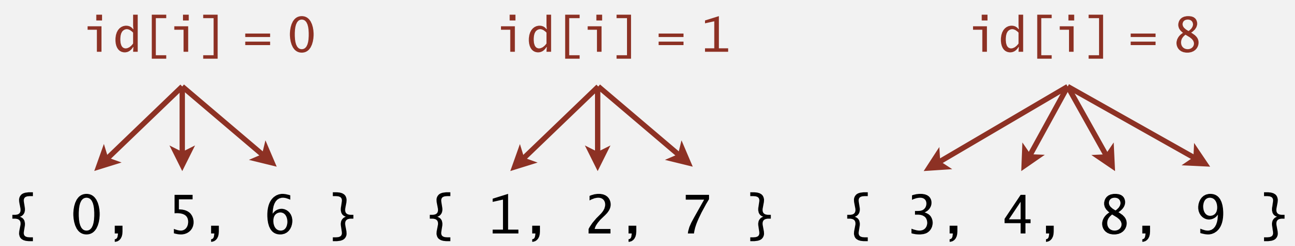
- union–find data type
- *quick-find*
- quick-union
- improvements
- applications

Quick-find [eager approach]

Data structure.

- Integer array `id[]` of length `n`.
- Interpretation: `id[p]` is canonical element in the set containing `p`.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	0	0	1	8	8



3 disjoint sets

Q. How to implement `find(p)`?

A. Easy, just return `id[p]`.

Quick-find [eager approach]

Data structure.

- Integer array `id[]` of length `n`.
- Interpretation: `id[p]` is canonical element in the set containing `p`.

union(6, 1)

	0	1	2	3	4	5	6	7	8	9
`id[]`	1	1	1	8	8	1	1	1	8	8

problem: many values can change

Q. How to implement `union(p, q)`?

A. Change all entries whose identifier equals `id[p]` to `id[q]`.

Quick-find: Java implementation

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int n)
   {
      id = new int[n];
      for (int i = 0; i < n; i++)
         id[i] = i;
   }

