

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<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?

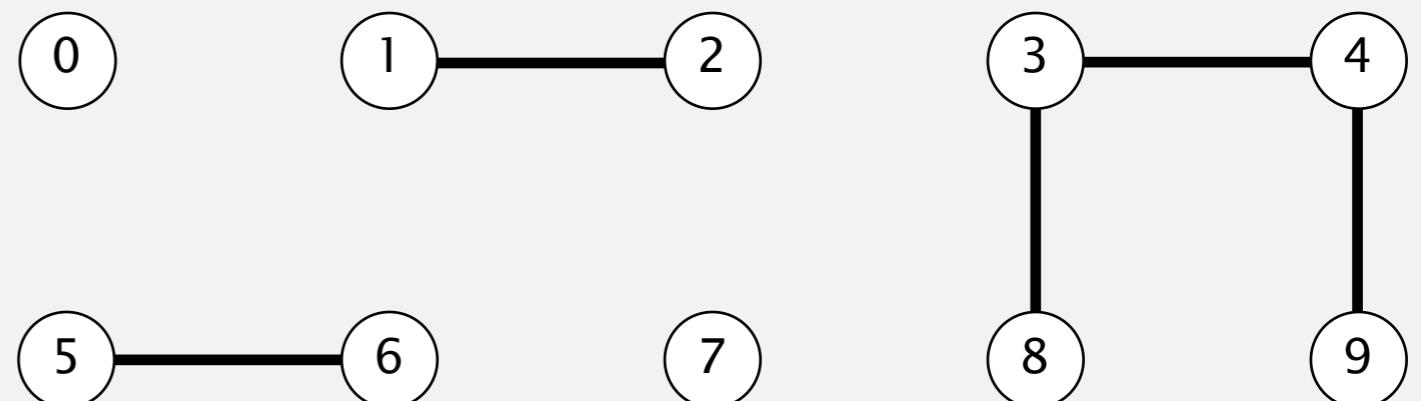
*connect 4 and 3*

*connect 3 and 8*

*connect 6 and 5*

*connect 9 and 4*

*connect 2 and 1*



# 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*

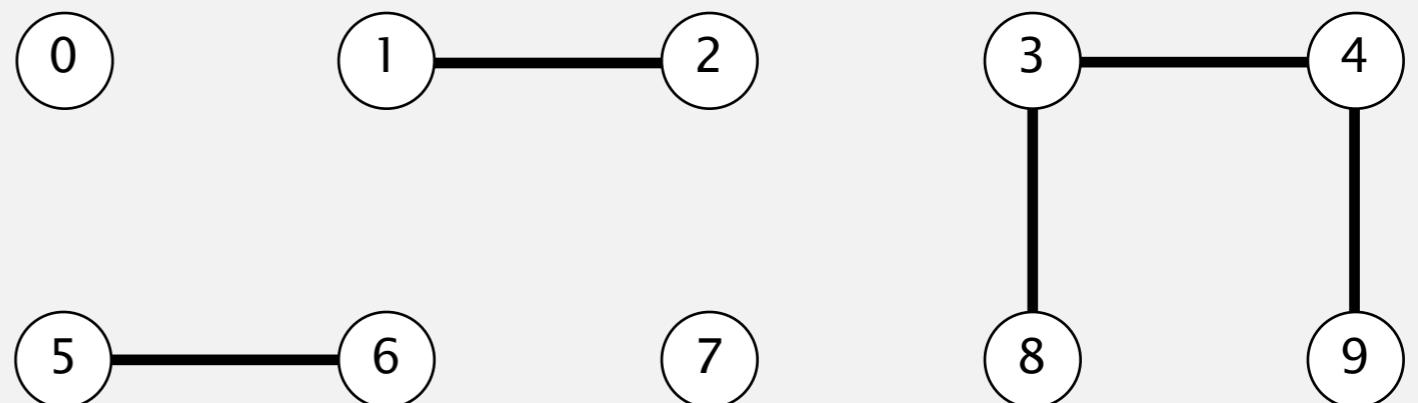
*connect 3 and 8*

*connect 6 and 5*

*connect 9 and 4*

*connect 2 and 1*

*are 0 and 7 connected?*



# 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*

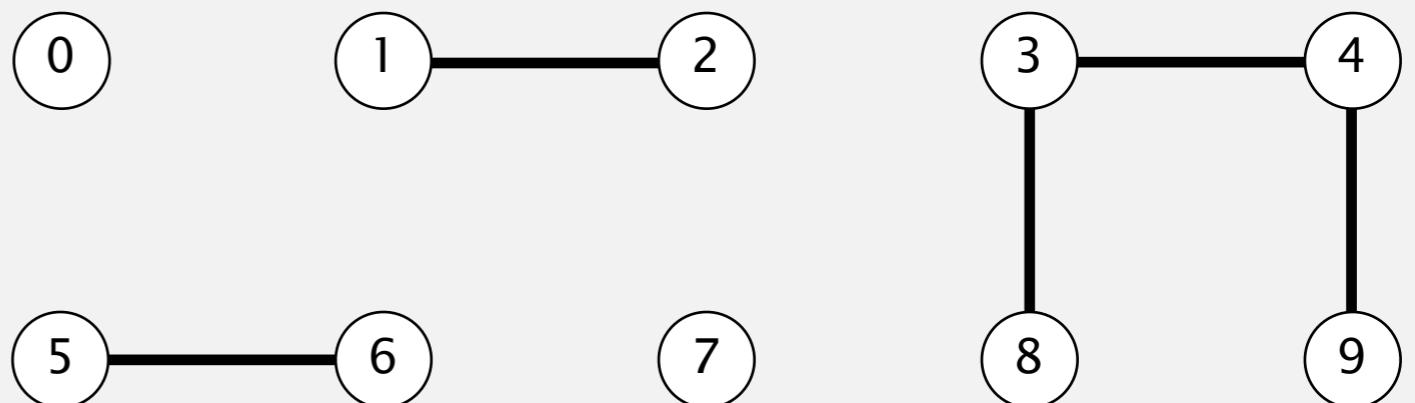
*connect 3 and 8*

*connect 6 and 5*

*connect 9 and 4*

*connect 2 and 1*

*are 0 and 7 connected?    ✗*



# 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*

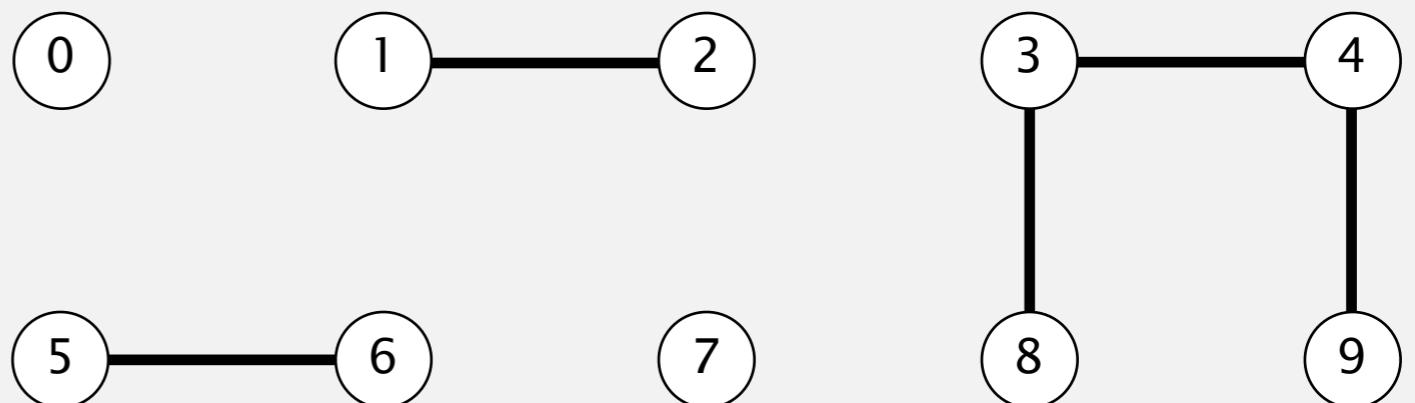
*connect 6 and 5*

*connect 9 and 4*

*connect 2 and 1*

*are 0 and 7 connected?    ✗*

*are 8 and 9 connected?*



# 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*

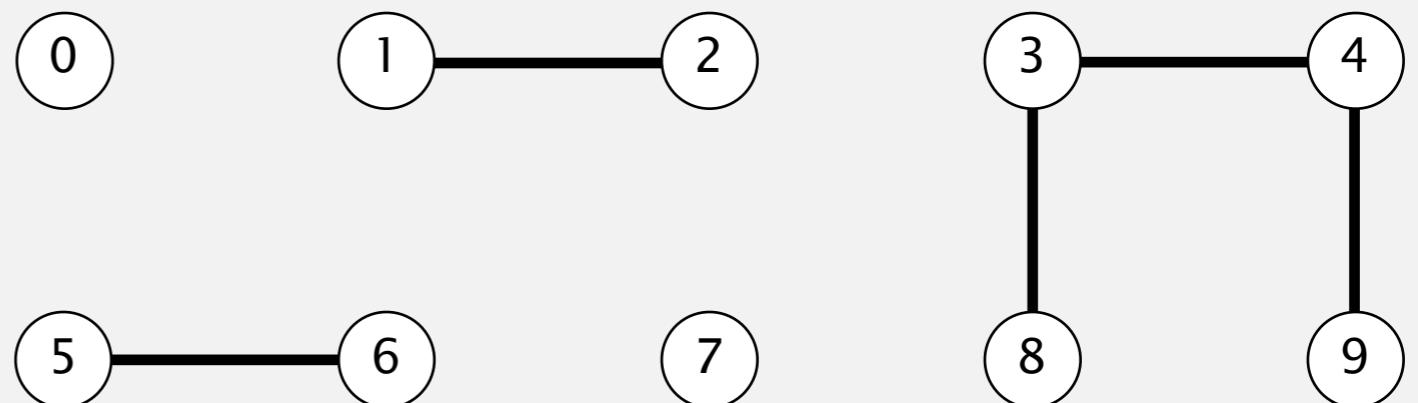
*connect 6 and 5*

*connect 9 and 4*

*connect 2 and 1*

*are 0 and 7 connected?    ✗*

*are 8 and 9 connected?    ✓*



# 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*

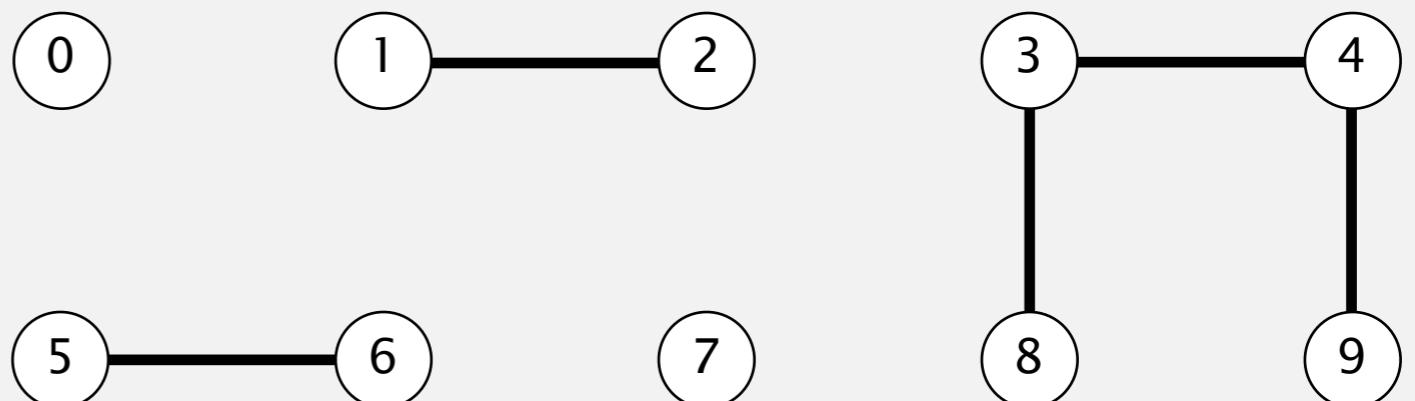
*connect 9 and 4*

*connect 2 and 1*

*are 0 and 7 connected?      ✗*

*are 8 and 9 connected?      ✓*

*connect 5 and 0*



# 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*

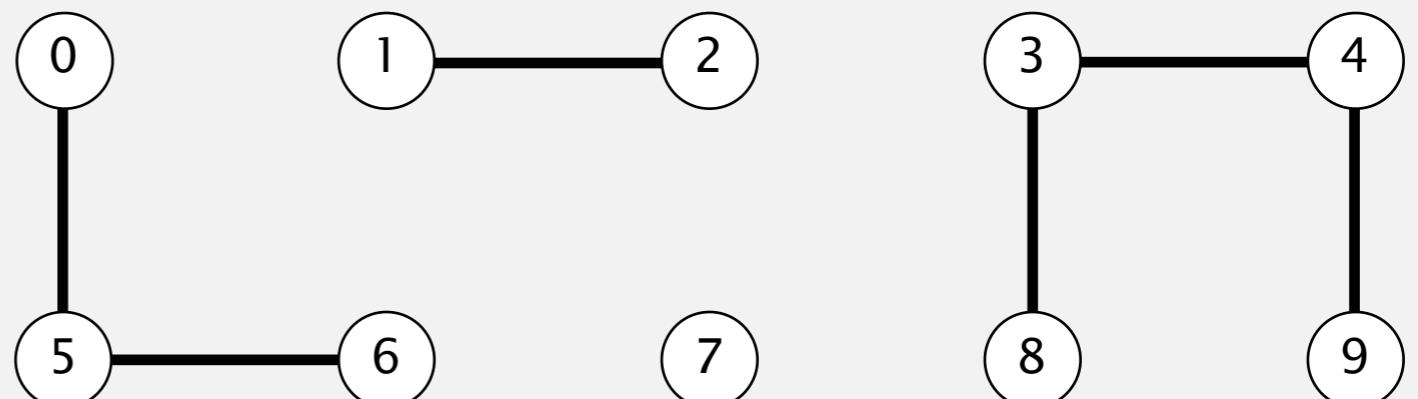
*connect 9 and 4*

*connect 2 and 1*

*are 0 and 7 connected?    ✗*

*are 8 and 9 connected?    ✓*

*connect 5 and 0*



# 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*

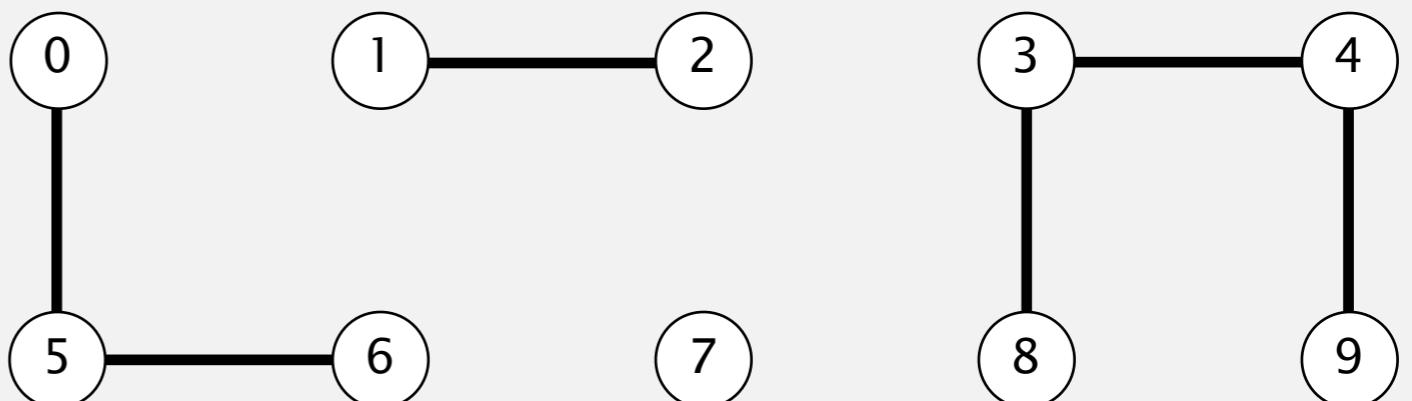
*connect 2 and 1*

*are 0 and 7 connected?      ✗*

*are 8 and 9 connected?      ✓*

*connect 5 and 0*

*connect 7 and 2*



# 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*

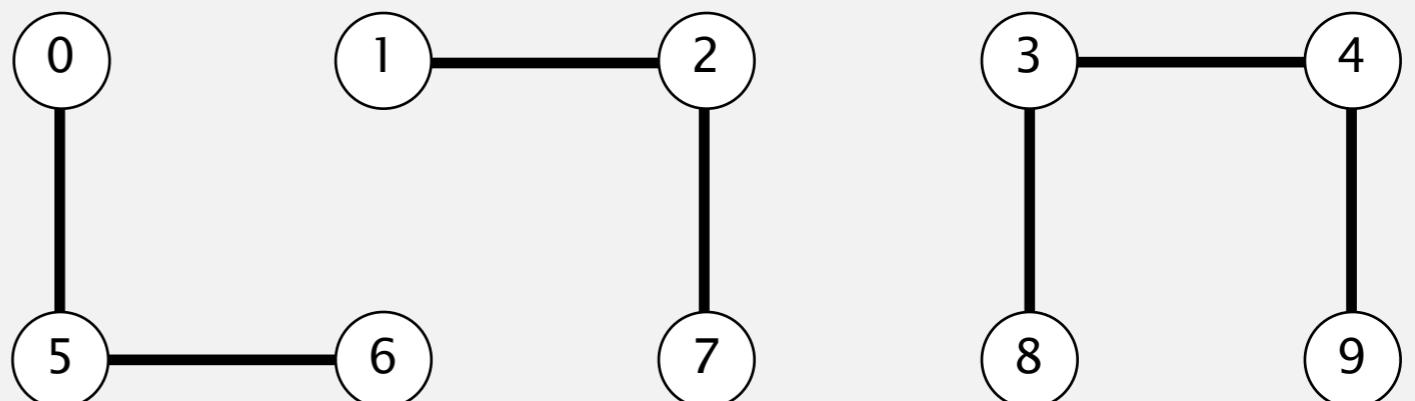