   public int find(int p)
   {  return id[p];  }

   public void union(int p, int q)
   {
      int pid = id[p];
      int qid = id[q];
      for (int i = 0; i < id.length; i++)
         if (id[i] == pid) id[i] = qid;
   }
}
```

set id of each element to itself
(n array accesses)

return the id of p
(1 array access)

change all entries with id[p] to id[q]
(n+2 to 2n+2 array accesses)

http://algs4.cs.princeton.edu/15uf/QuickFindUF.java.html

Quick-find is too slow

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	n	n	1

number of array accesses (ignoring leading constant)

Union is too expensive. Processing a sequence of n union operations on n elements takes more than n^2 array accesses.

quadratic

http://algs4.cs.princeton.edu

1.5 UNION–FIND

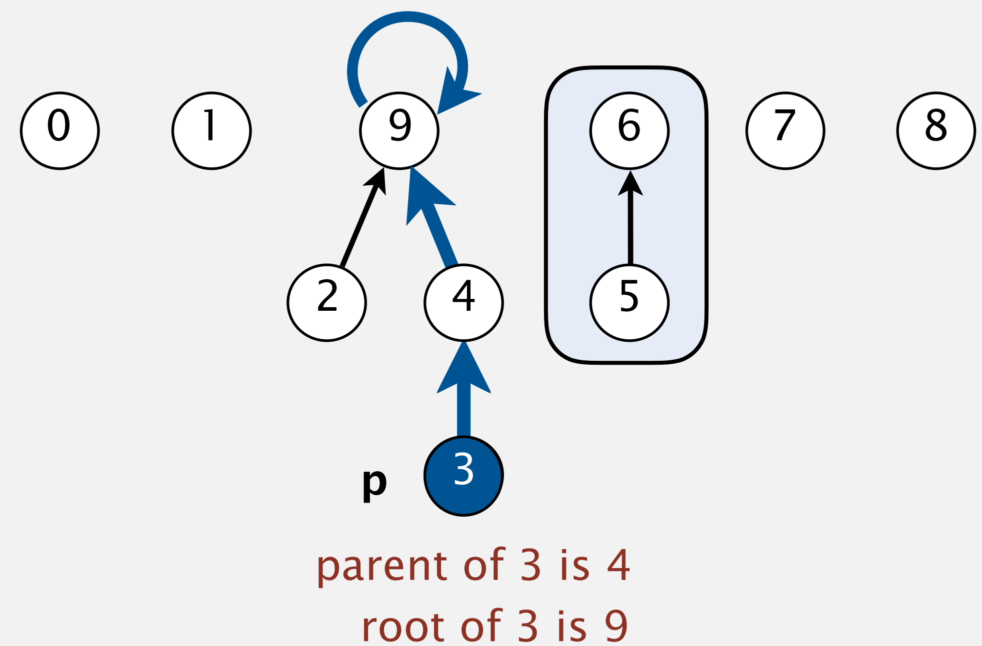
- union–find data type
- quick-find
- **quick-union**
- improvements
- applications

Quick-union [lazy approach]

Data structure.

- Integer array `parent[]` of length `n`, where `parent[i]` is parent of `i` in tree.
- Interpretation: elements in one tree correspond to one set.

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



`find(i) = 9`

{ 0 } { 1 } { 2, 3, 4, 9 } { 5, 6 } { 7 } { 8 }

6 disjoint sets (6 trees)

Q. How to implement `find(p)` operation?

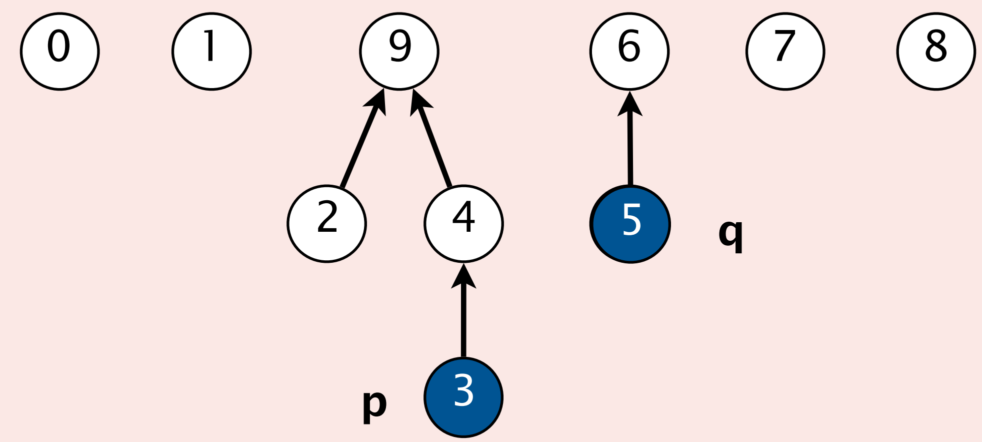
A. Return root of tree containing `p`.