*connect 2 and 1*

*are 0 and 7 connected?      ✗*

*are 8 and 9 connected?      ✓*

*connect 5 and 0*

*connect 7 and 2*



# 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*

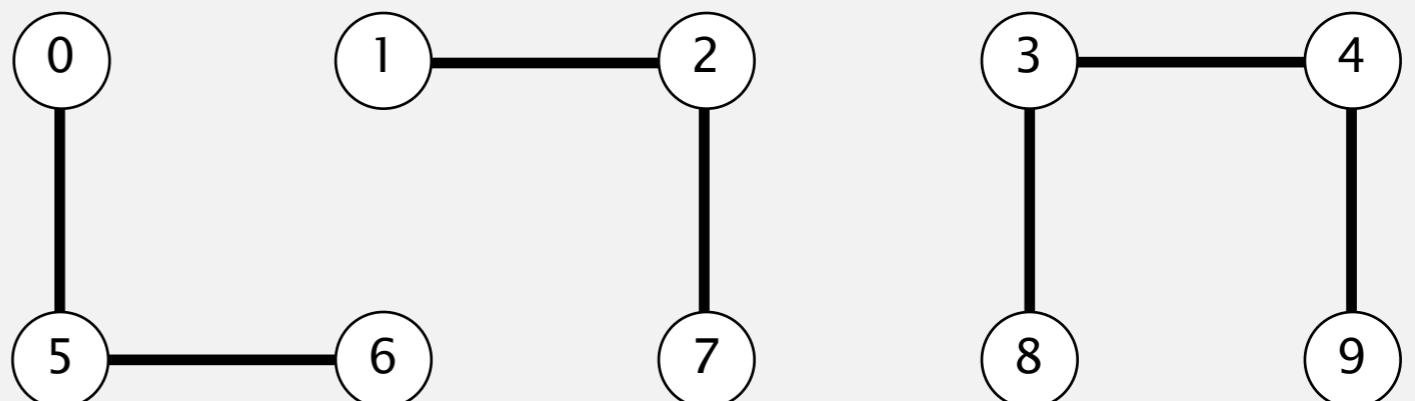
*are 0 and 7 connected?      ✗*

*are 8 and 9 connected?      ✓*

*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*



# 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*

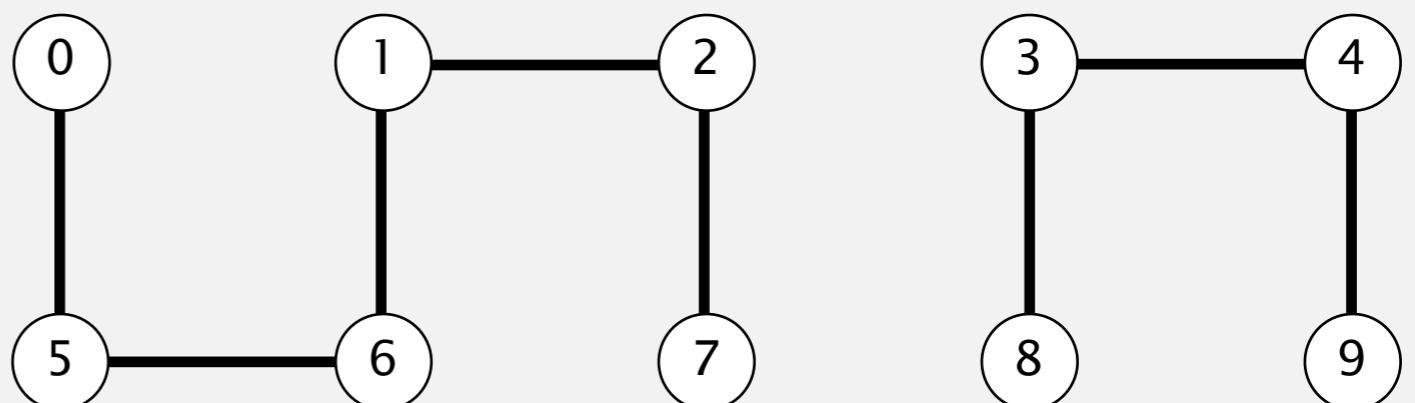
*are 0 and 7 connected?      ✗*

*are 8 and 9 connected?      ✓*

*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*



# 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?      ✗*

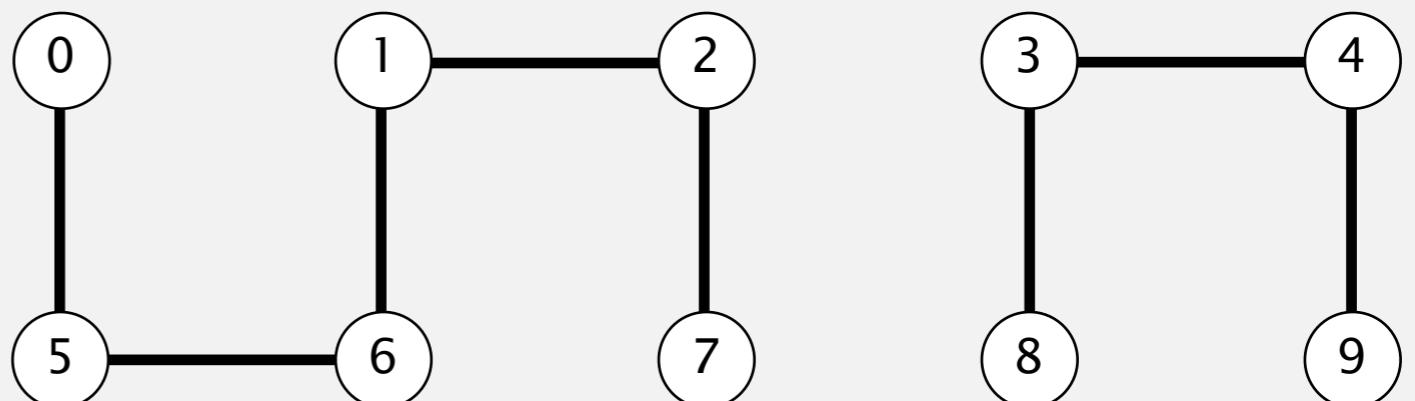
*are 8 and 9 connected?      ✓*

*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*

*connect 1 and 0*



# 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?      ✗*

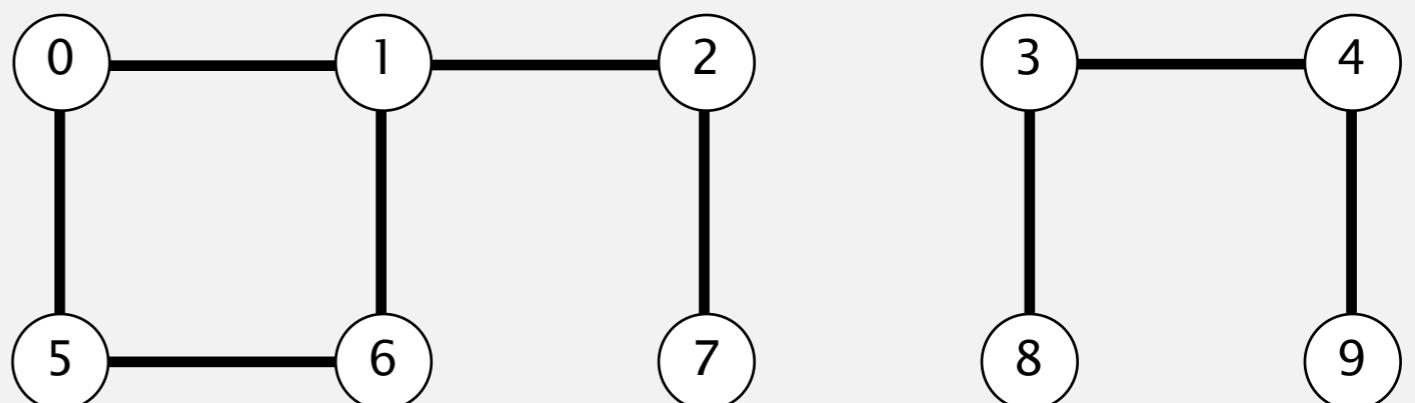
*are 8 and 9 connected?      ✓*

*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*

*connect 1 and 0*



# 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? ✓*

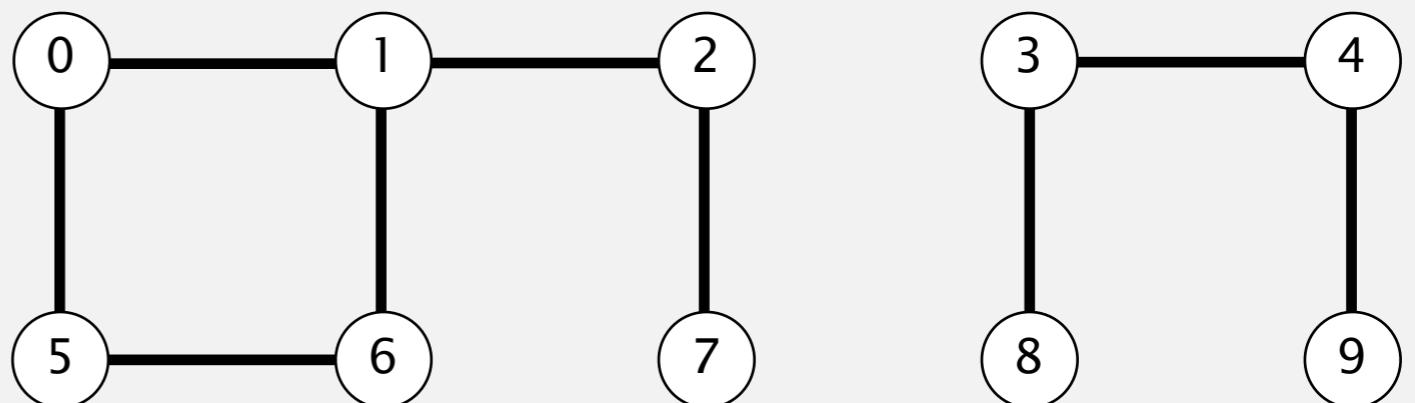
*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*

*connect 1 and 0*

*are 0 and 7 connected?*



# 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? ✓*

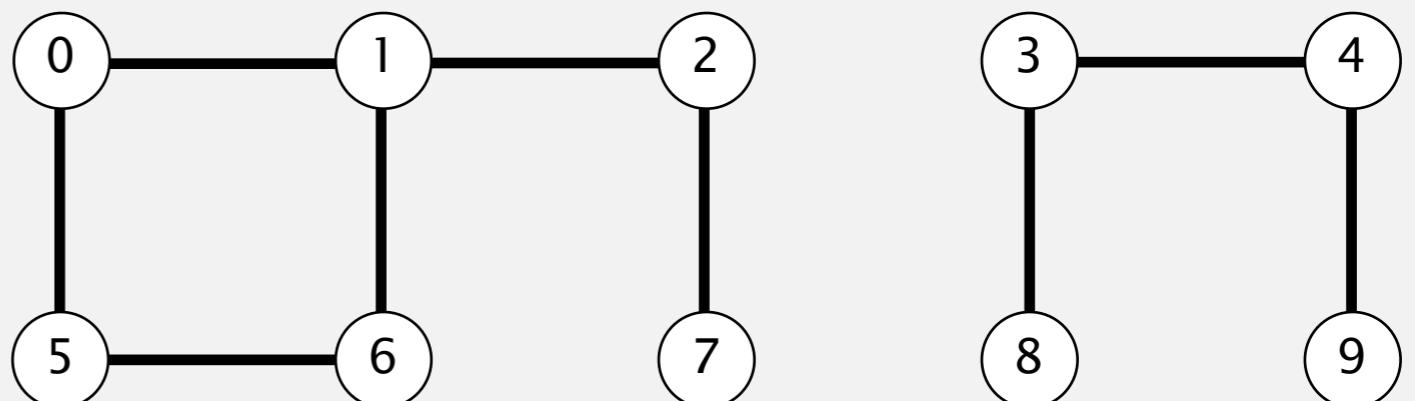
*connect 5 and 0*

*connect 7 and 2*

*connect 6 and 1*

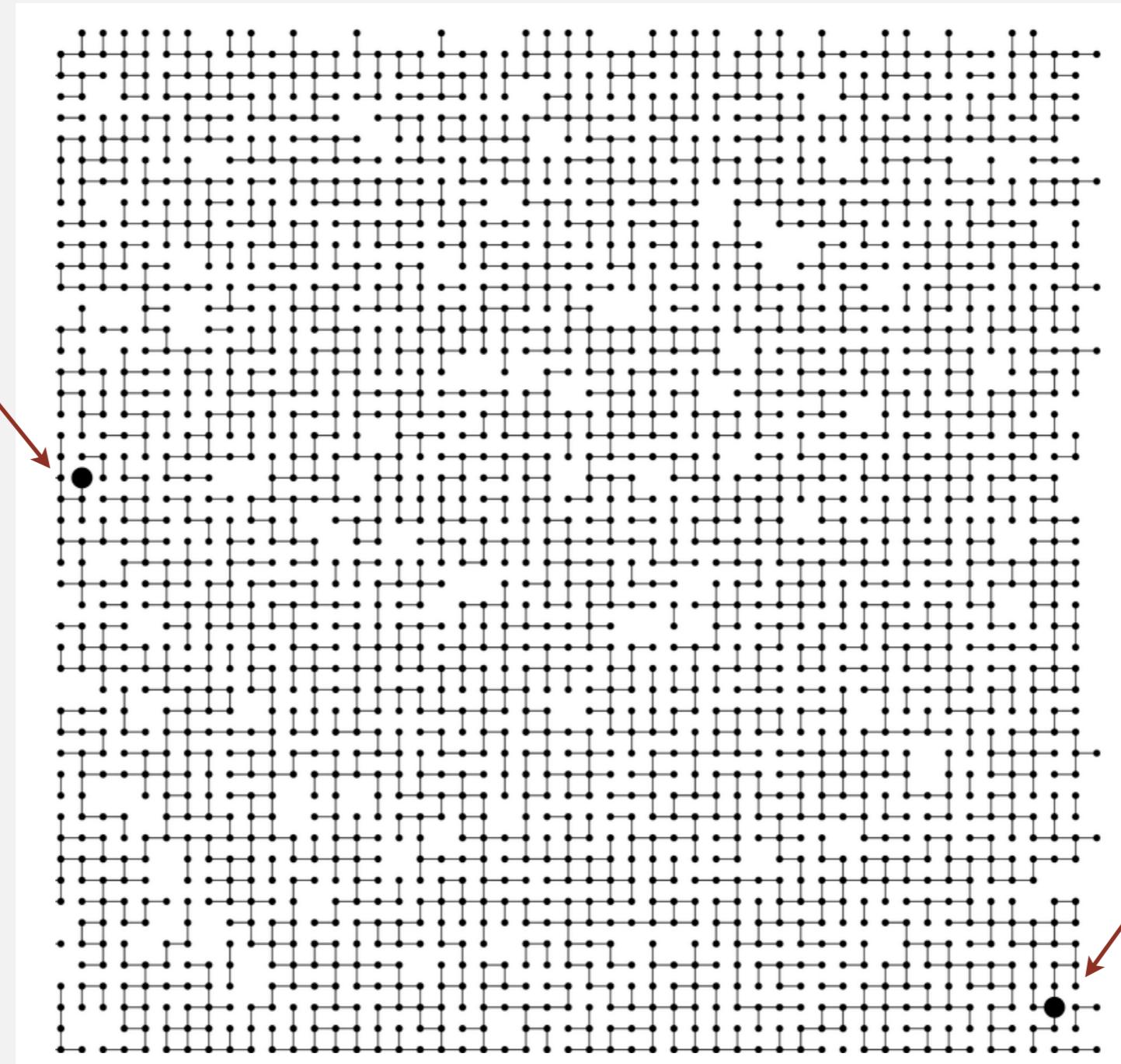
*connect 1 and 0*

*are 0 and 7 connected? ✓*



# A larger connectivity example

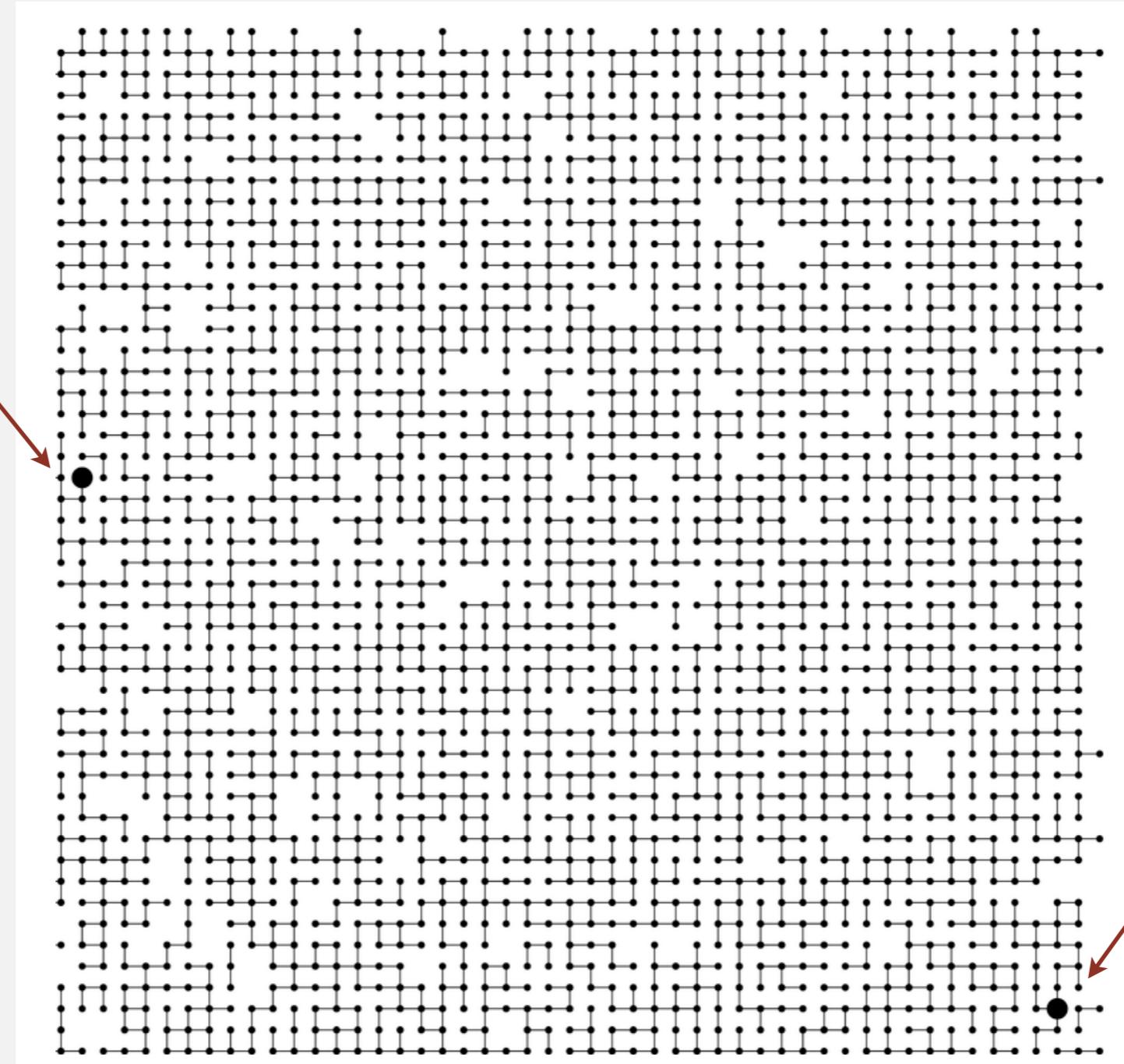
---



# A larger connectivity example

---

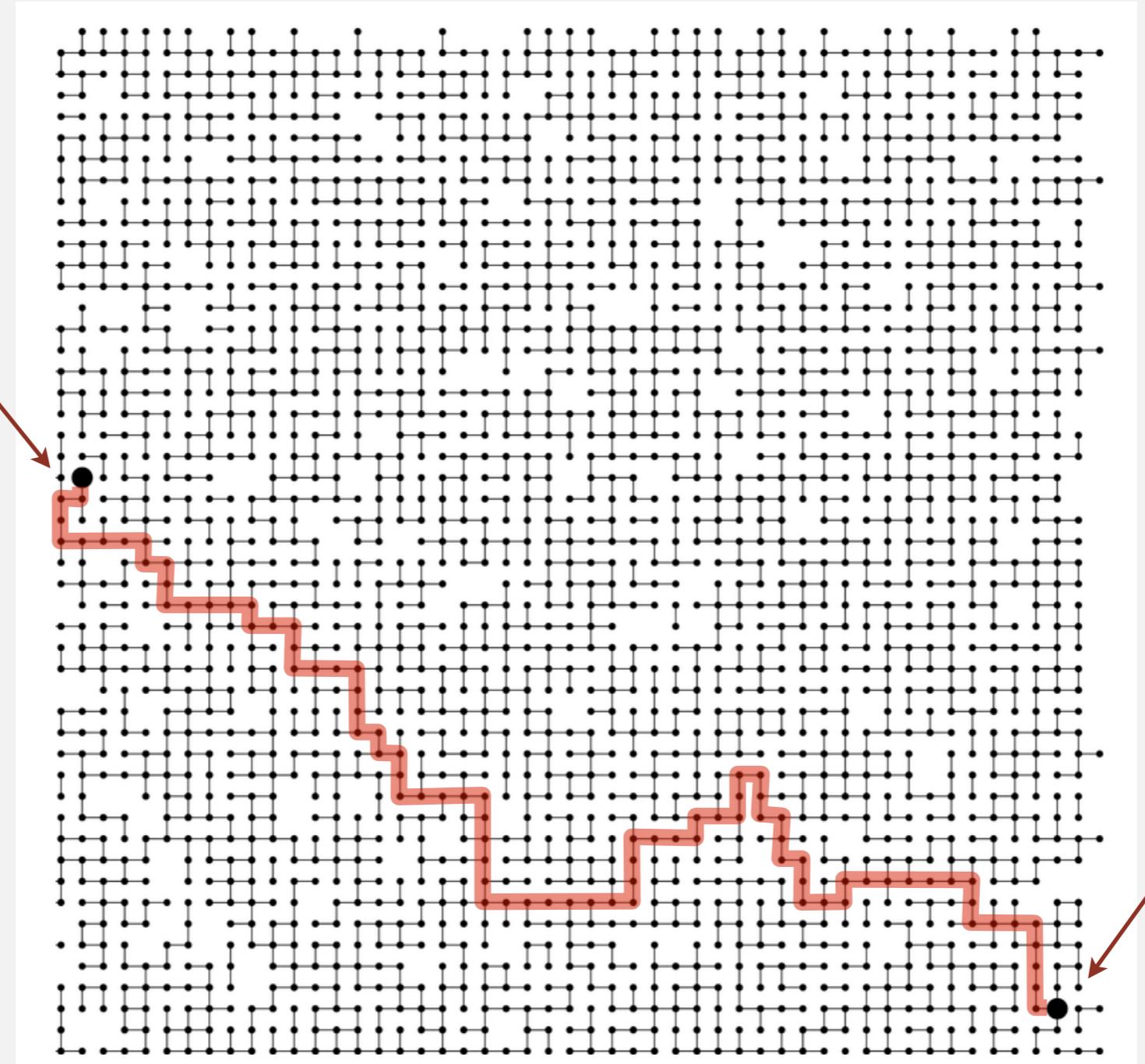
Q. Is there a path connecting  $p$  and  $q$  ?



# A larger connectivity example

---

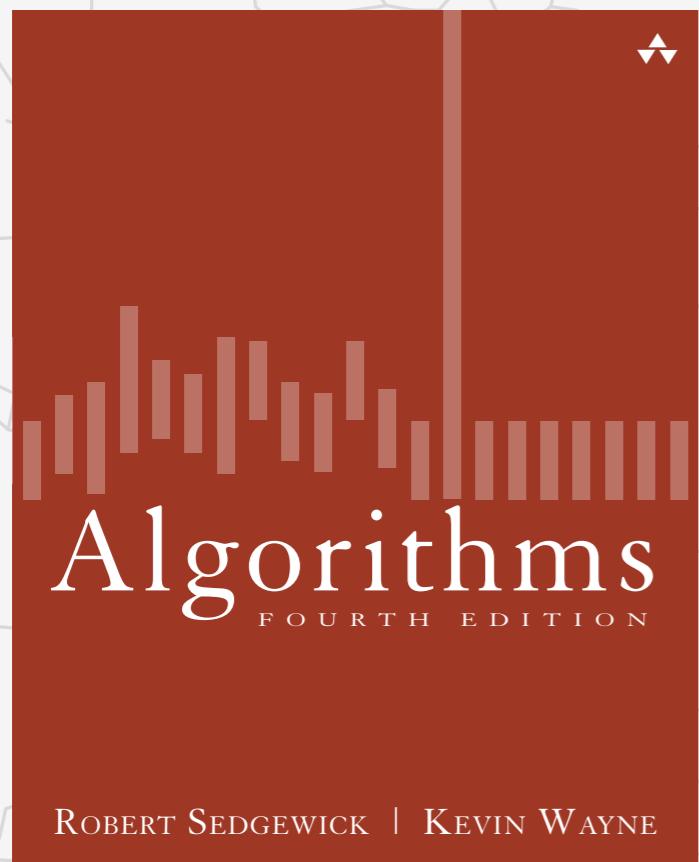
Q. Is there a path connecting  $p$  and  $q$  ?



A. Yes.

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

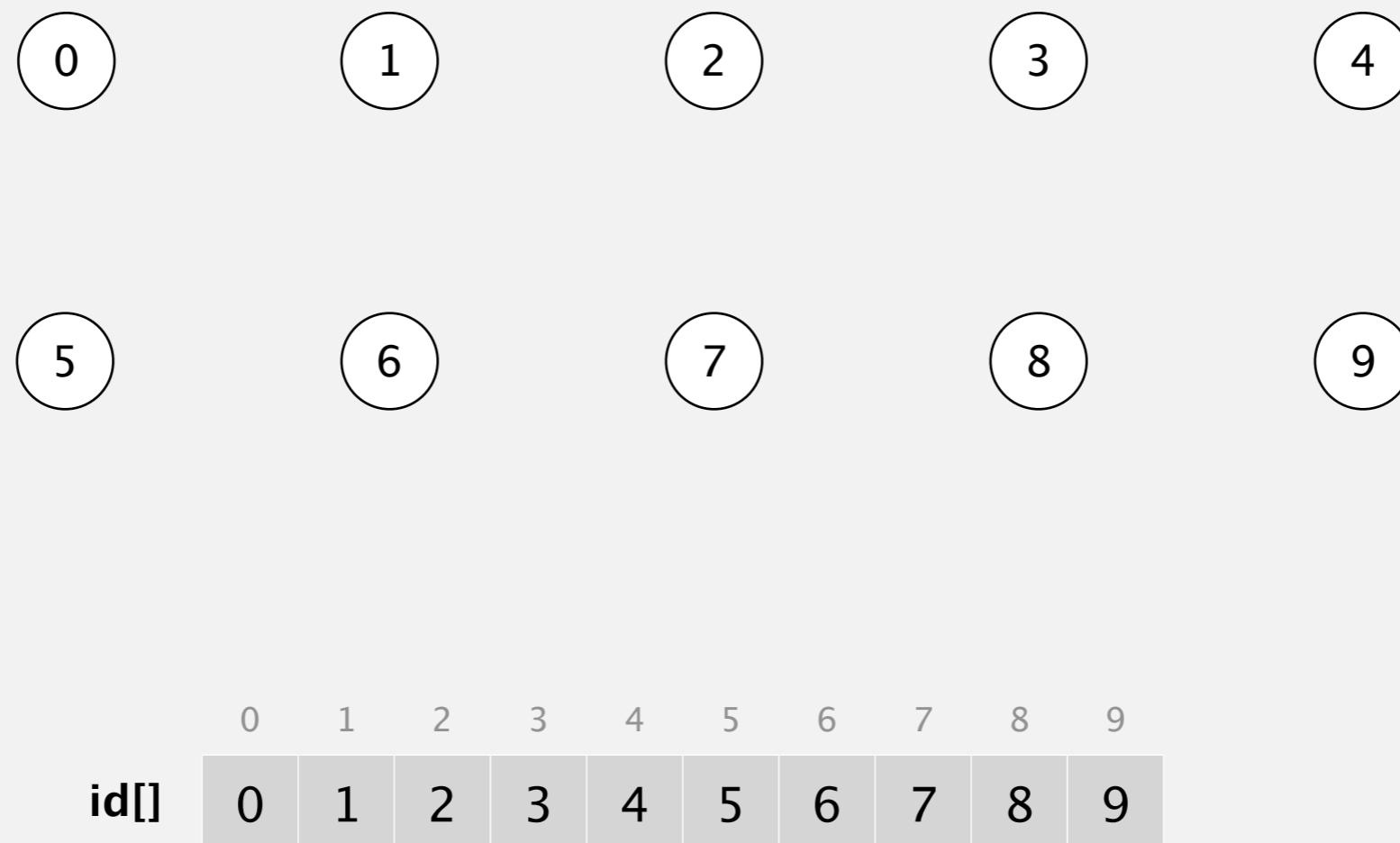


<http://algs4.cs.princeton.edu>

## 1.5 QUICK-FIND DEMO

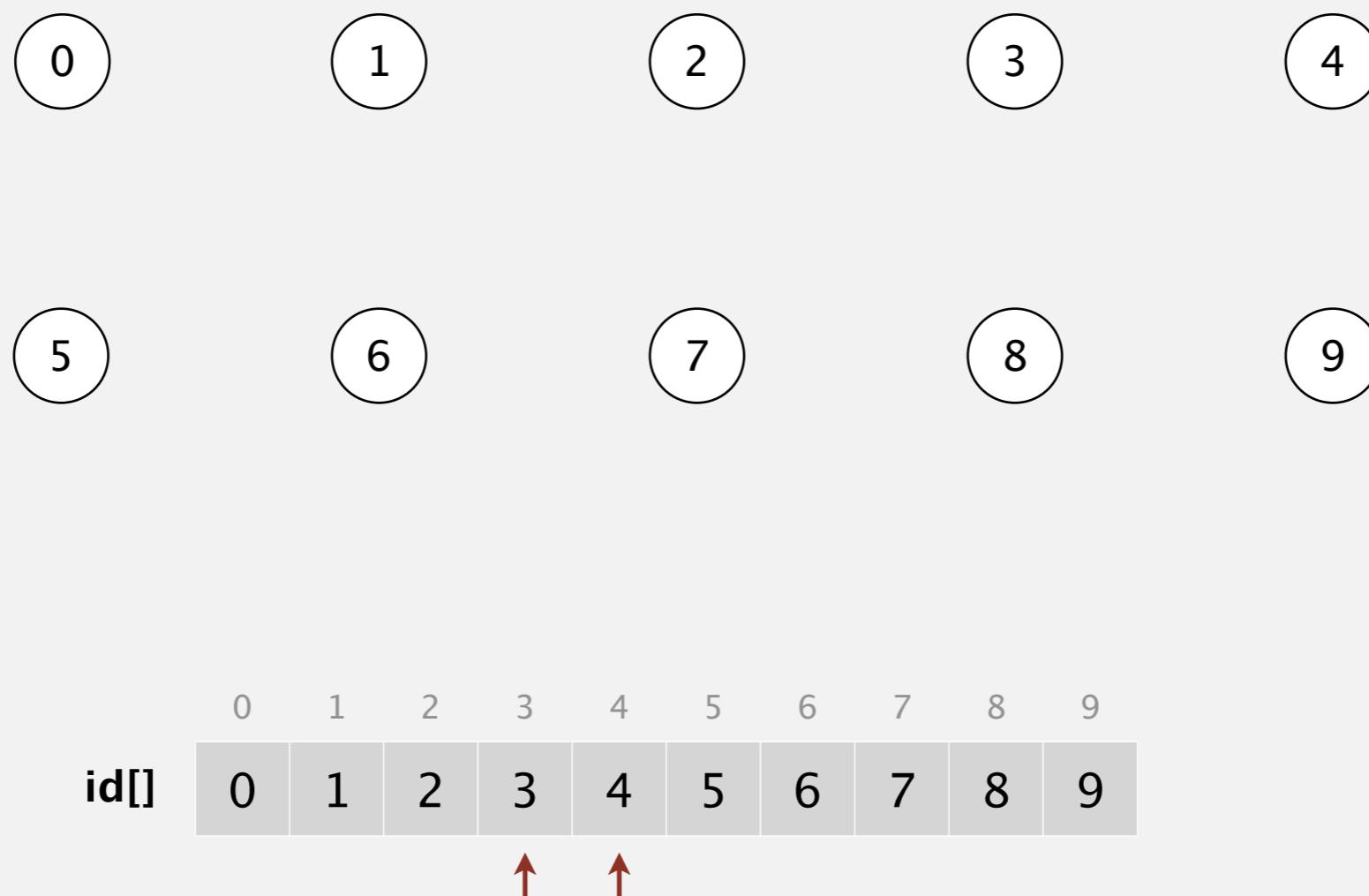
---

## Quick-find demo



## Quick-find demo

**union(4, 3)**



## Quick-find demo

**union(4, 3)**



A diagram illustrating a 1D array of size 10. The array is represented by a horizontal row of 10 boxes, each containing a black digit from 0 to 9. Above the array, the indices 0 through 9 are shown in gray. Two red arrows point to the elements at index 3 and index 4.

0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Quick-find demo

---

**union(4, 3)**



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

↑    ↑

# Quick-find demo

---



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	3	3	5	6	7	8	9

# Quick-find demo

---

**union(3, 8)**



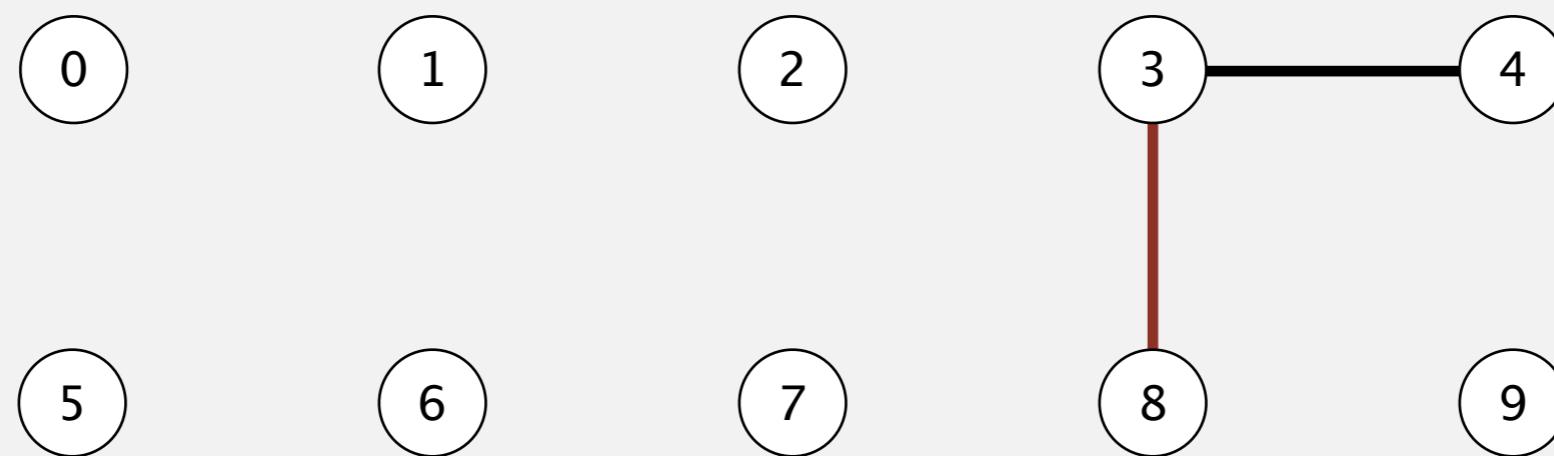
id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	3	5	6	7	8	9

Two red arrows point upwards from the value 3 in the id[3] cell to the bottom of the 3 in the id[8] cell, indicating that node 3 is being unioned with node 8.

# Quick-find demo

---

**union(3, 8)**

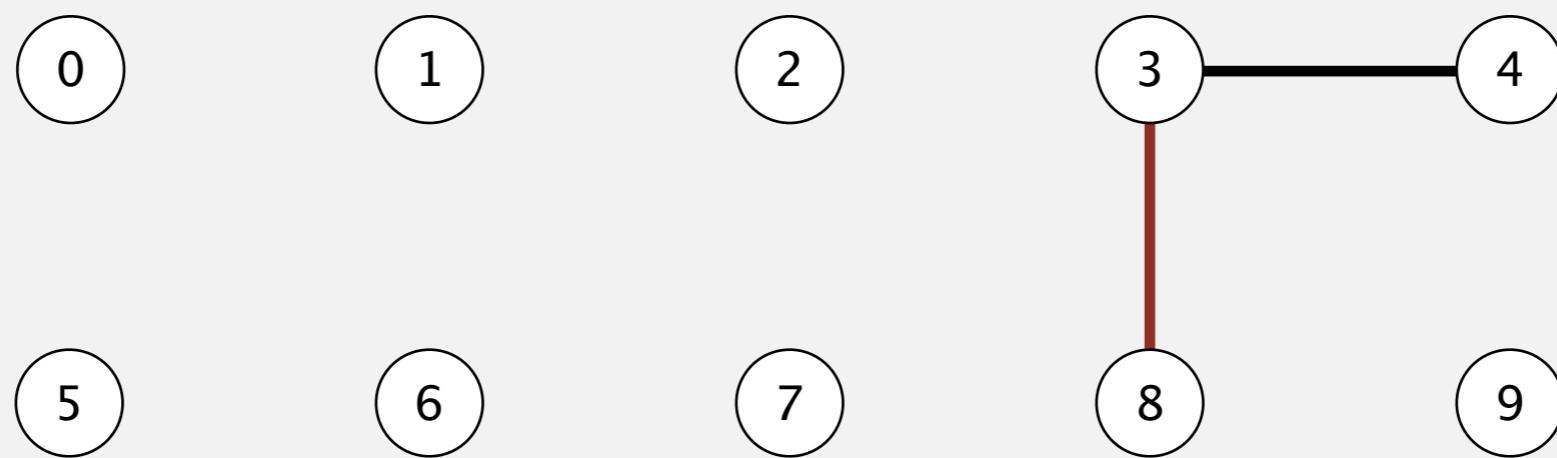


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	3	5	6	7	8	9

Two red arrows point to the '3' in the id[] array at indices 3 and 8, indicating the union operation is merging the components containing node 3 and node 8.

## Quick-find demo

**union(3, 8)**



A diagram illustrating a 1D array of integers. The array elements are shown in boxes below their corresponding indices:

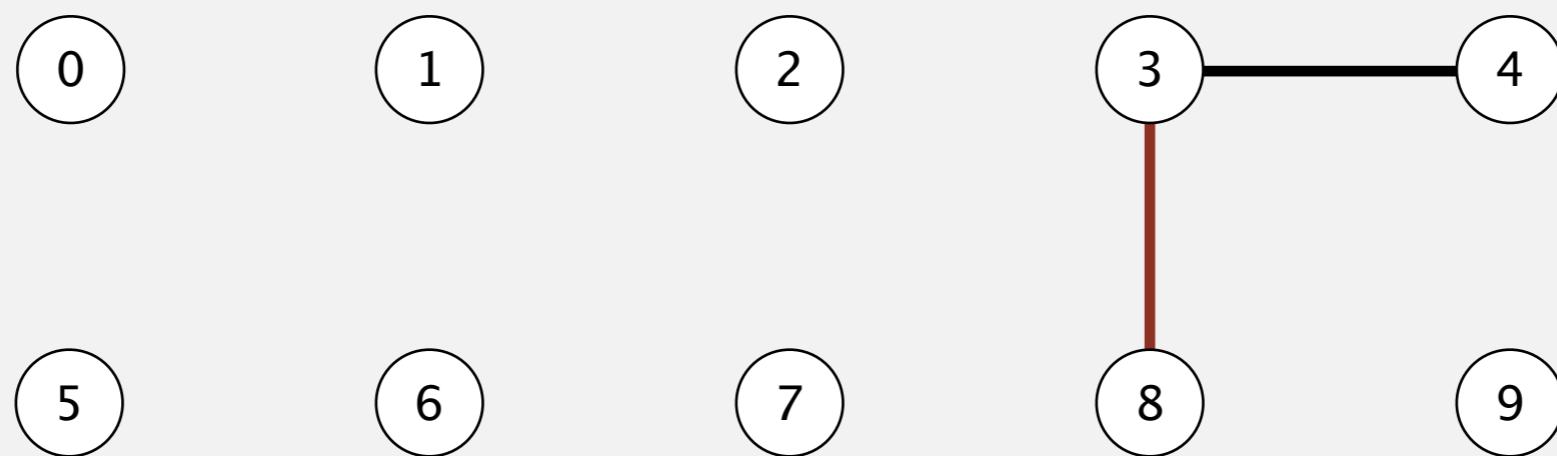
Index	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	<b>8</b>	3	5	6	7	8	9

An arrow points to the value 8 at index 3.

# Quick-find demo

---

**union(3, 8)**

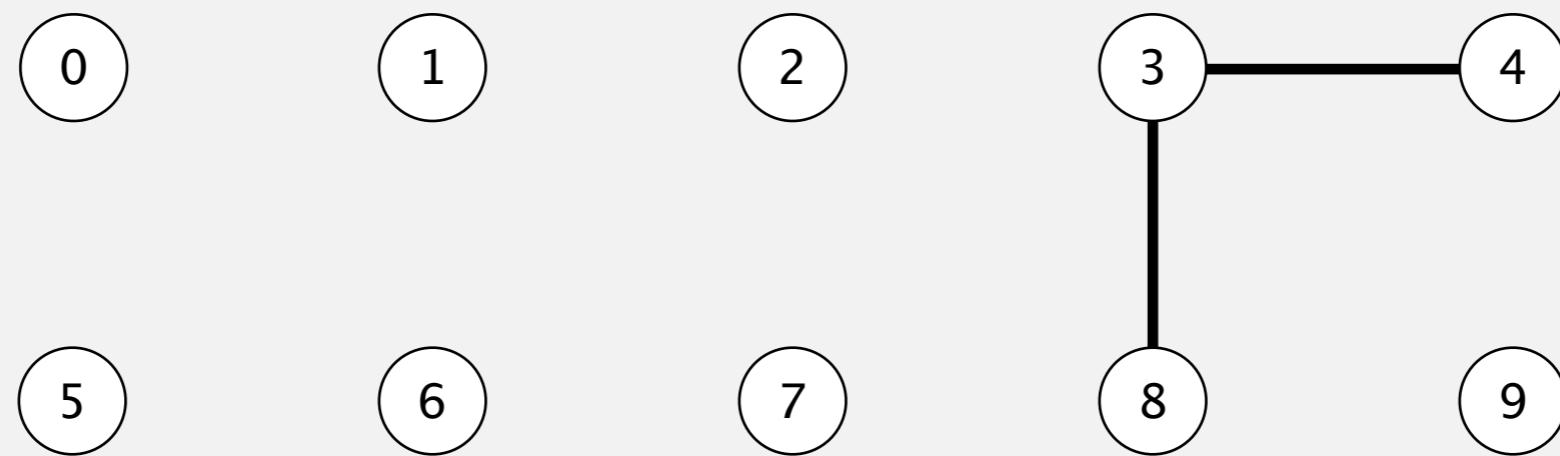


id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	8	5	6	7	8	9

Below the array, two red arrows point upwards from the value 8 in the 3rd index to the 8 in the 8th index, indicating the union operation.

# Quick-find demo

---

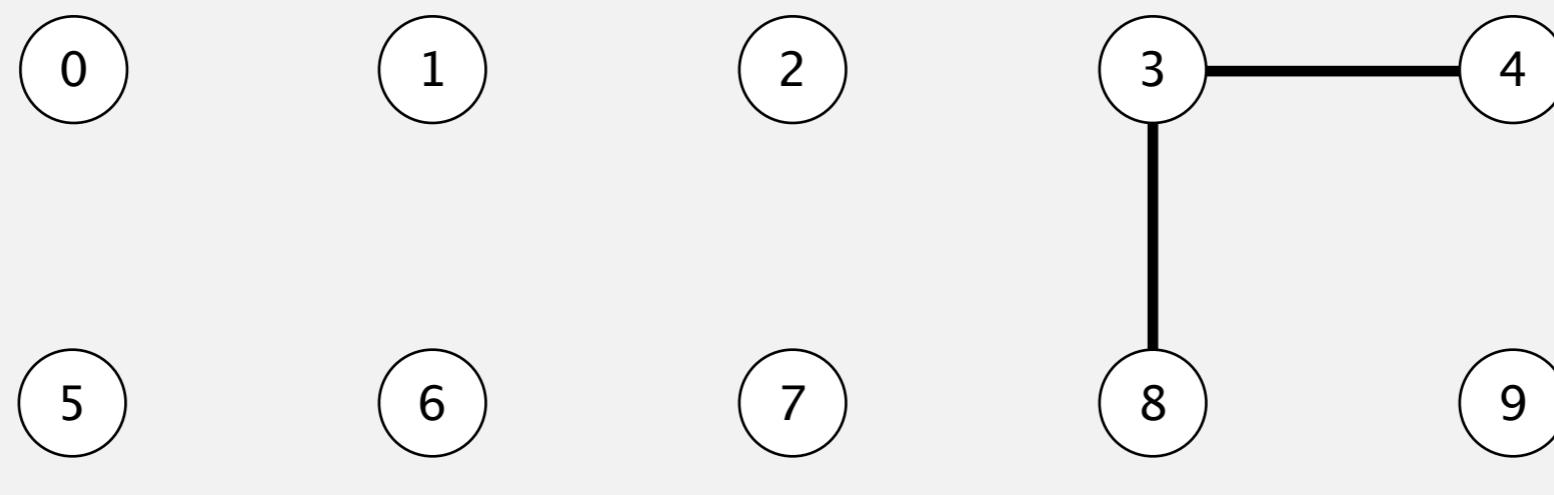


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	6	7	8	9

# Quick-find demo

---

**union(6, 5)**



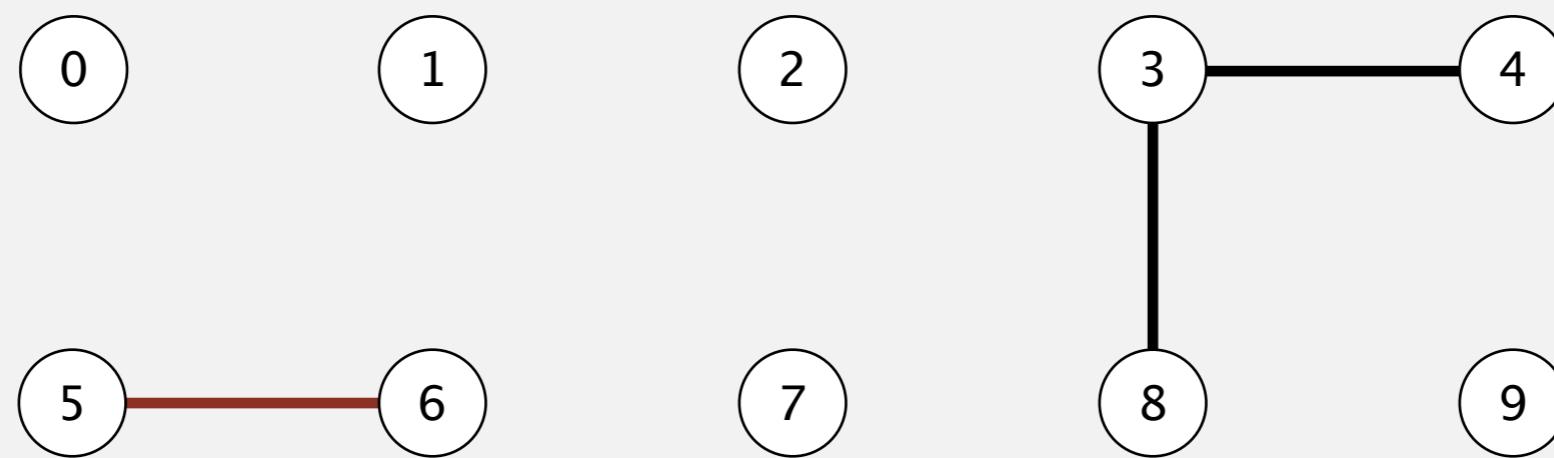
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	6	7	8	9