Quick-union quiz

Data structure.

- Integer array `parent[]` of length `n`, where `parent[i]` is parent of `i` in tree.
- Interpretation: elements in one tree correspond to one set.

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



How to implement `union(3, 5)` ?

A. Set parent of 3 to 5.

B. Set parent of 9 to 5.

C. Set parent of 9 to 6.

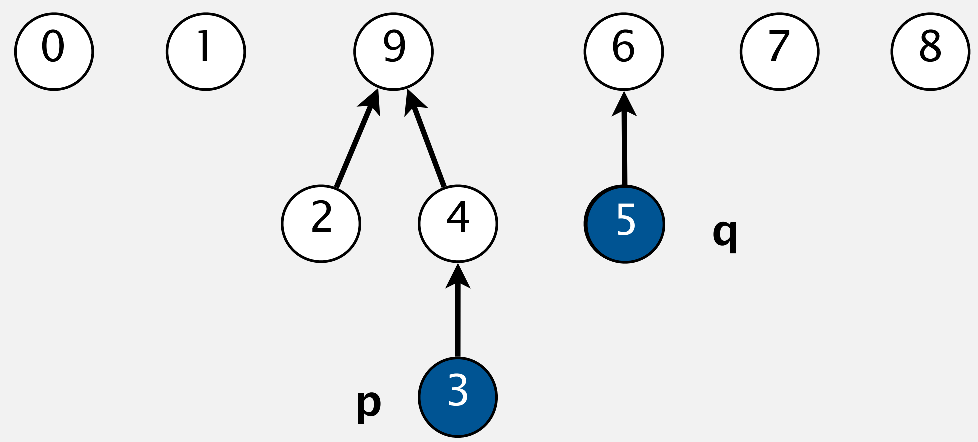
D. Set parents of 2, 3, 4, and 9 each to 6.

Quick-union [lazy approach]

Data structure.

- Integer array `parent[]` of length `n`, where `parent[i]` is parent of `i` in tree.
- Interpretation: elements in one tree correspond to one set.

	0	1	2	3	4	5	6	7	8	9
union(3, 5)	0	1	9	4	9	6	6	7	8	9



Q. How to implement `union(p, q)`?

A. Set parent of `p`'s root to parent of `q`'s root.

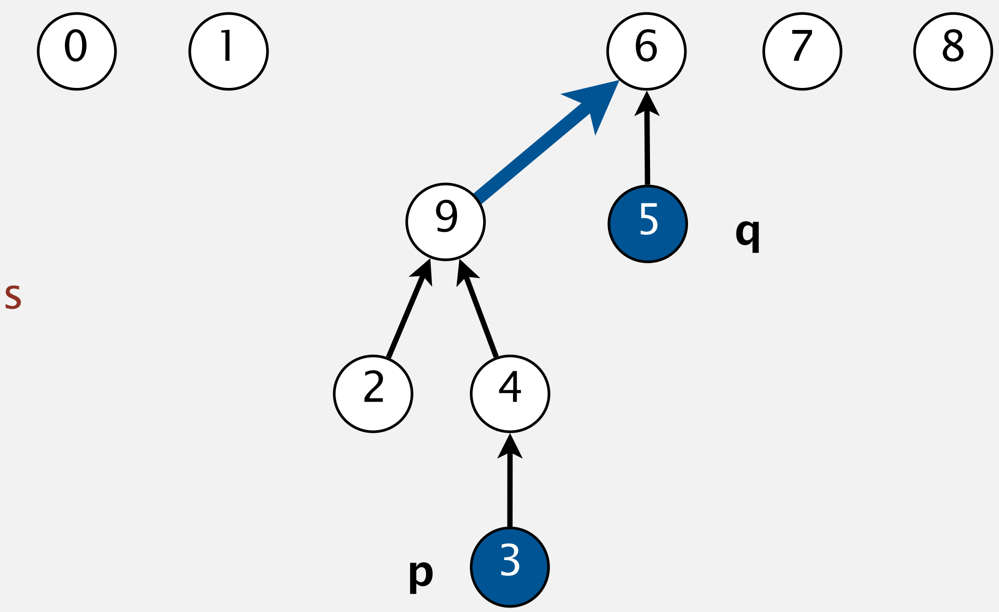
Quick-union [lazy approach]

Data structure.

- Integer array `parent[]` of length `n`, where `parent[i]` is parent of `i` in tree.
- Interpretation: elements in one tree correspond to one set.

	0	1	2	3	4	5	6	7	8	9
union(3, 5)	0	1	9	4	9	6	6	7	8	6

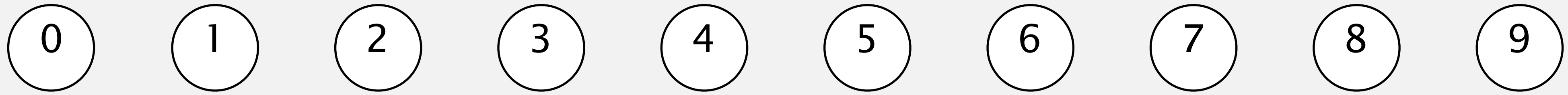
only one value changes



Q. How to implement `union(p, q)`?

A. Set parent of `p`'s root to parent of `q`'s root.

Quick-union demo



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

Quick-union: Java implementation