↑      ↑

# Quick-find demo

---

**union(6, 5)**



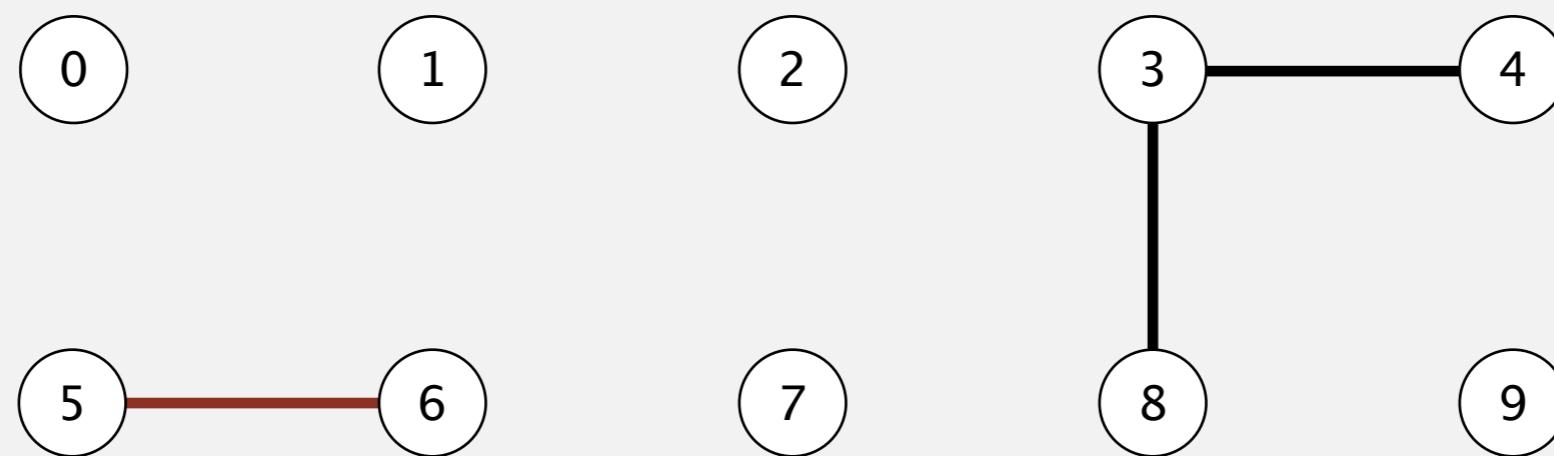
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	6	7	8	9

↑      ↑

# Quick-find demo

---

**union(6, 5)**

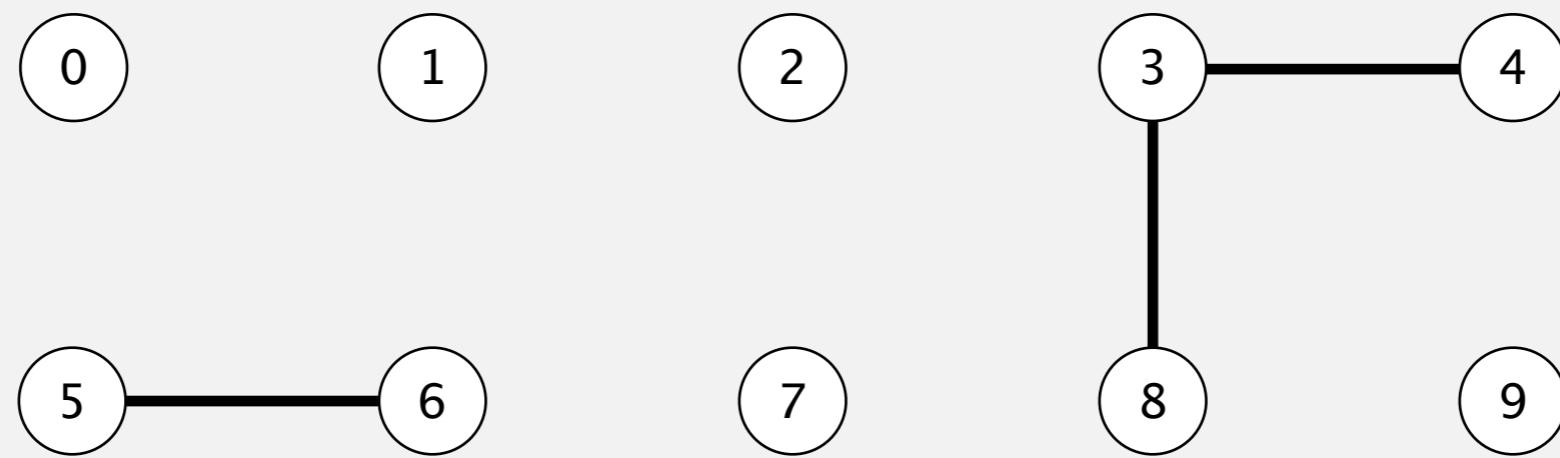


<b>id[]</b>	0	1	2	3	4	5	6	7	8	9
	0	1	2	8	8	5	5	7	8	9

↑      ↑

# Quick-find demo

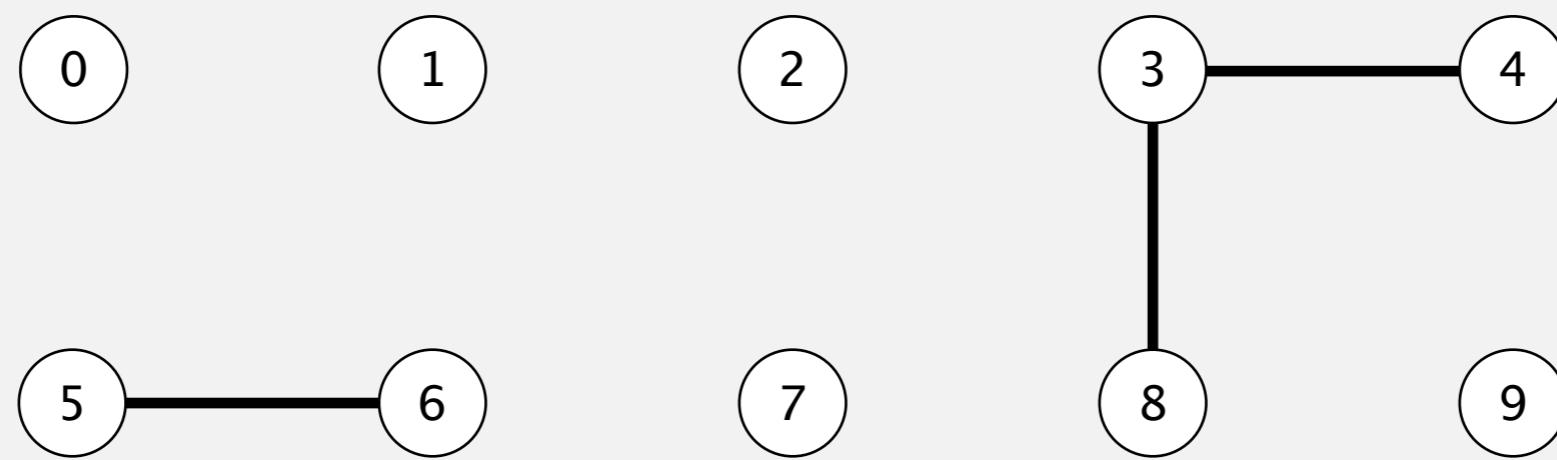
---



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

## Quick-find demo

**union(9, 4)**

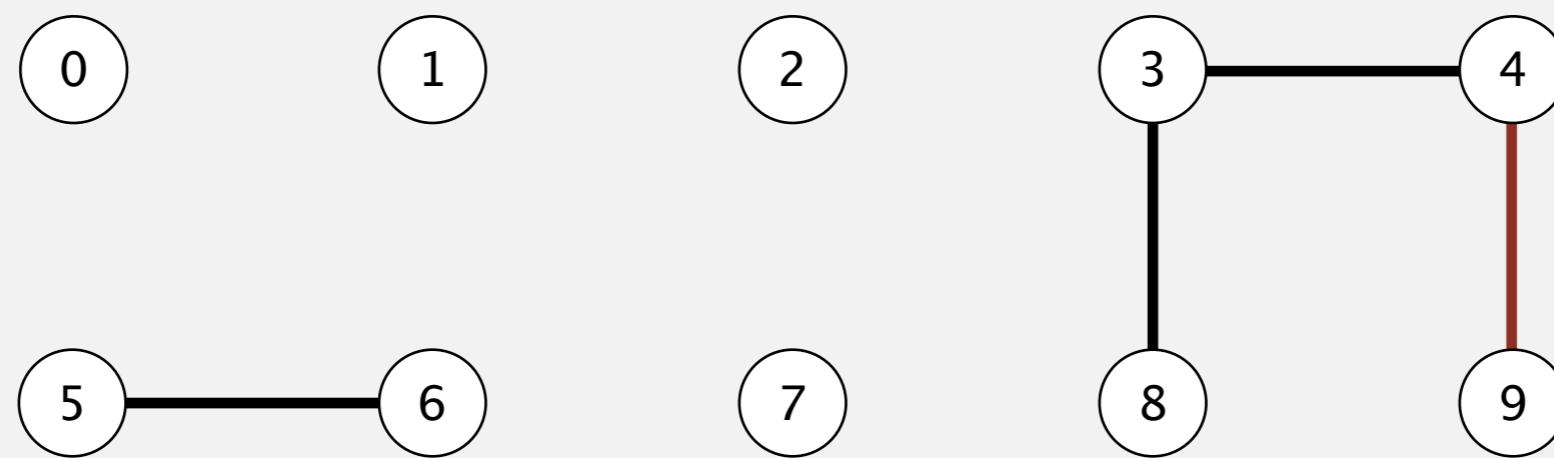


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	5	7	8	9

# Quick-find demo

---

**union(9, 4)**

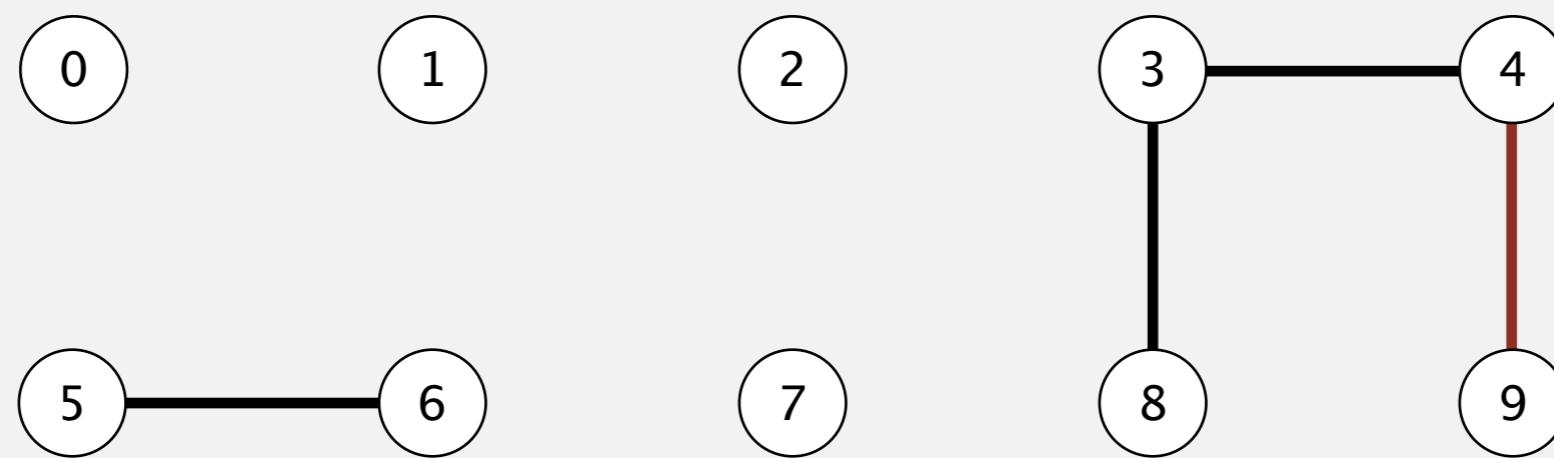


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	5	7	8	9

↑   ↑

## Quick-find demo

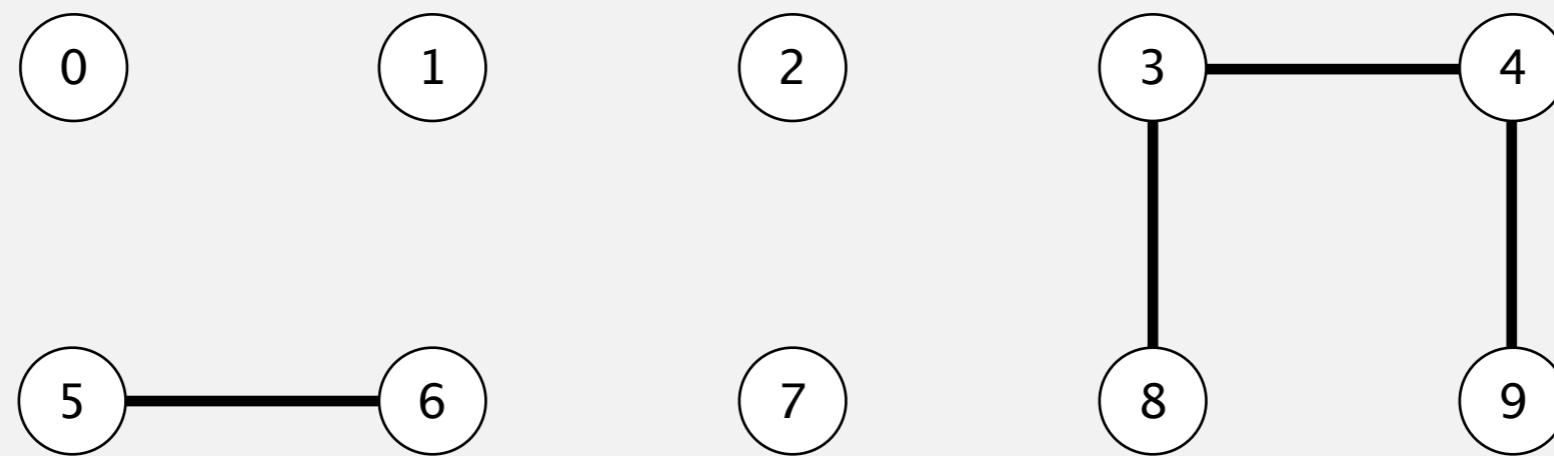
**union(9, 4)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	5	7	8	8

# Quick-find demo

---

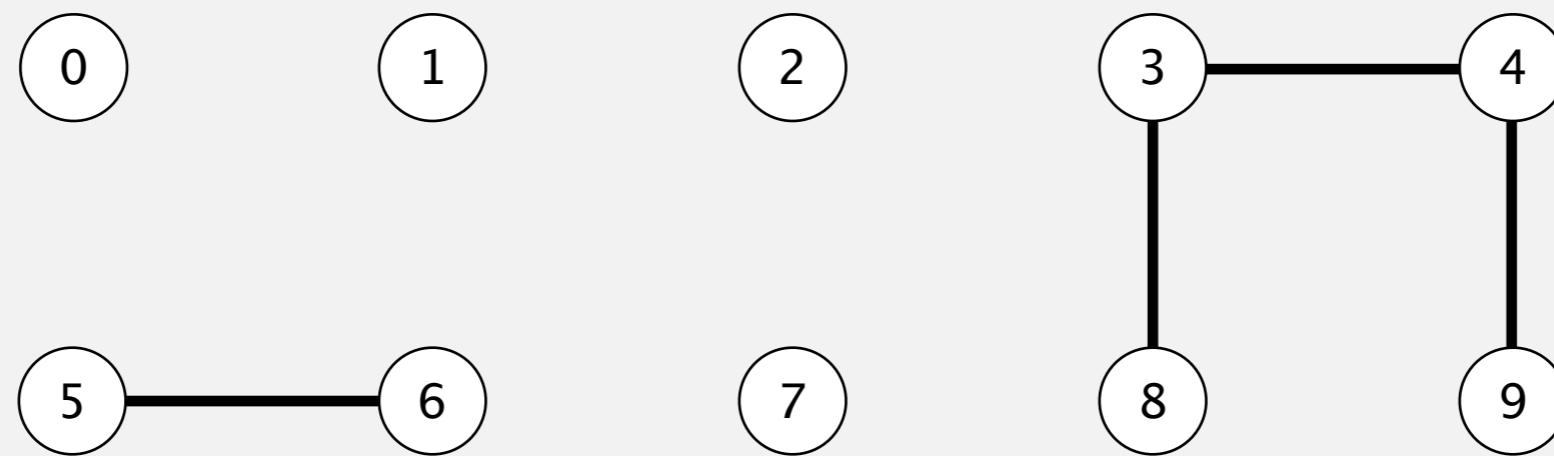


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	8	8	5	5	7	8	8

# Quick-find demo

---

**union(2, 1)**



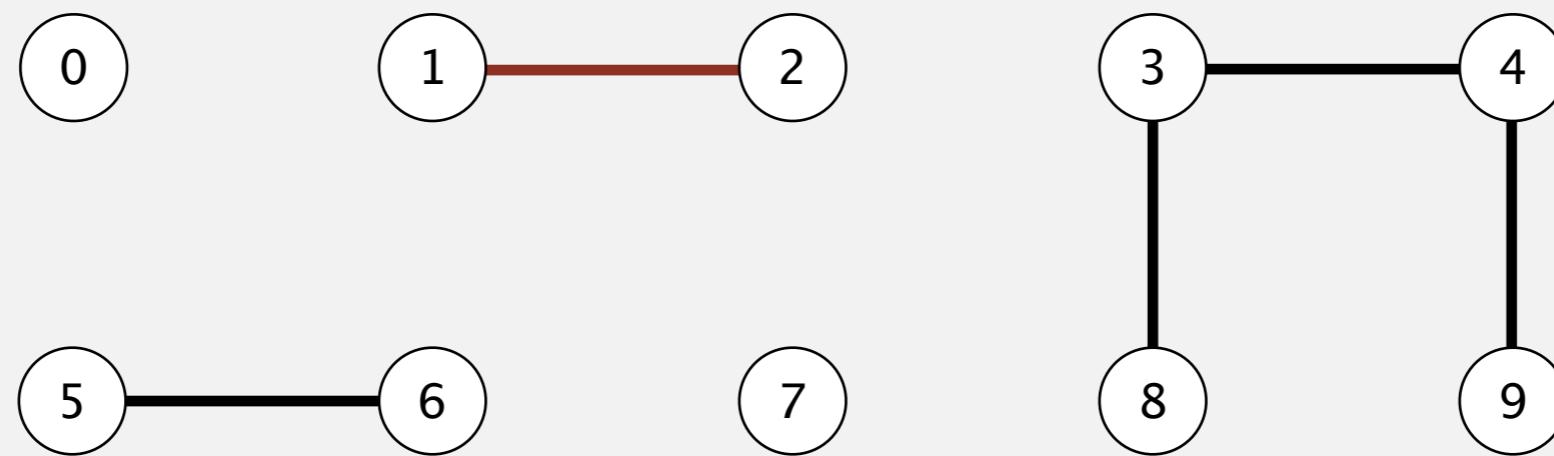
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	5	7	8	8

↑    ↑

# Quick-find demo

---

**union(2, 1)**



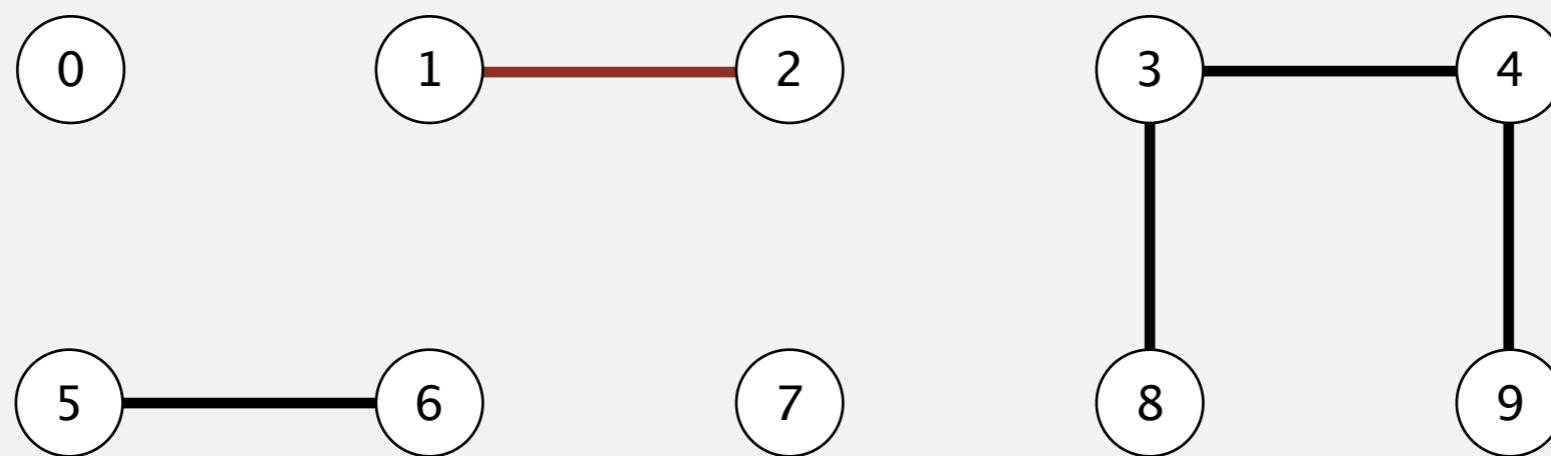
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	5	7	8	8

Two red arrows point to the '2' and '1' entries in the id[] array, indicating the nodes being unioned.