```
public class QuickUnionUF
{
   private int[] parent;

   public QuickUnionUF(int n)
   {
      parent = new int[n];
      for (int i = 0; i < n; i++)
         parent[i] = i;
   }

   public int find(int p)
   {
      while (p != parent[p])
         p = parent[p];
      return p;
   }

   public void union(int p, int q)
   {
      int rootP = find(p);
      int rootQ = find(q);
      parent[rootP] = rootQ;
   }
}
```

set parent of each element to itself (n array accesses)

chase parent pointers until reach root (depth of p array accesses)

change root of p to point to root of q (depth of p and q array accesses)

http://algs4.cs.princeton.edu/15uf/QuickUnionUF.java.html

Quick-union is also too slow

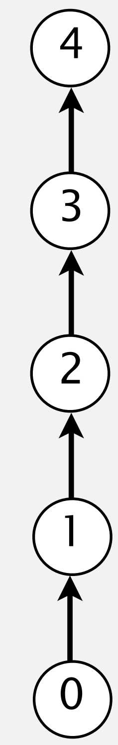
Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	n	n	1
quick-union	n	n	n

worst case

number of array accesses (ignoring leading constant)

worst-case input



union(0, 1)
union(0, 2)
union(0, 3)
union(0, 4)

Quick-find defect.

- Union too expensive (more than n array accesses).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.

- Trees can get tall.
- Find too expensive (could be more than n array accesses).

http://algs4.cs.princeton.edu

1.5 Union–find

- union–find data type
- quick-find
- quick-union
- **improvements**
- applications

Weighted quick-union quiz

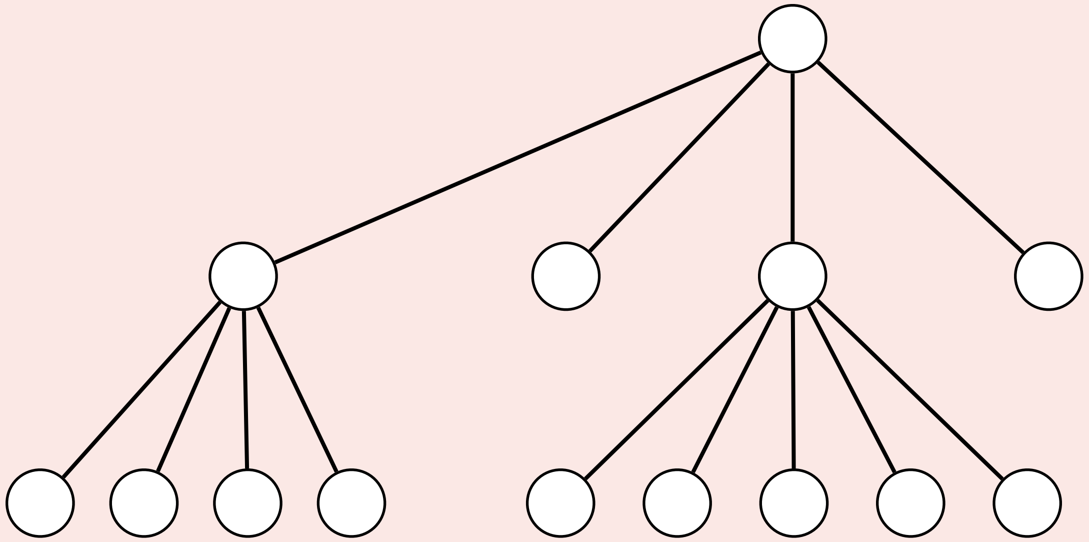
When merging two trees, which strategy is most effective?

A. Link the root of the smaller tree to the root of the larger tree.

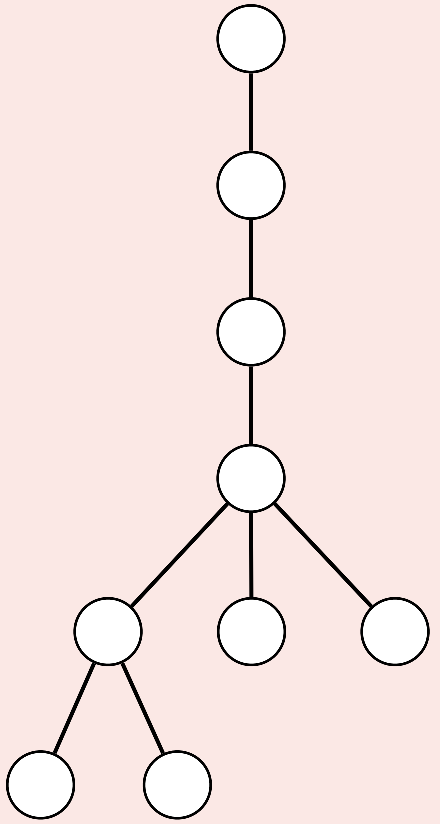