# Quick-find demo

---

**union(2, 1)**

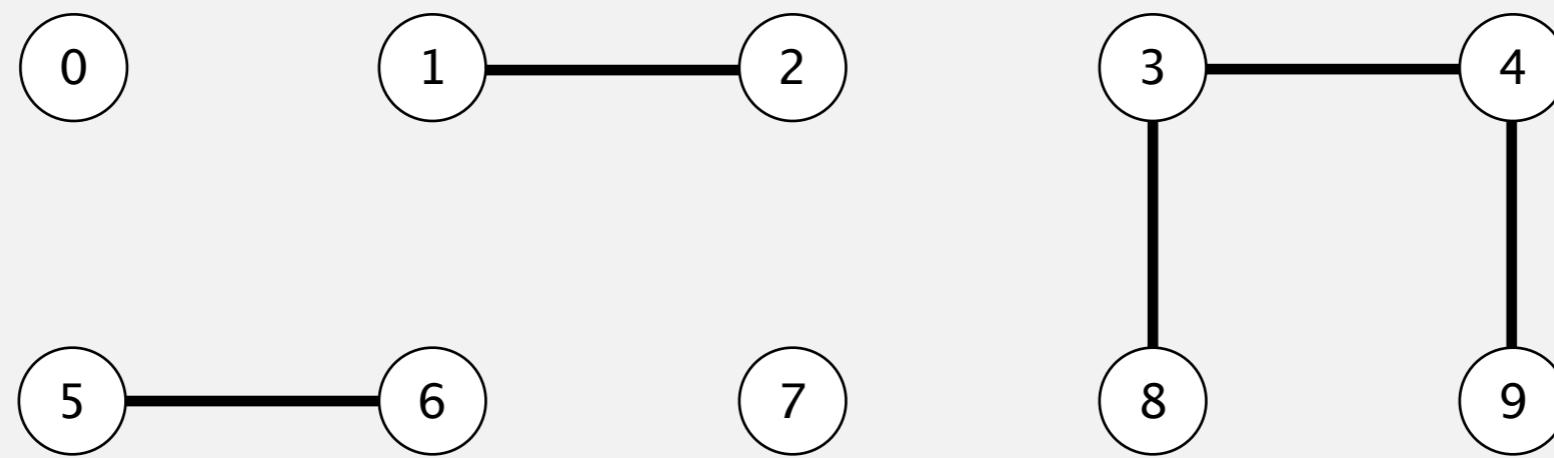


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

Two red arrows point to the second and third elements of the **id[]** array, specifically to the values 1 and 1.

# Quick-find demo

---

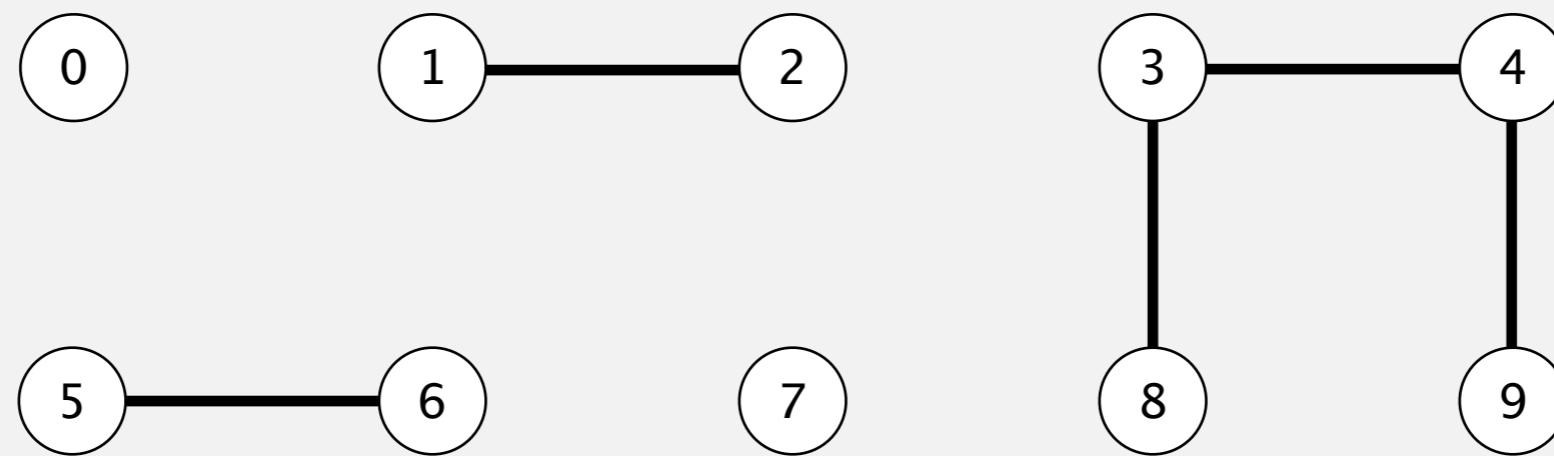


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

# Quick-find demo

---

**connected(8, 9)**



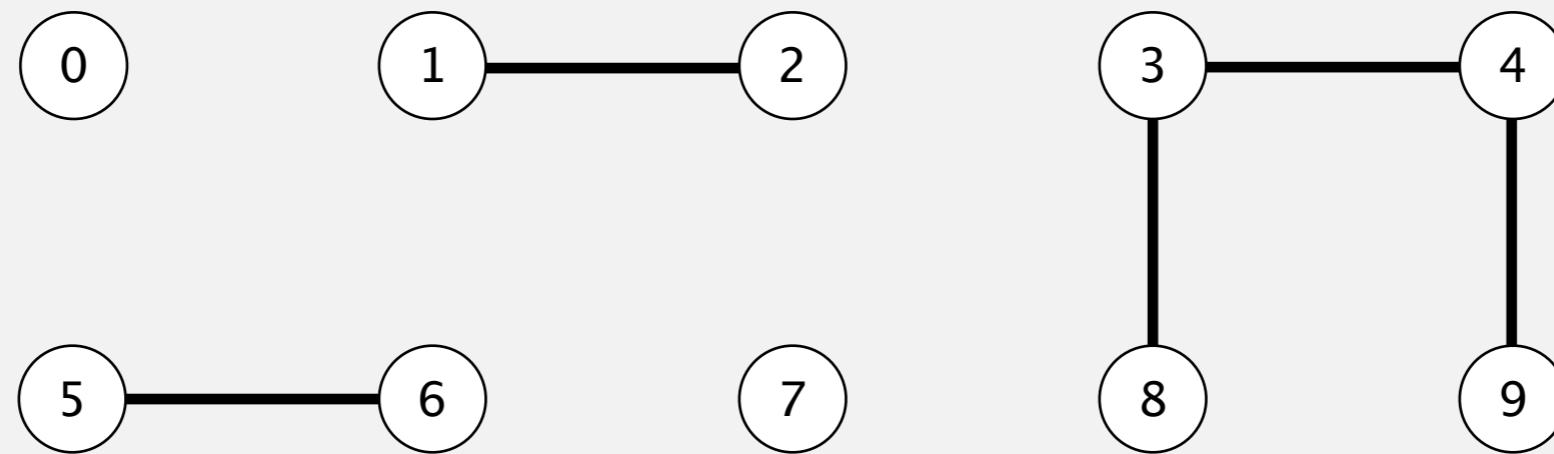
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

↑      ↑

# Quick-find demo

---

**connected(8, 9)**



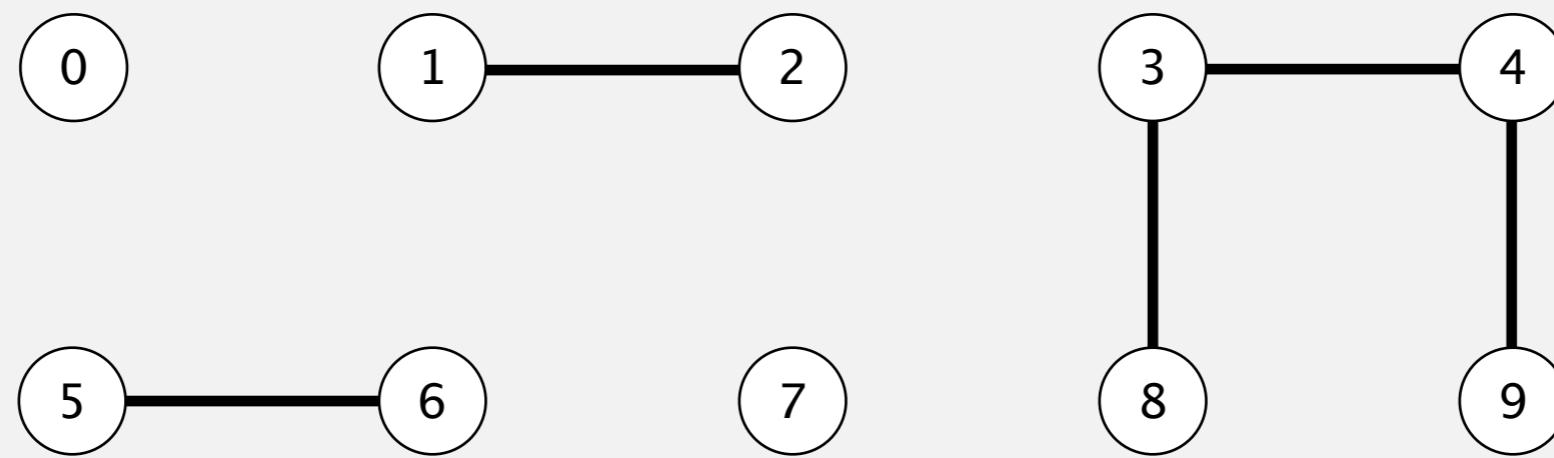
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

↑      ↑

**already connected**

# Quick-find demo

---

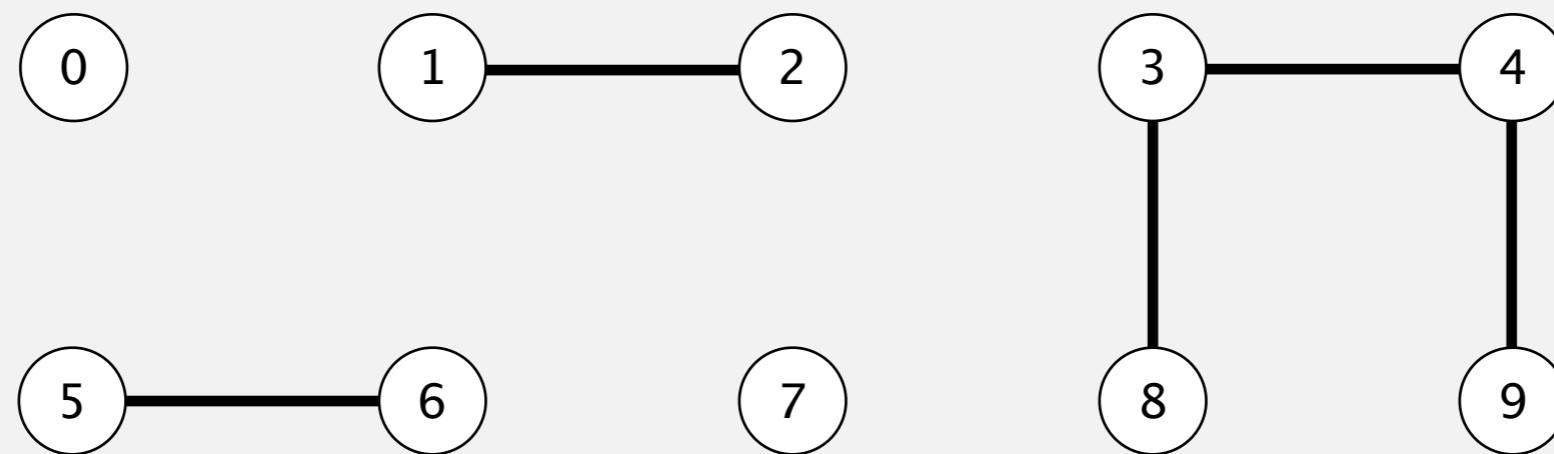


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

# Quick-find demo

---

**connected(5, 0)**



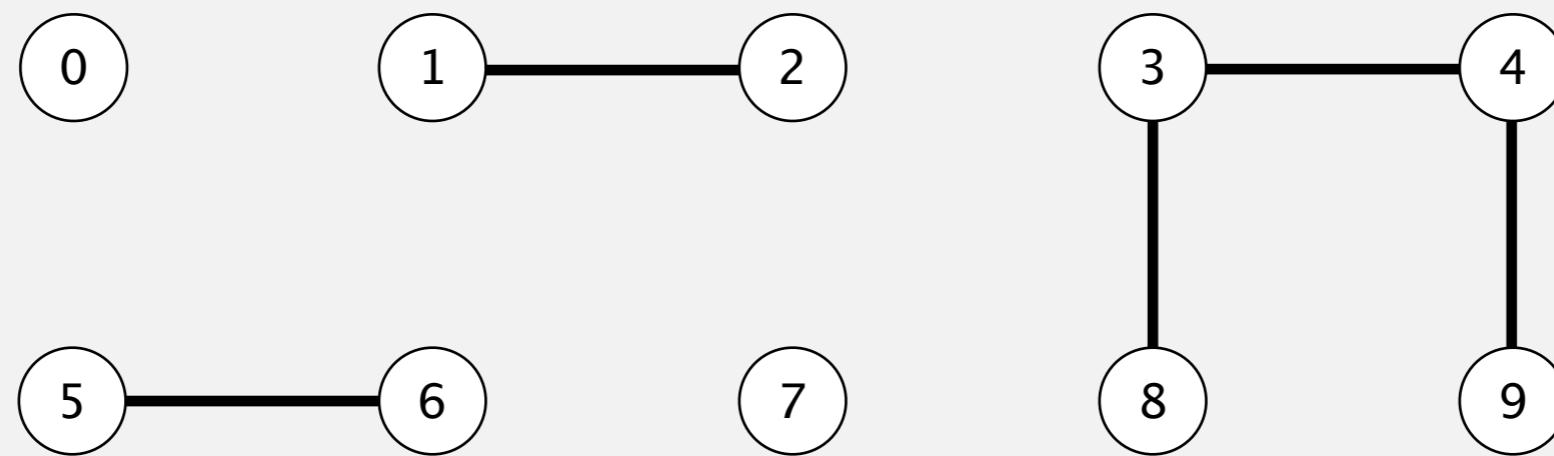
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

Two red arrows point to the 'id' array at index 1 and index 8, indicating the root nodes of components 1 and 8 respectively.

# Quick-find demo

---

**connected(5, 0)**

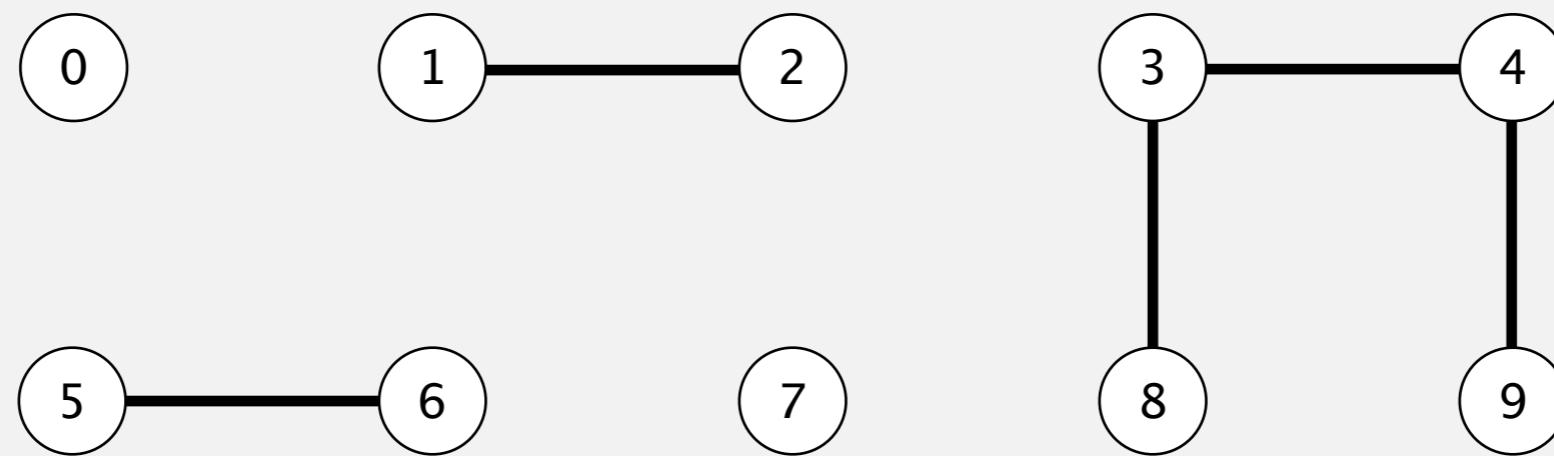


0	1	2	3	4	5	6	7	8	9	
id[]	0	1	1	8	8	5	5	7	8	8
	↑			↑						

not connected

# Quick-find demo

---

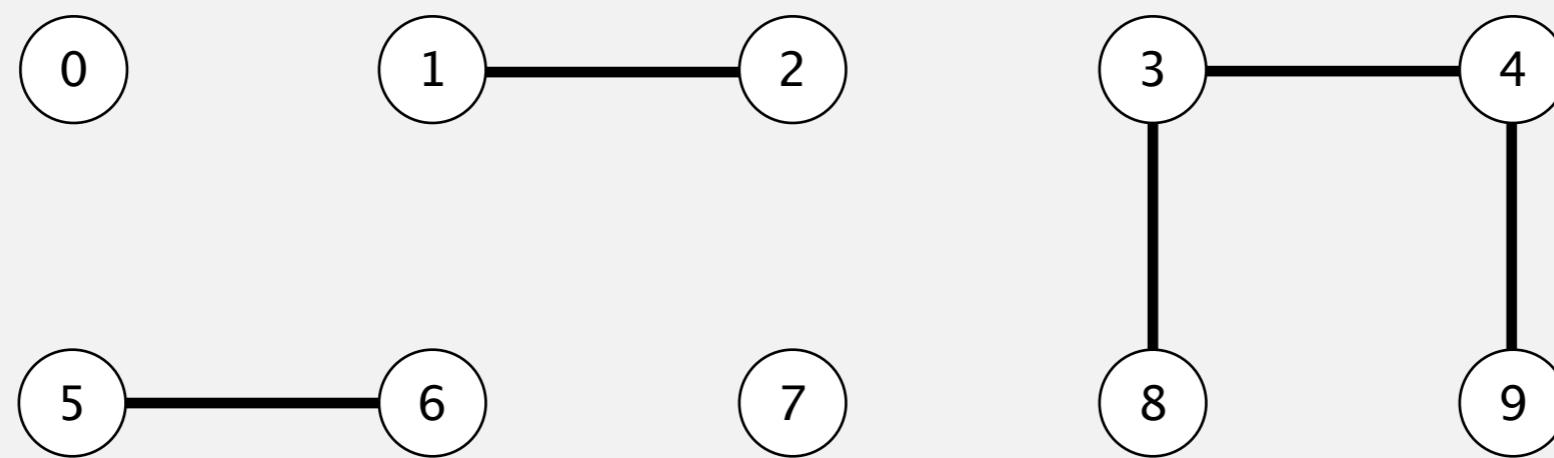


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

# Quick-find demo

---

**union(5, 0)**

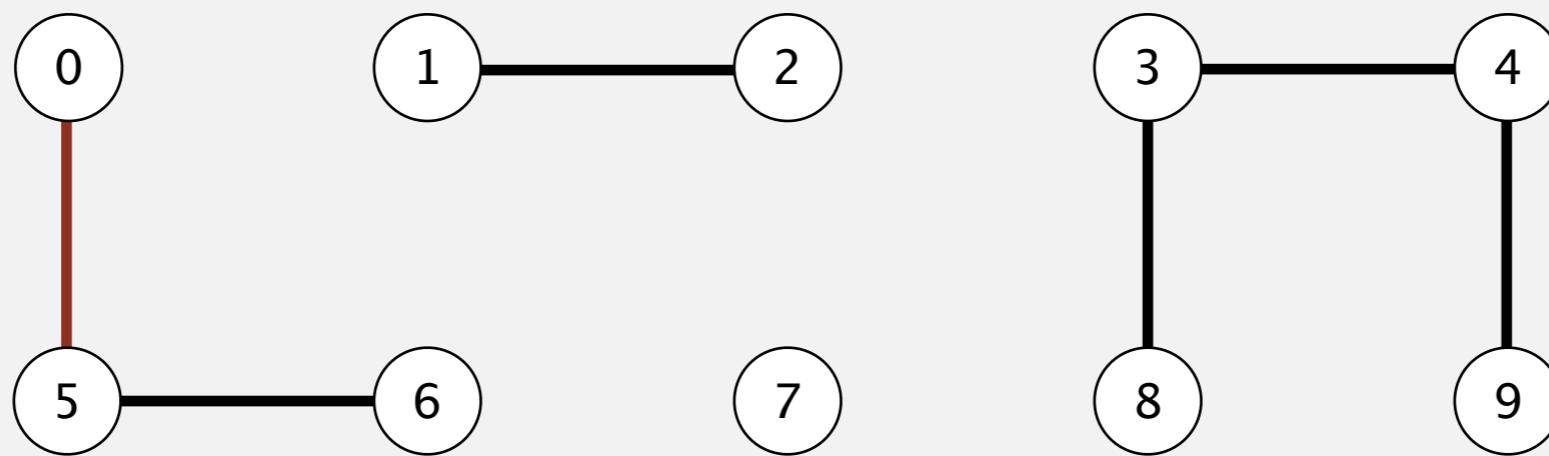


id[]	0	1	1	8	8	5	5	7	8	8
↑				↑						

# Quick-find demo

---

**union(5, 0)**



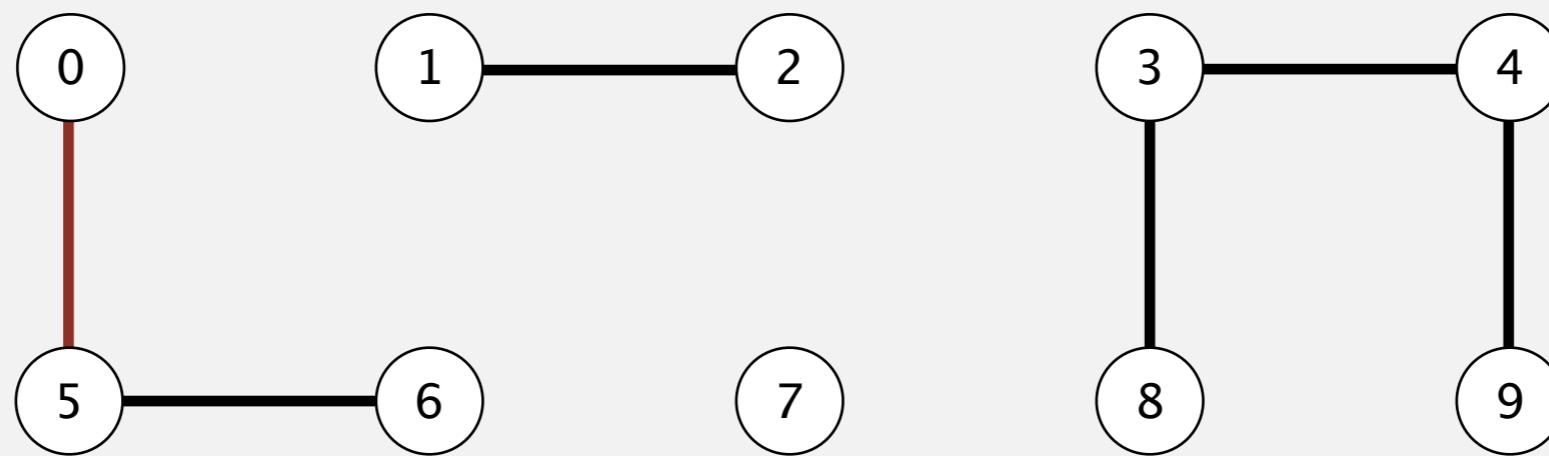
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