B. Link the root of the larger tree to the root of the smaller tree.

C. Link the root of the shorter tree to the root of the taller tree.

D. Link the root of the taller tree to the root of the shorter tree.



shorter and larger tree
(height = 2, size = 14)



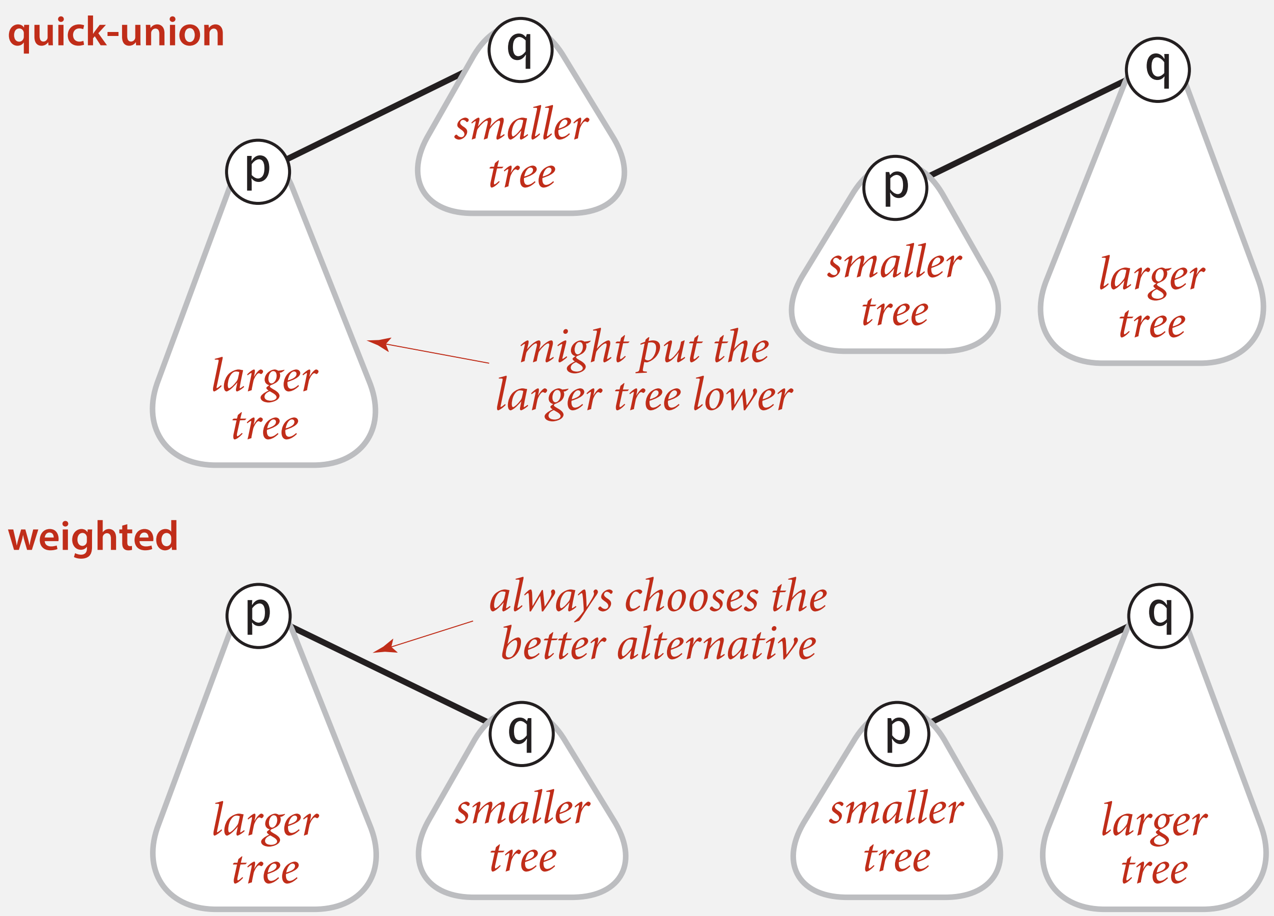
taller and smaller tree
(height = 5, size = 9)

Improvement 1: weighting

Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each tree (number of elements).
- Always link root of smaller tree to root of larger tree.

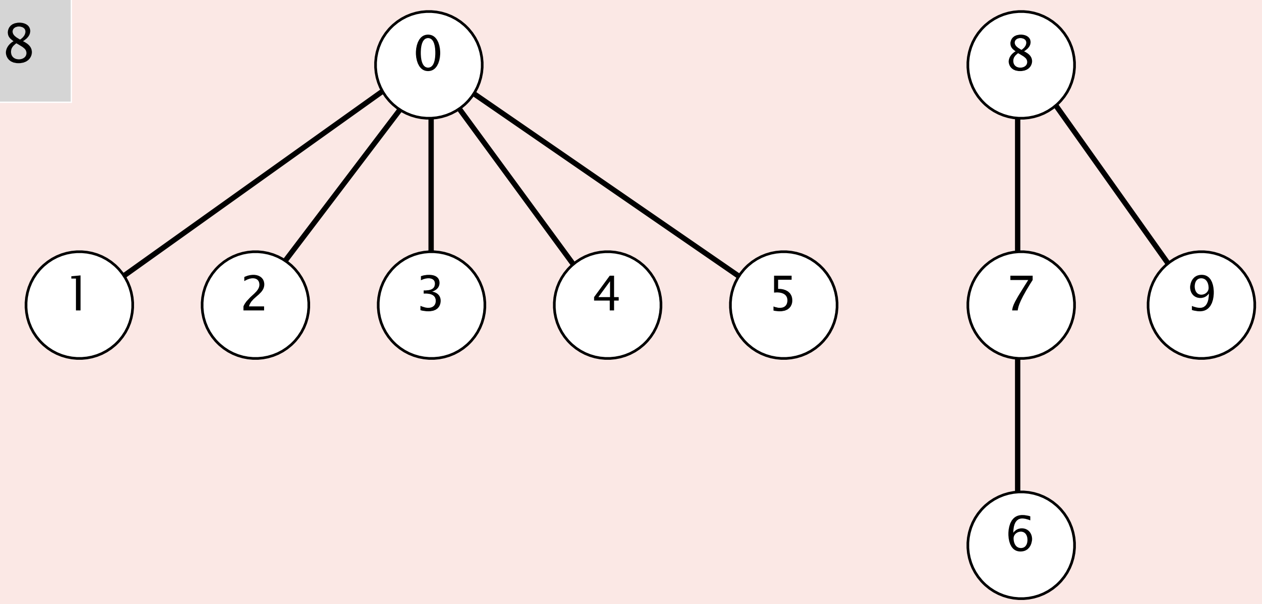
reasonable alternative:
union by height/rank



Weighted quick-union quiz

Suppose that the `parent[]` array during weighted quick-union is:

	0	1	2	3	4	5	6	7	8	9
parent[]	0	0	0	0	0	0	7	8	8	8



Which `parent[]` entry changes during `union(2, 6)`?