↑                           ↑

# Quick-find demo

---

**union(5, 0)**



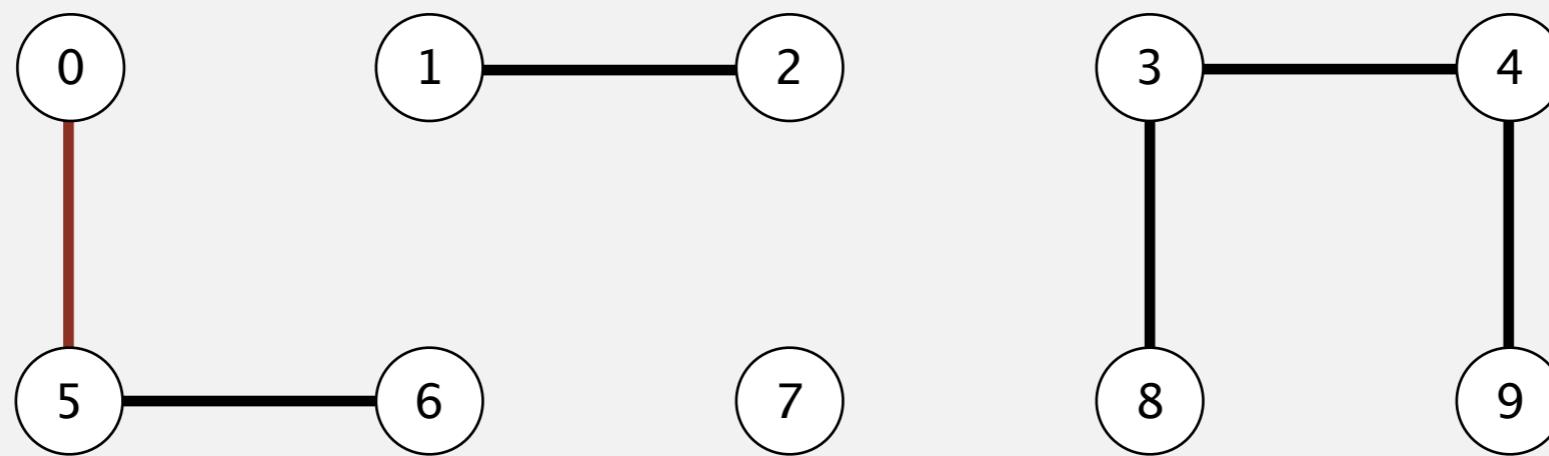
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	5	7	8	8

↑                           ↑

# Quick-find demo

---

**union(5, 0)**

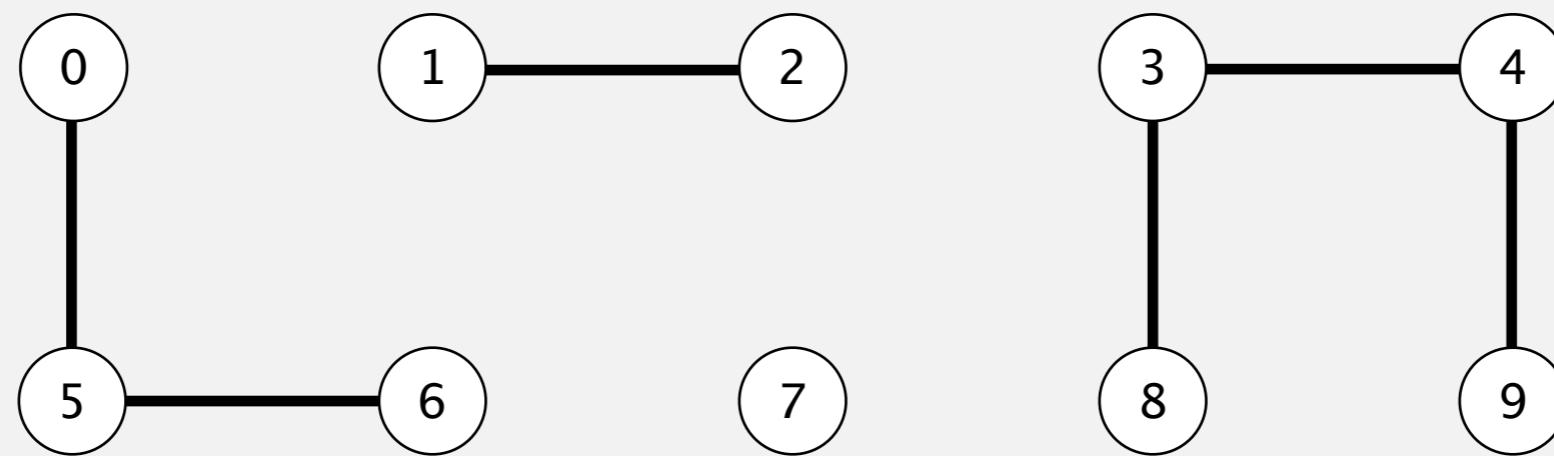


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	7	8	8

↑                           ↑

# Quick-find demo

---

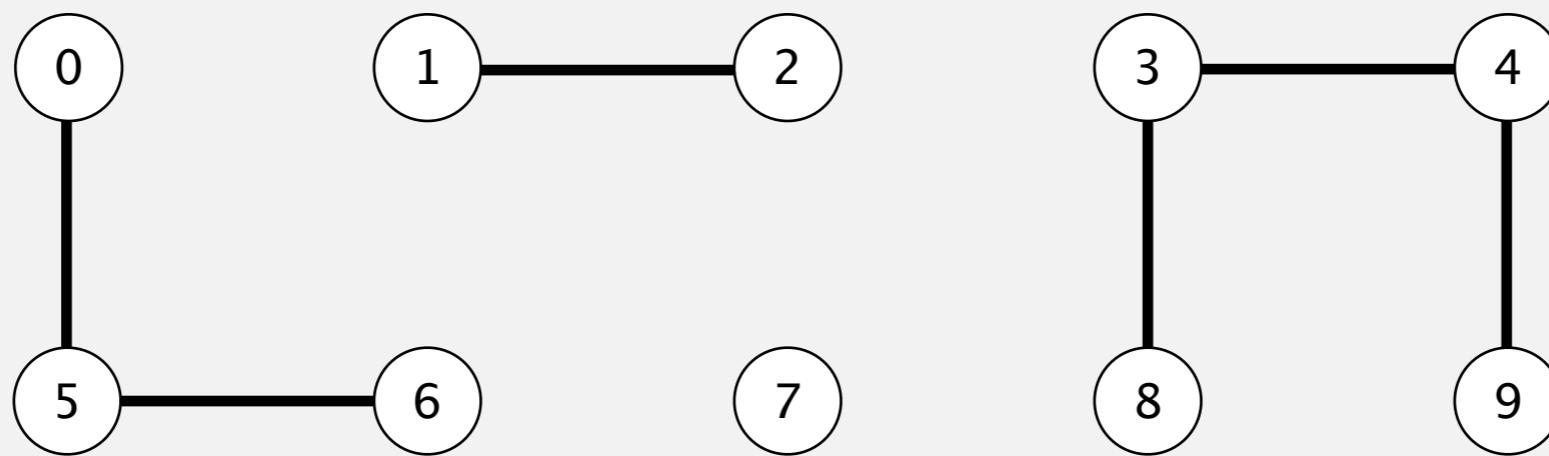


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	7	8	8

# Quick-find demo

---

**union(7, 2)**

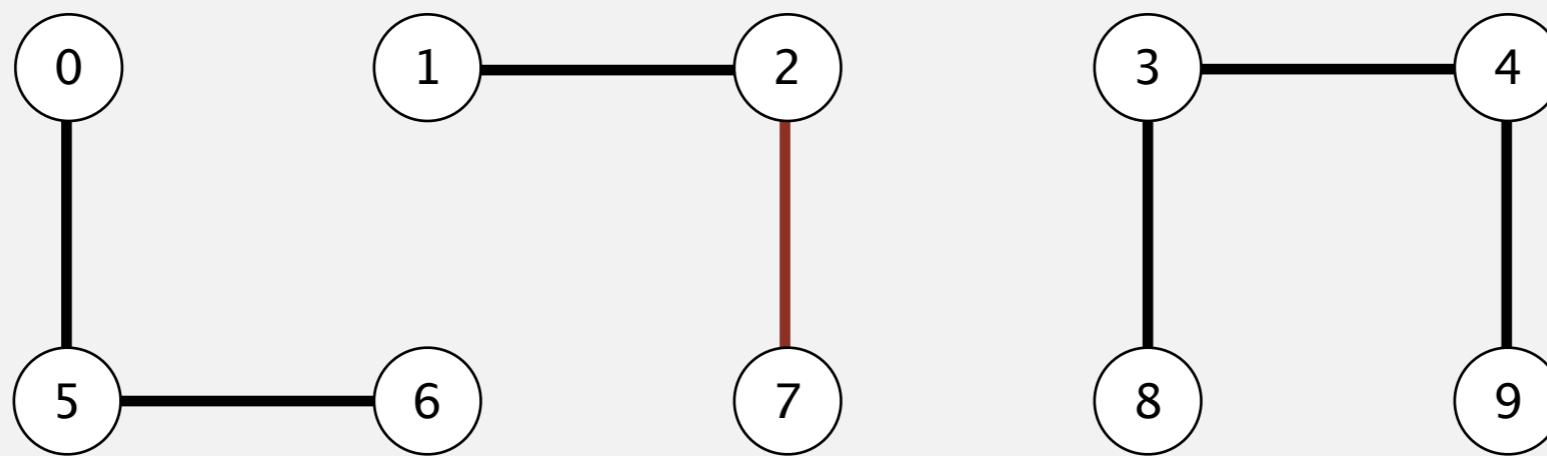


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	7	8	8
			↑					↑		

# Quick-find demo

---

**union(7, 2)**

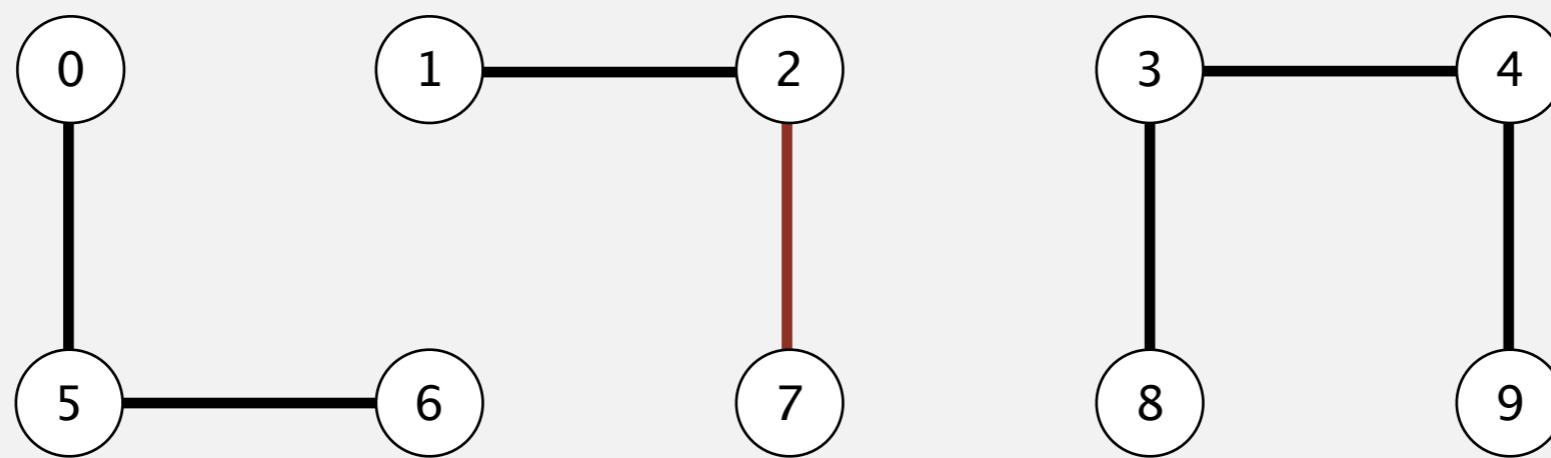


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	7	8	8

↑                           ↑

## Quick-find demo

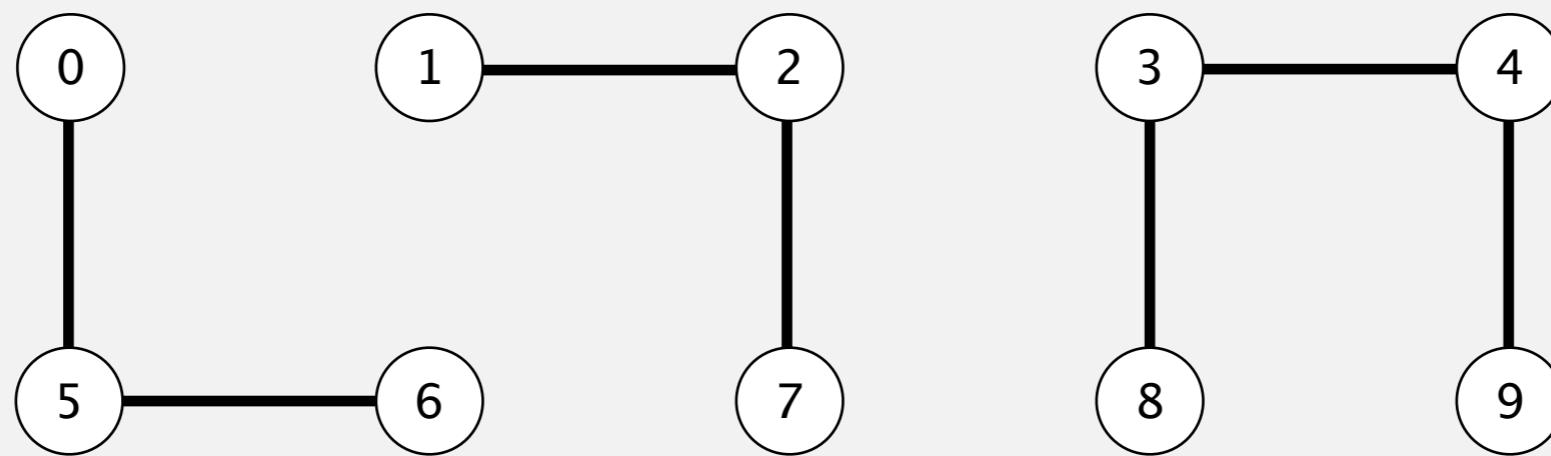
**union(7, 2)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	<b>1</b>	8	8

# Quick-find demo

---

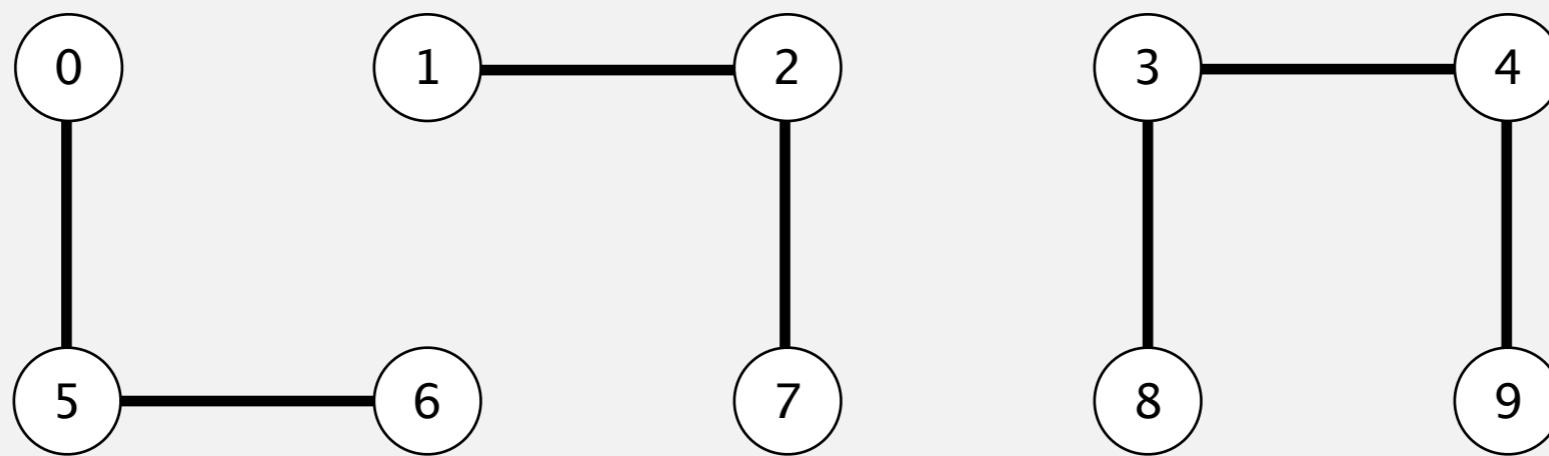


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	1	8	8

# Quick-find demo

---

**union(6, 1)**



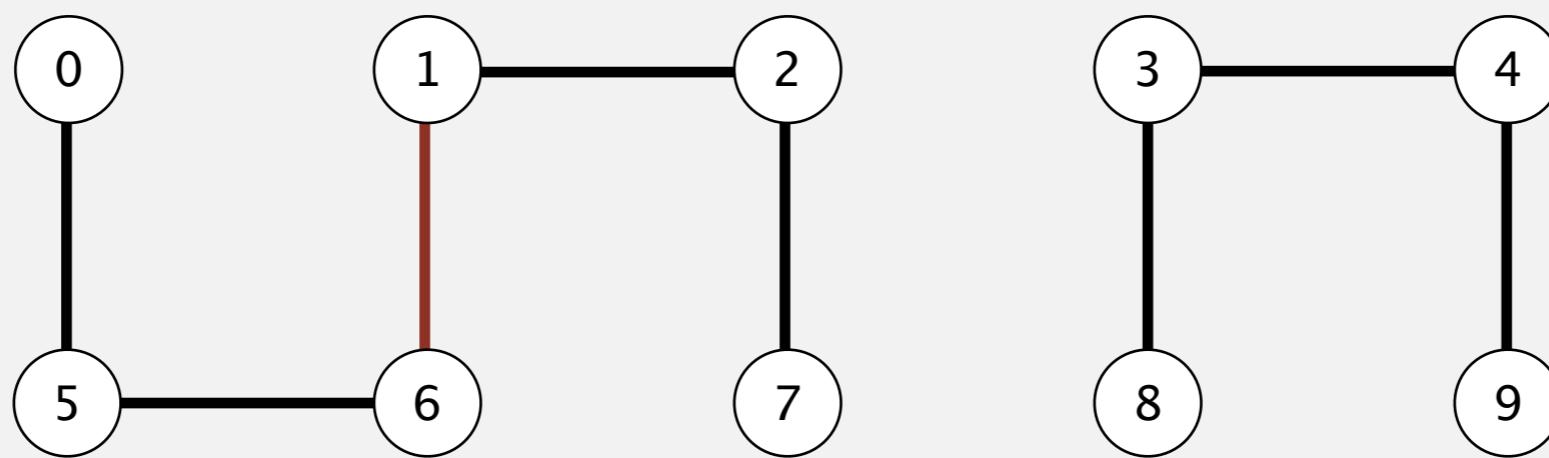
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	1	8	8

↑                                   ↑

# Quick-find demo

---

**union(6, 1)**



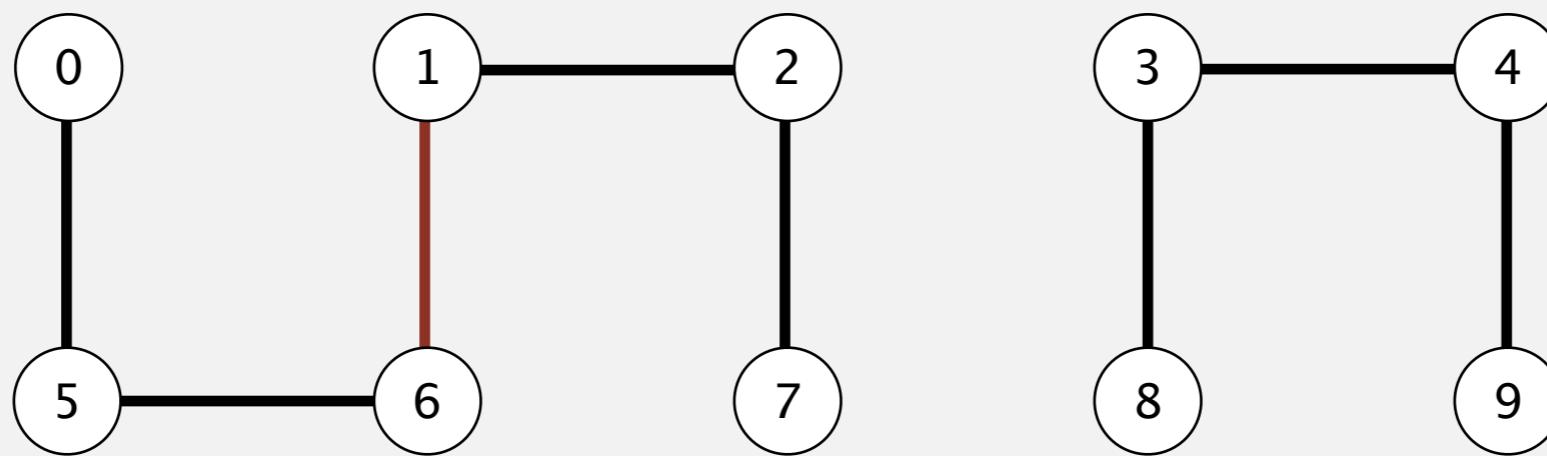
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	1	8	8

↑                              ↑

# Quick-find demo

---

**union(6, 1)**



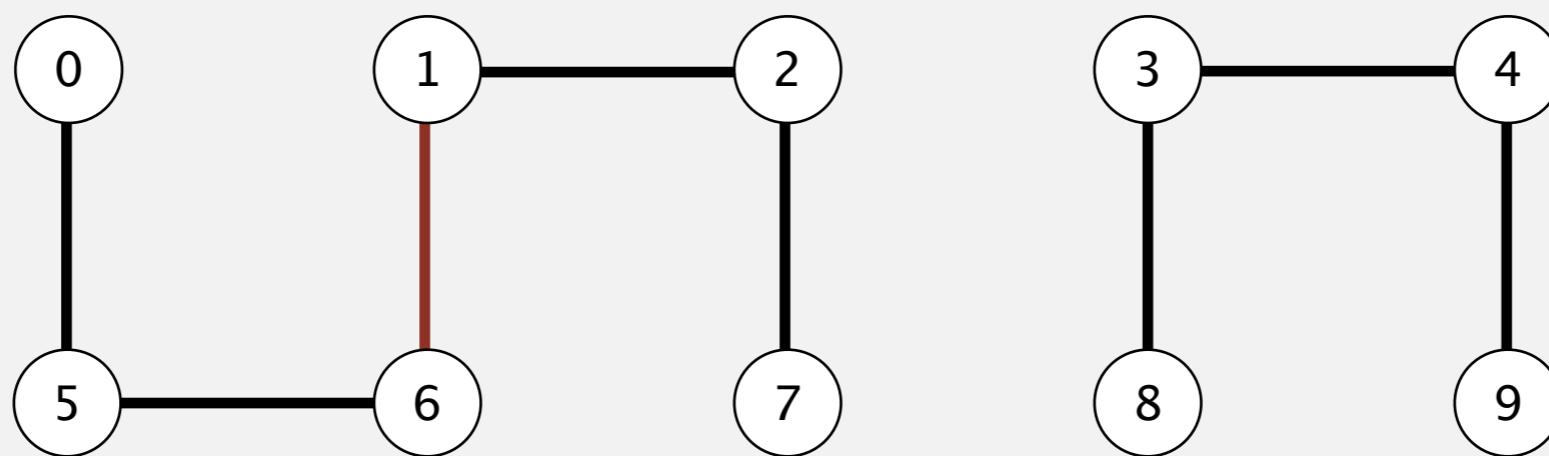
id[]	0	1	2	3	4	5	6	7	8	9
1	1	1	8	8	0	0	1	8	8	

↑                              ↑

# Quick-find demo

---

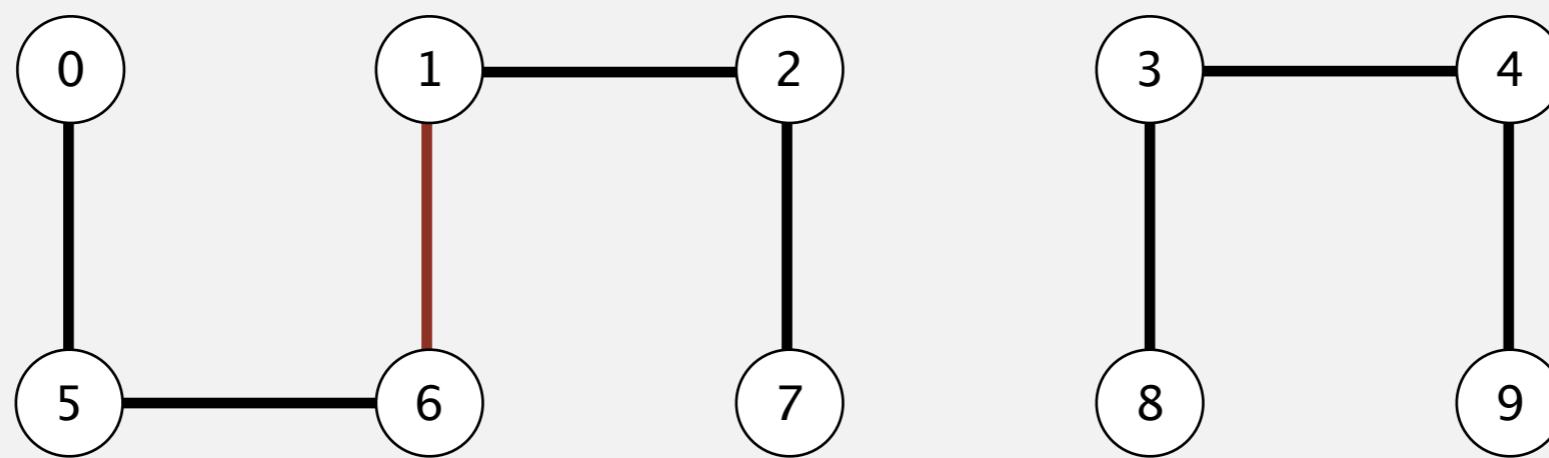
**union(6, 1)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	1	1	1	8	8	1	0	1	8	8
	↑					↑				

## Quick-find demo

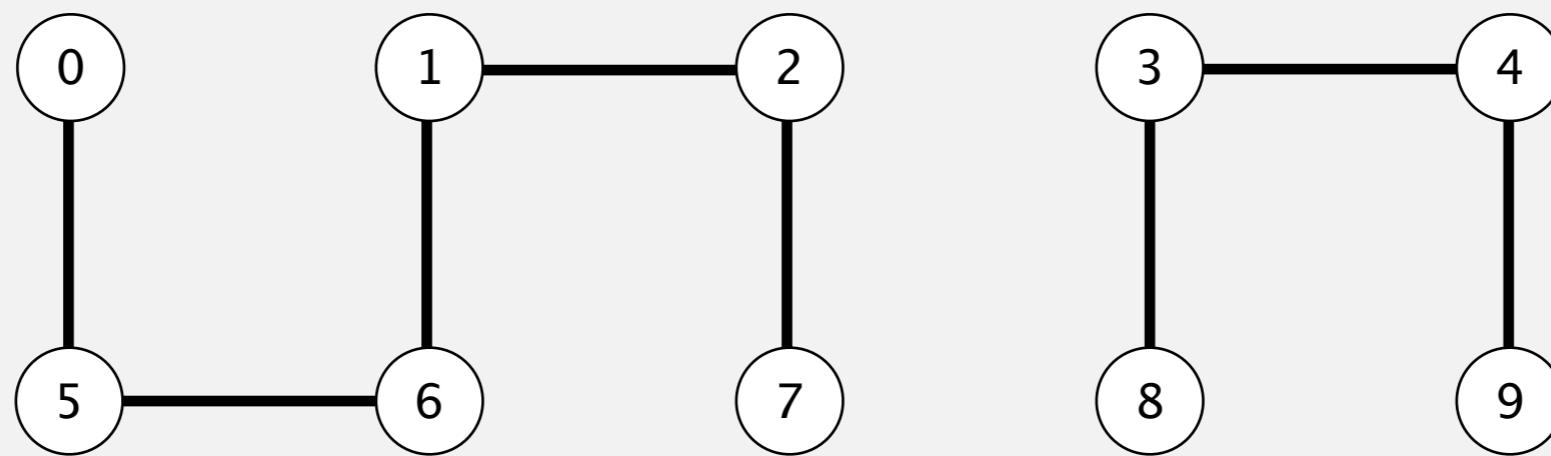
**union(6, 1)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	1	1	1	8	8	1	1	1	8	8
	↑					↑				