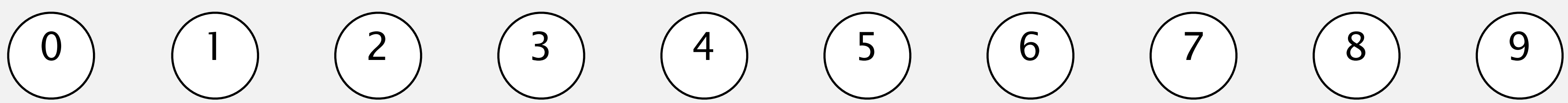
A. `parent[0]`

B. `parent[2]`

C. `parent[6]`

D. `parent[8]`

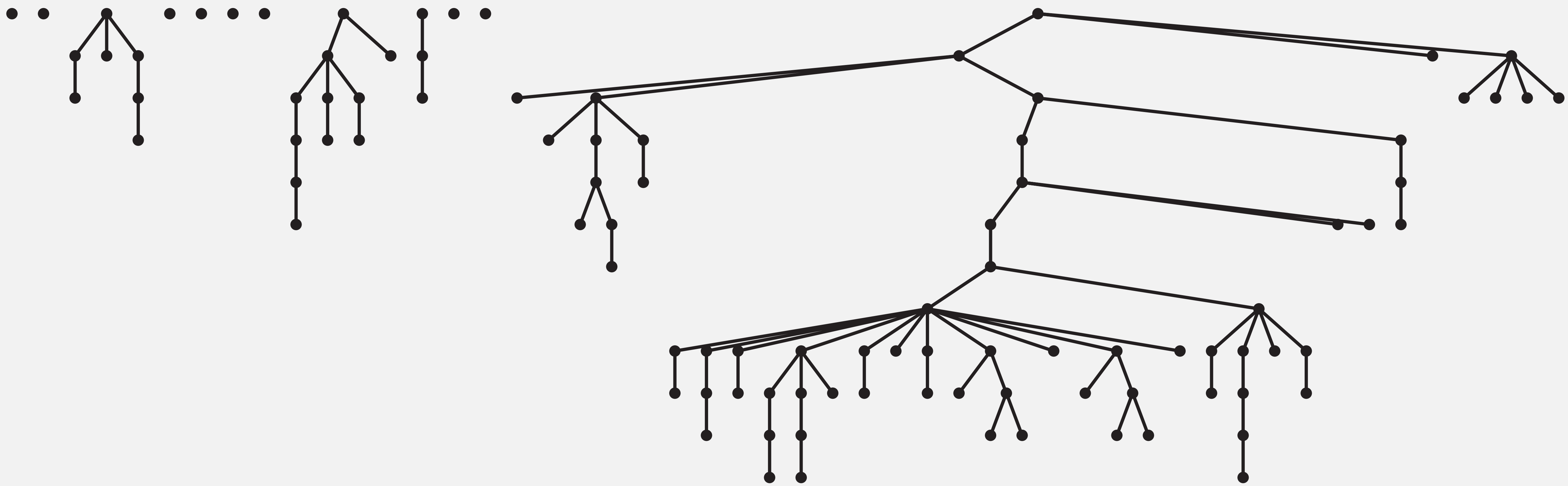
Weighted quick-union demo



	0	1	2	3	4	5	6	7	8	9
parent[]	0	1	2	3	4	5	6	7	8	9

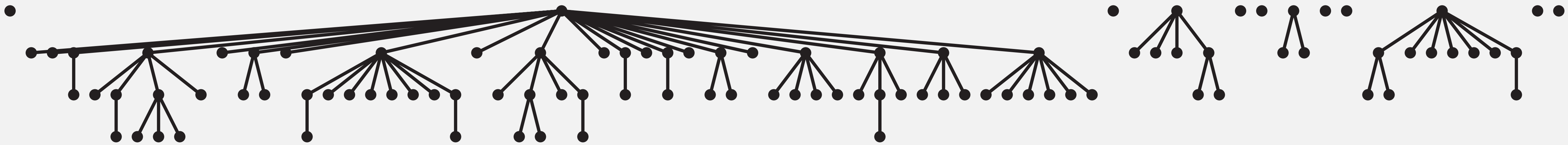
Quick-union vs. weighted quick-union: larger example

quick-union



average distance to root: 5.11

weighted



average distance to root: 1.52

Quick-union and weighted quick-union (100 sites, 88 `union()` operations)

Weighted quick-union: Java implementation

Data structure. Same as quick-union, but maintain extra array `size[i]` to count number of elements in the tree rooted at `i`, initially `1`.

- Find: identical to quick-union.
- Union: link root of smaller tree to root of larger tree; update `size[]`.

```
public void union(int p, int q)
{
   int root1 = find(p);
   int root2 = find(q);
   if (root1 == root2) return;

   if (size[root1] >= size[root2])
   { int temp = root1; root1 = root2; root2 = temp; }

   parent[root1] = root2;
   size[root2] += size[root1];
}
```

root1 is root of smaller tree

link root of smaller tree to root of larger tree

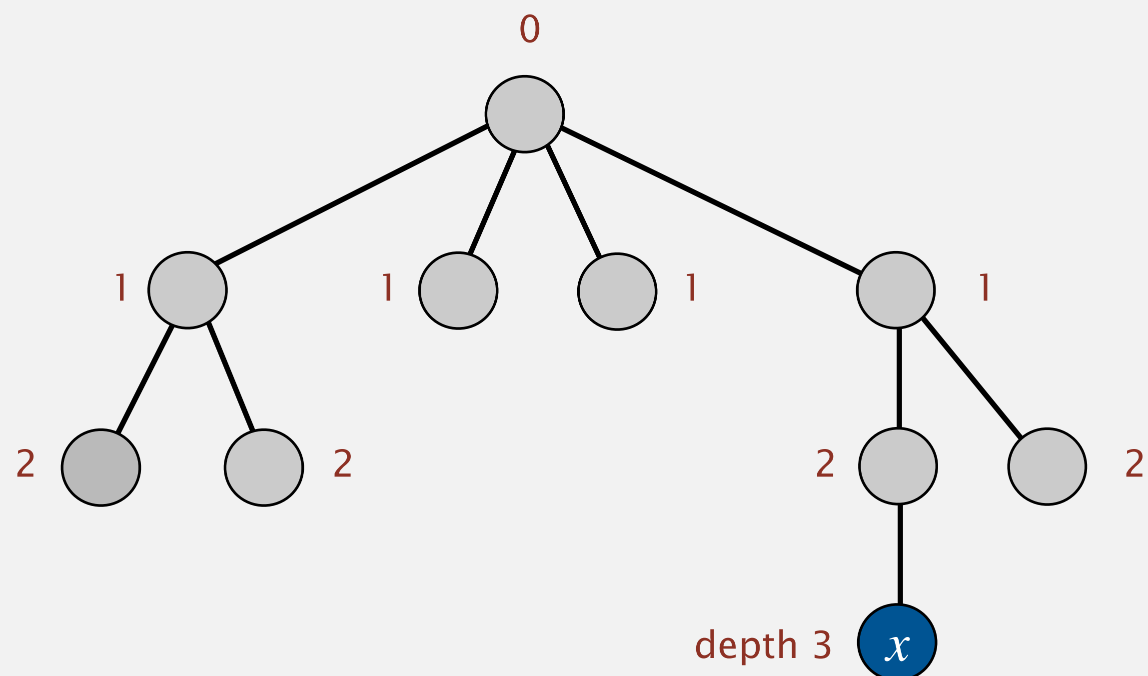
http://algs4.cs.princeton.edu/15uf/WeightedQuickUnionUF.java.html

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p.
- Union: takes constant time, given two roots.

Proposition. Depth of any node x is at most $\lg n$. ← in computer science, lg means base-2 logarithm



n = 10
depth(x) = 3 ≤ lg n

Weighted quick-union analysis

Running time.

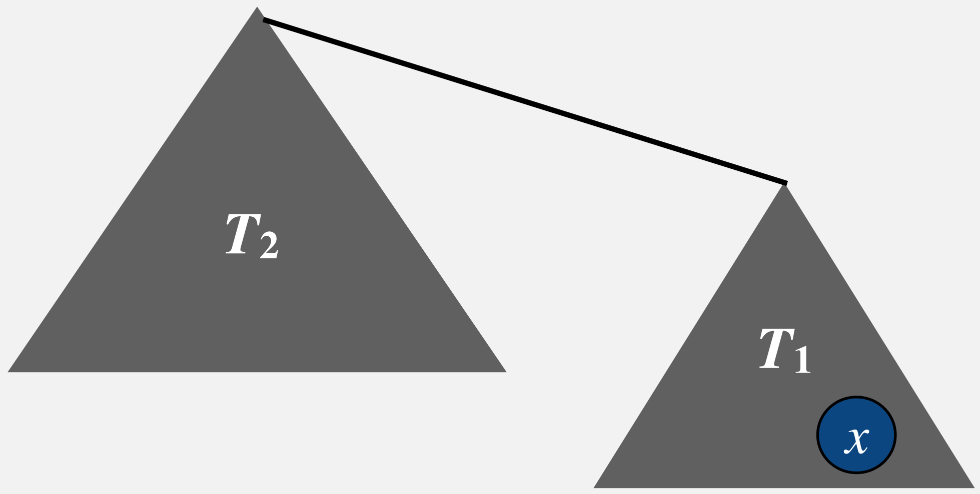
- Find: takes time proportional to depth of p.
- Union: takes constant time, given two roots.

Proposition. Depth of any node x is at most $\lg n$. ← in computer science, lg means base-2 logarithm

Pf. What causes the depth of element x to increase?

Increases by 1 when root of tree T_1 containing x is linked to root of tree T_2.

- The size of the tree containing x at least doubles since $|T_2| \geq |T_1|$.
- Size of tree containing x can double at most $\lg n$ times. Why?



$$\left.\begin{array}{c}1\\2\\4\\8\\16\\\vdots\\n\end{array}\right\}\ \lg n$$

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p.
- Union: takes constant time, given two roots.

Proposition. Depth of any node x is at most $\lg n$.

algorithm	initialize	union	find
quick-find	n	n	1
quick-union	n	n	n
weighted quick-union	n	$\log n$	$\log n$

← log mean logarithm, for some constant base

number of array accesses (ignoring leading constant)

Summary

Key point. Weighted quick-union makes it possible to solve problems that could not otherwise be addressed.

algorithm	worst-case time
quick-find	$m\,n$
quick-union	$m\,n$
weighted quick-union	$n + m \log n$
QU + path compression	$n + m \log n$
weighted QU + path compression	$n + m \log^* n$

order of growth for m union-find operations on a set of n elements

Ex. [10^9 unions and finds with 10^9 elements]

- Weighted quick-union reduces time from 30 years to 6 seconds.
- Supercomputer won't help much; good algorithm enables solution.