# Quick-find demo

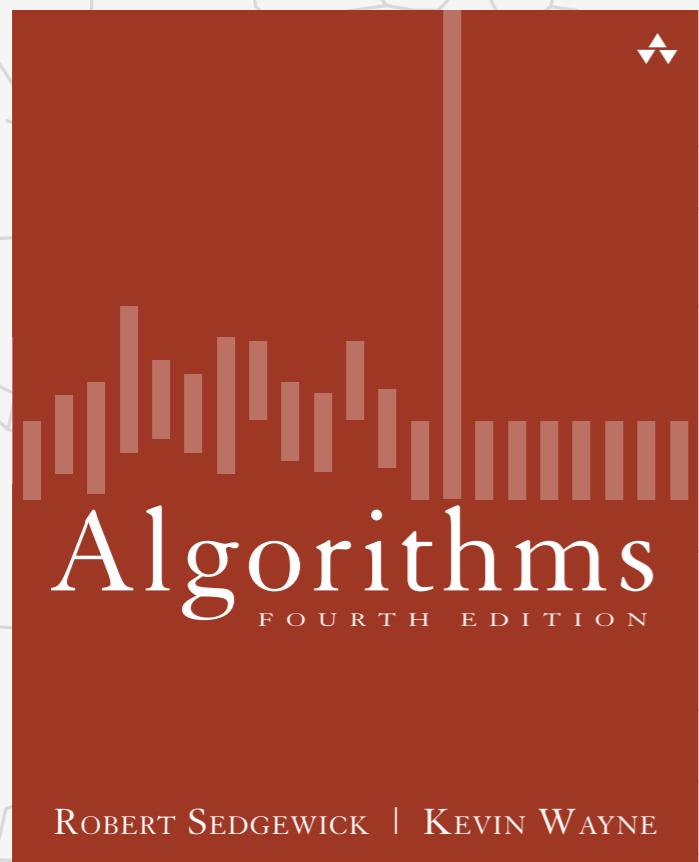
---



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	1	1	1	8	8	1	1	1	8	8

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 1.5 QUICK-UNION DEMO

---

# Quick-union demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Quick-union demo

---

**union(4, 3)**

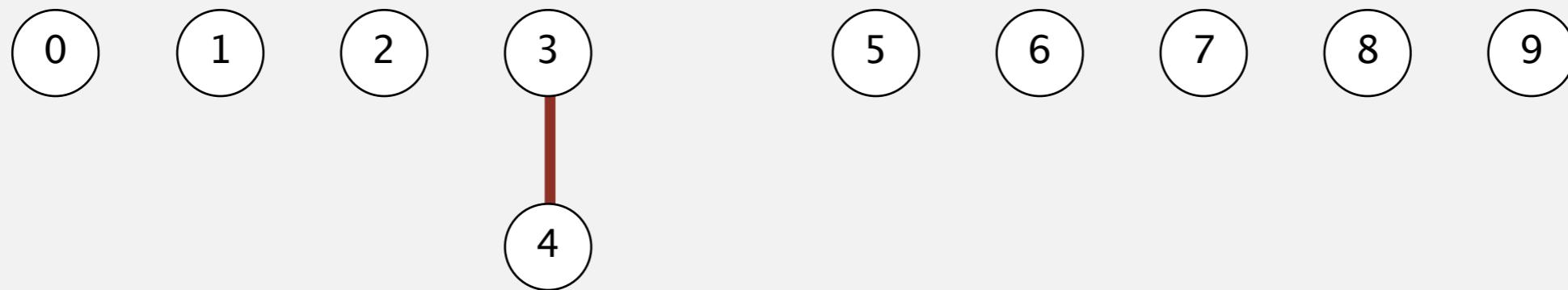


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Quick-union demo

---

**union(4, 3)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	3	<b>3</b>	5	6	7	8	9

# Quick-union demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	3	5	6	7	8	9

# Quick-union demo

---

**union(3, 8)**

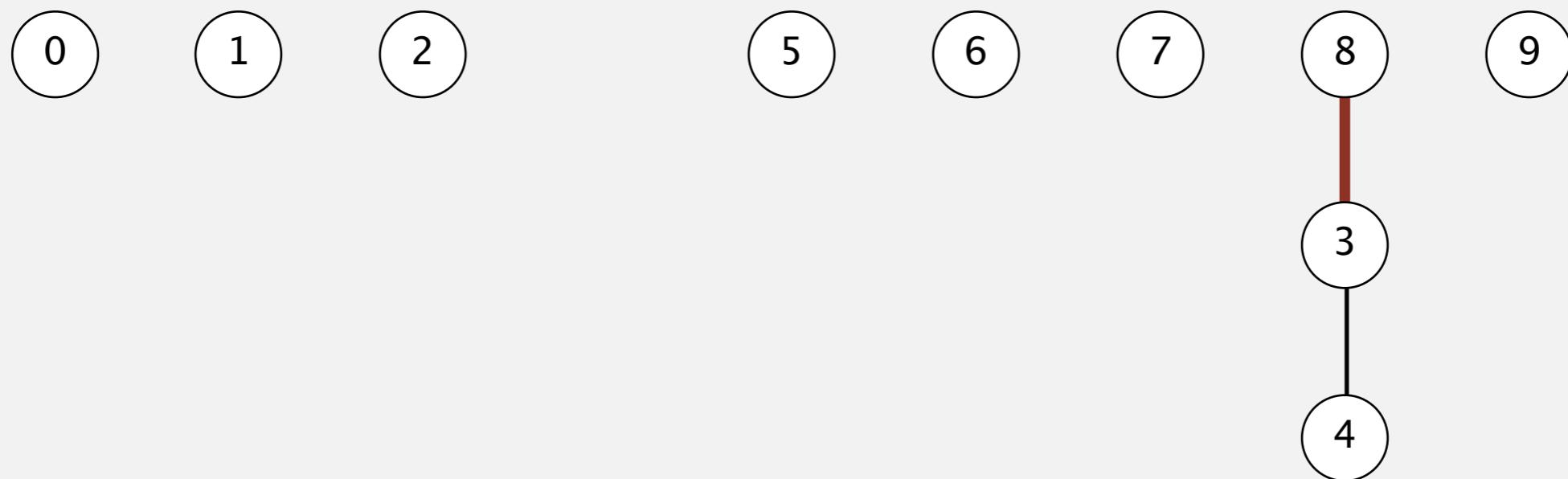


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	3	3	5	6	7	8	9

# Quick-union demo

---

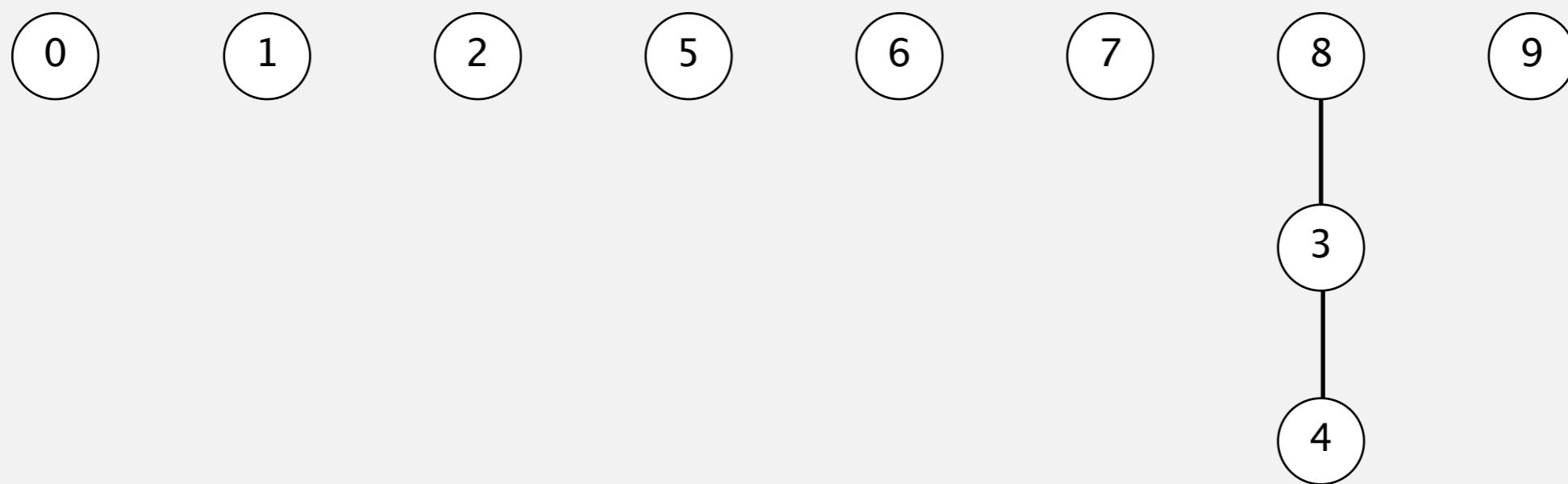
**union(3, 8)**



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

# Quick-union demo

---

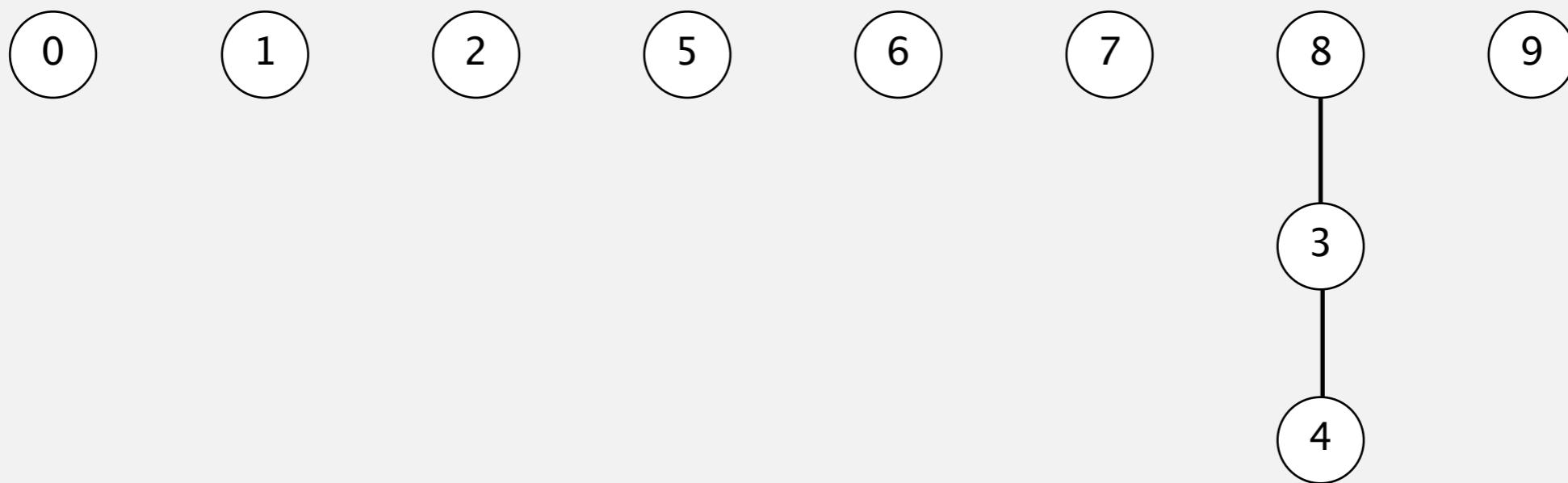


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	6	7	8	9

# Quick-union demo

---

**union(6, 5)**

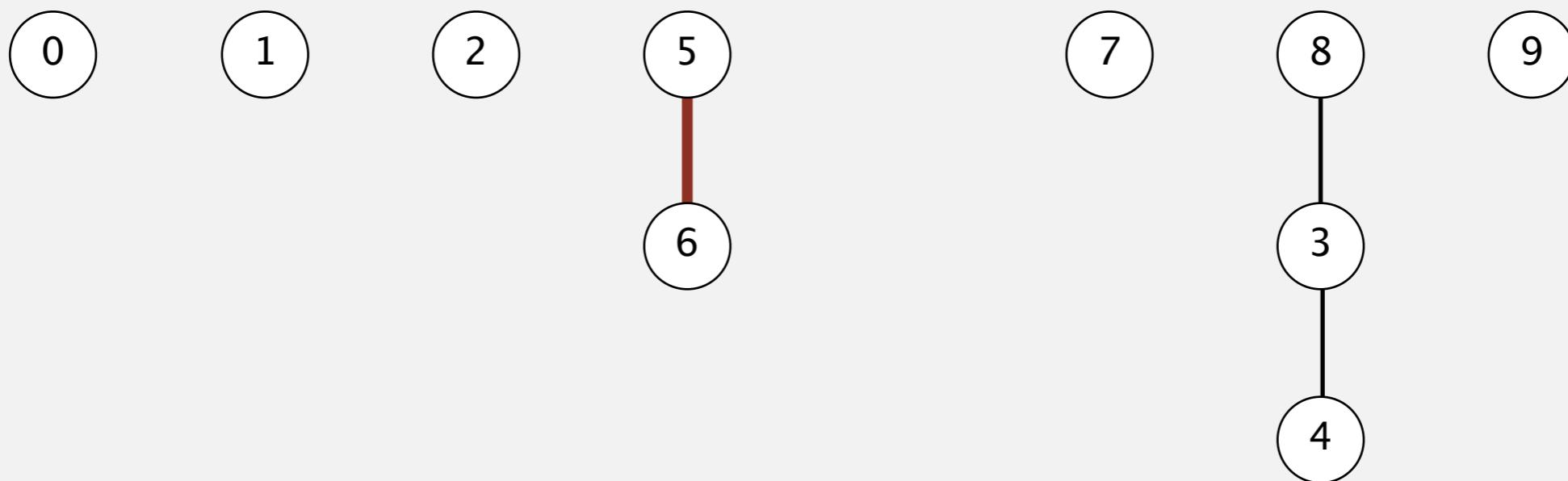


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	6	7	8	9

# Quick-union demo

---

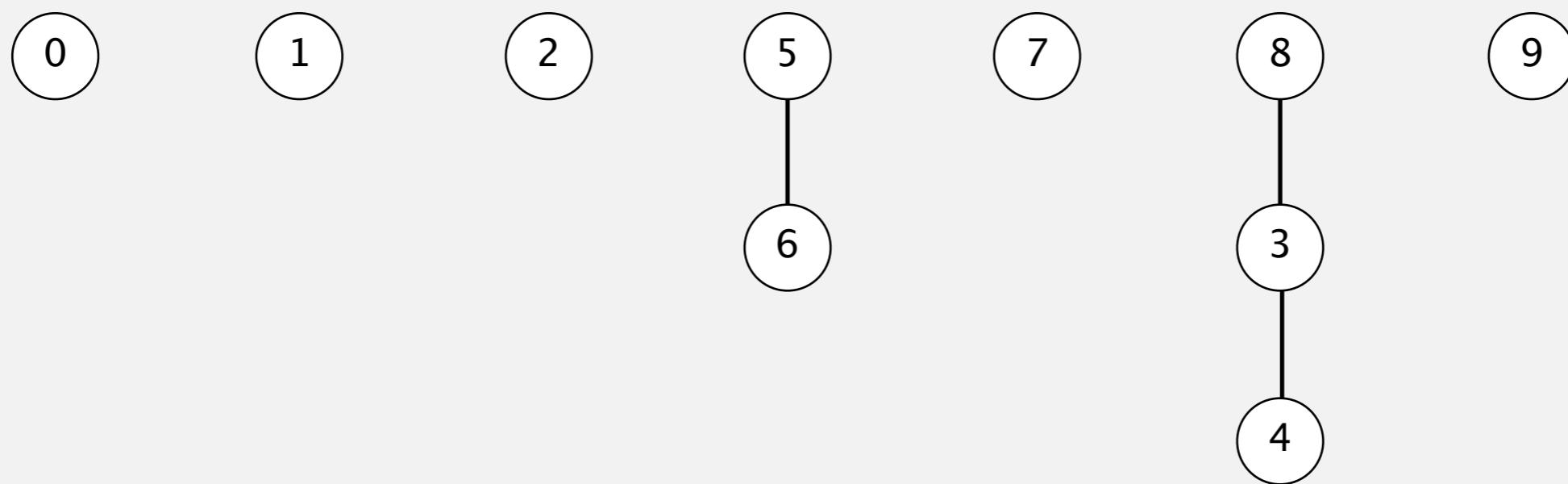
**union(6, 5)**



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

# Quick-union demo

---

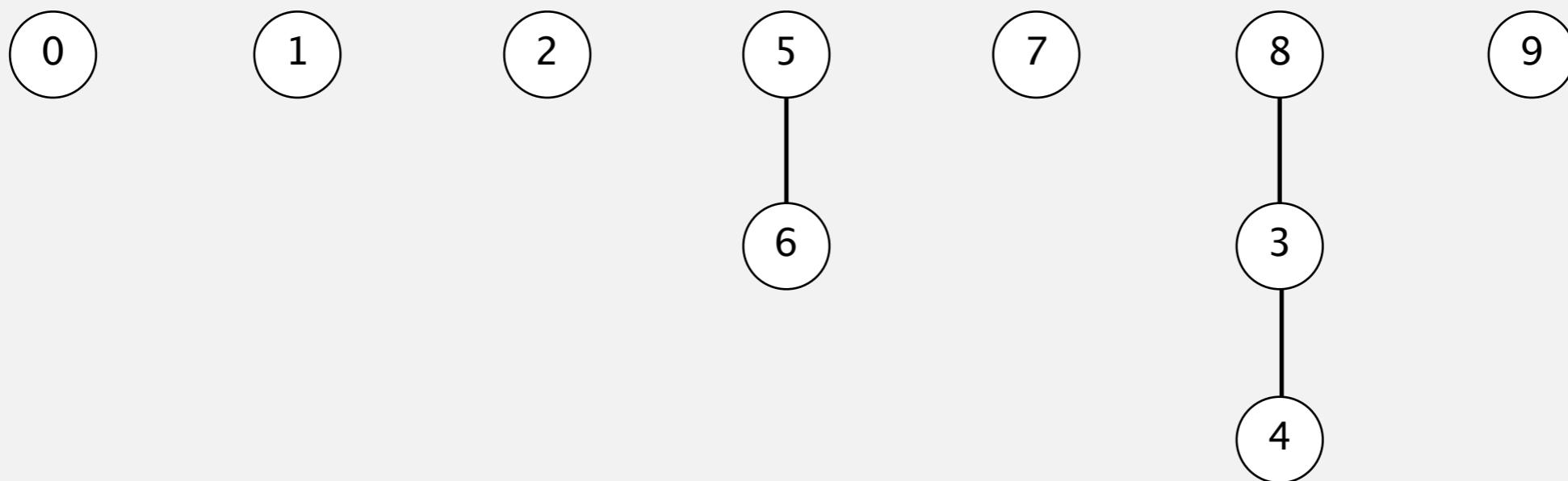


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	8	3	5	5	7	8	9

# Quick-union demo

---

**union(9, 4)**

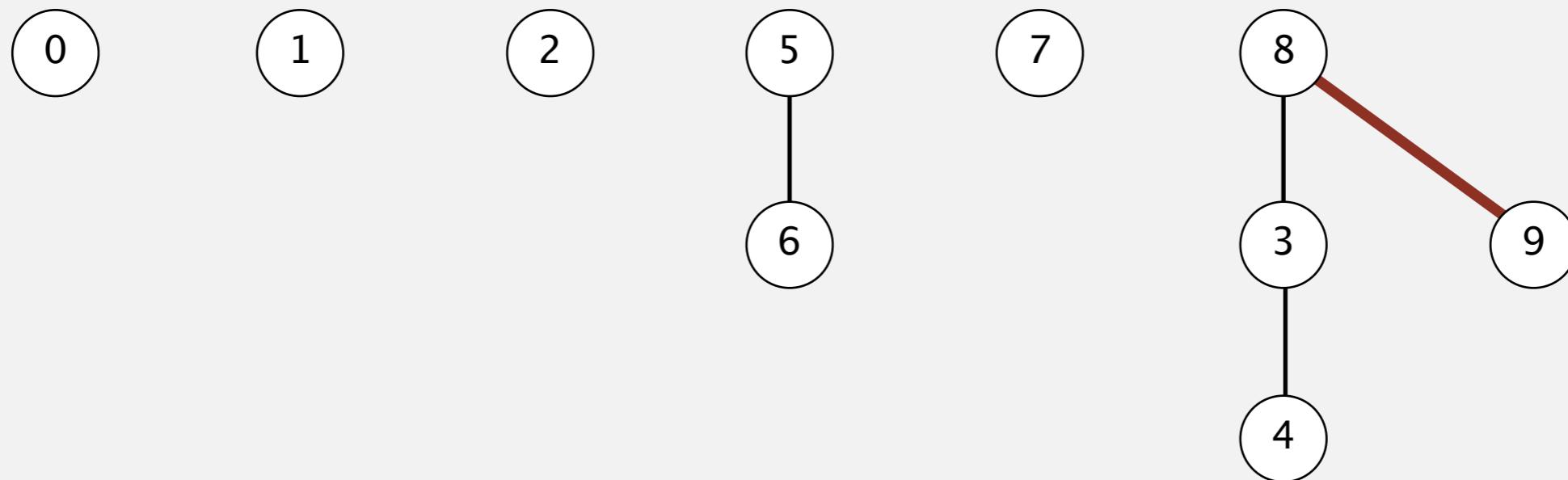


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	8	3	5	5	7	8	9

# Quick-union demo

---

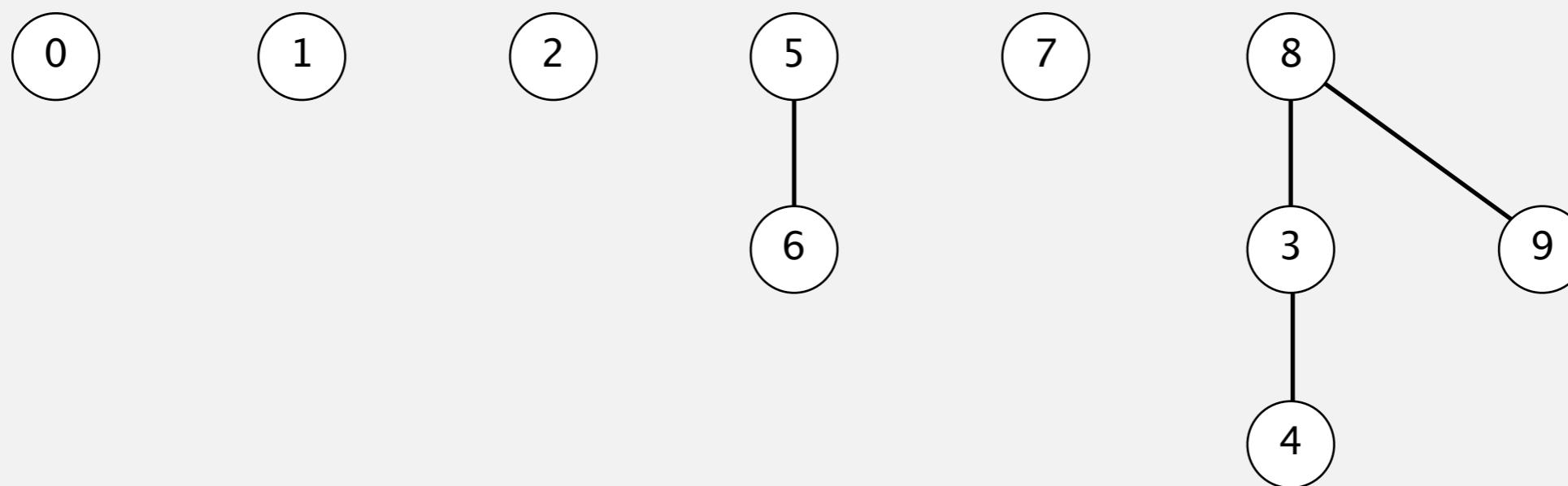
**union(9, 4)**



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

# Quick-union demo

---

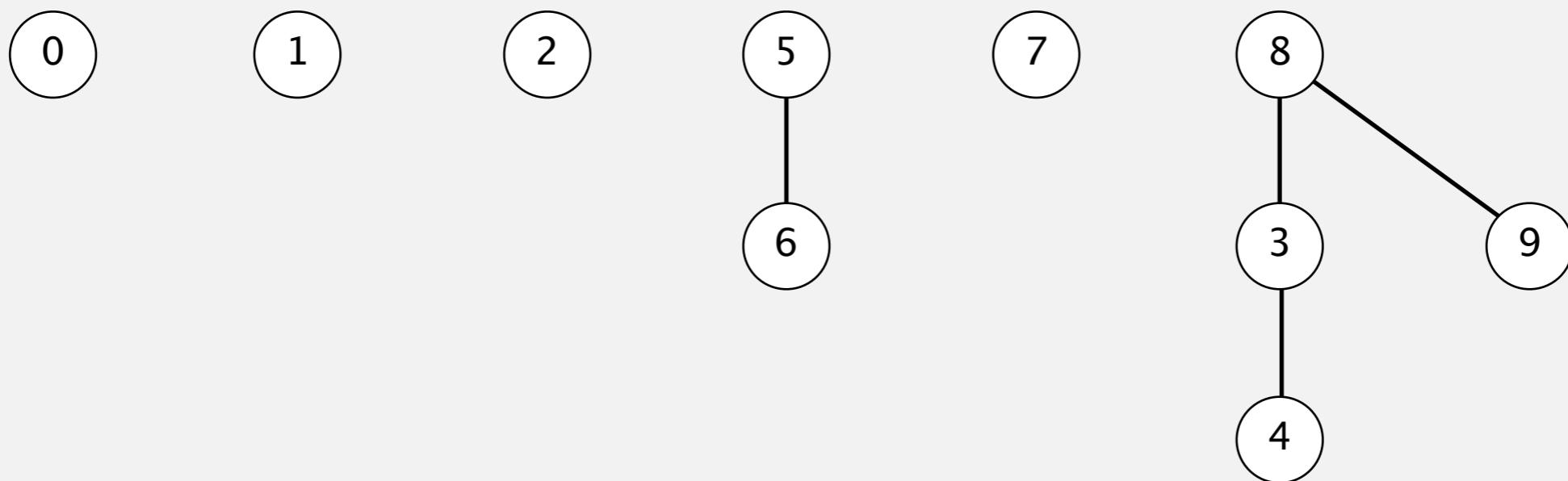


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	8	3	5	5	7	8	8

# Quick-union demo

---

**union(2, 1)**

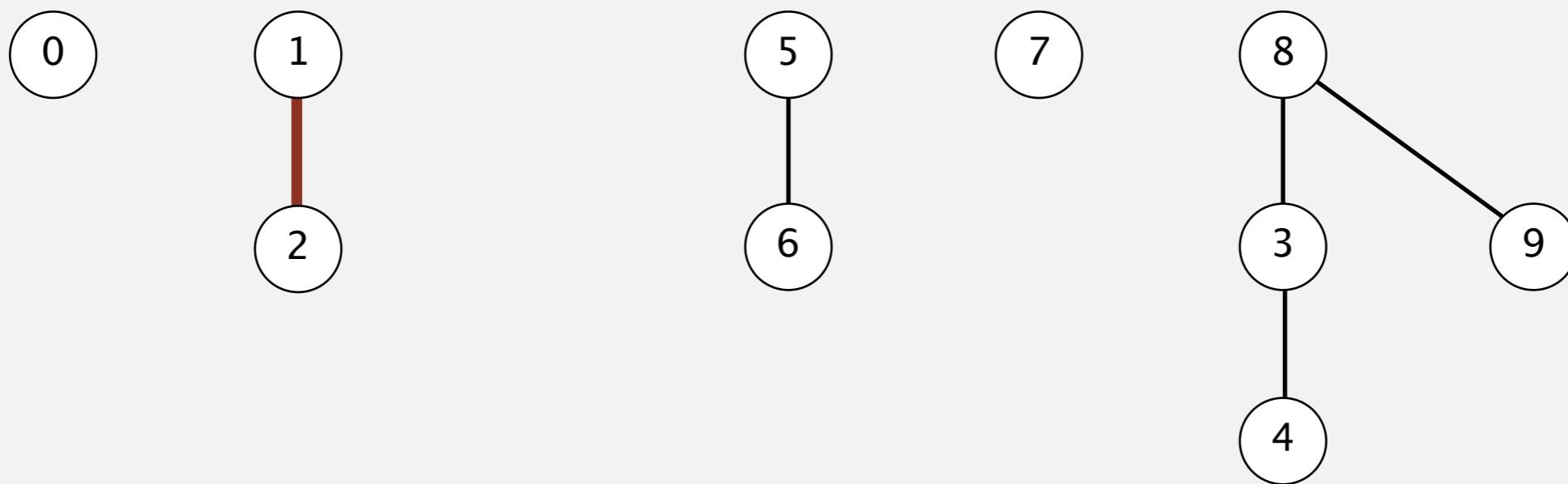


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	8

# Quick-union demo

---

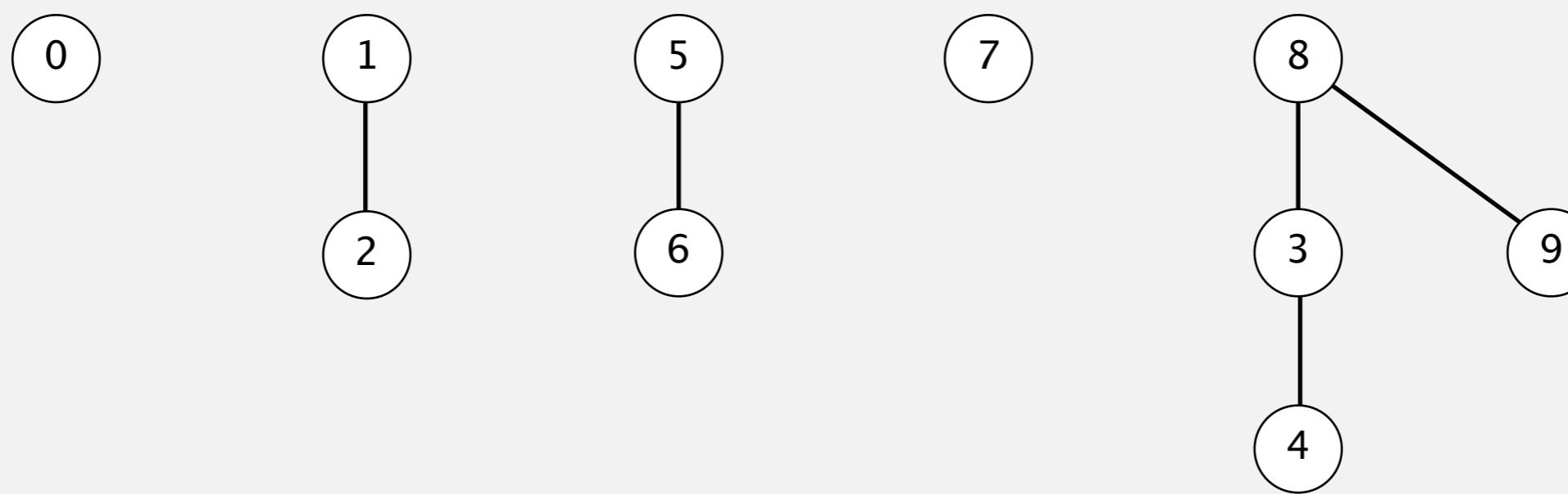
**union(2, 1)**



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

# Quick-union demo

---

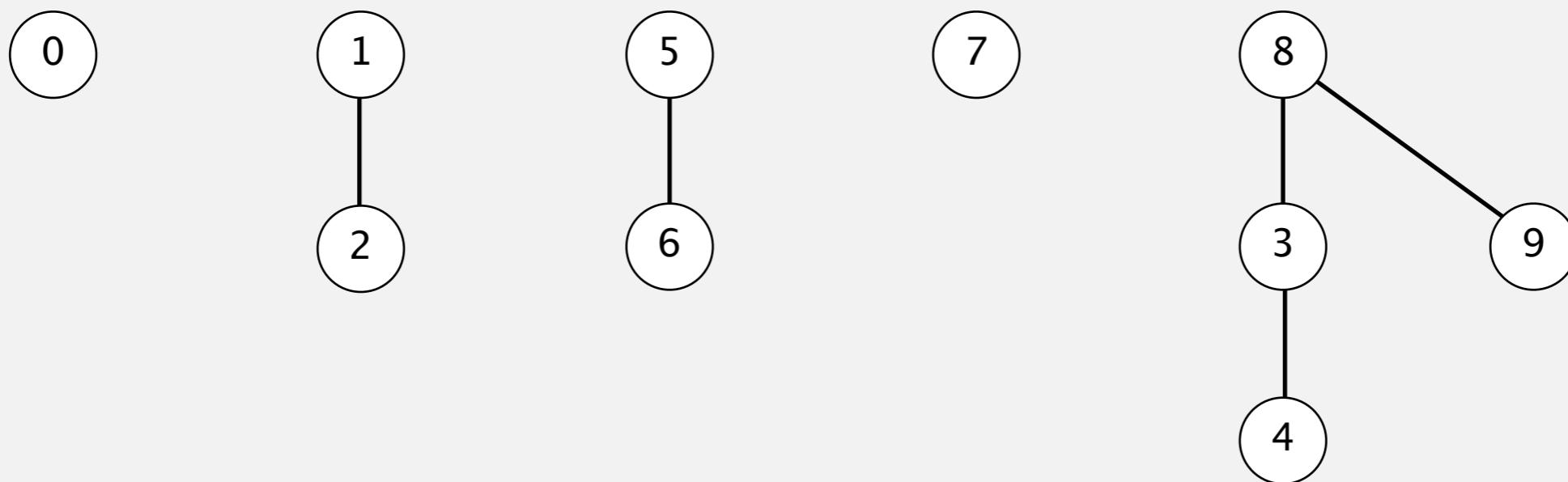


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

# Quick-union demo

---

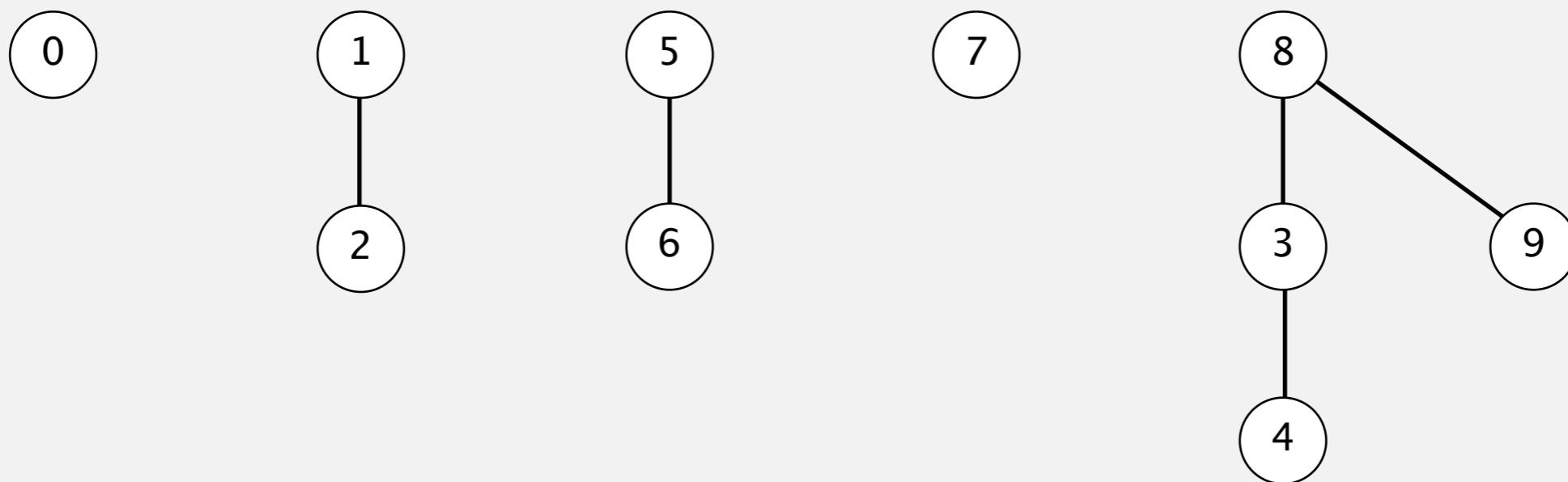
connected(8, 9) ✓



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

# Quick-union demo

connected(5, 4) X

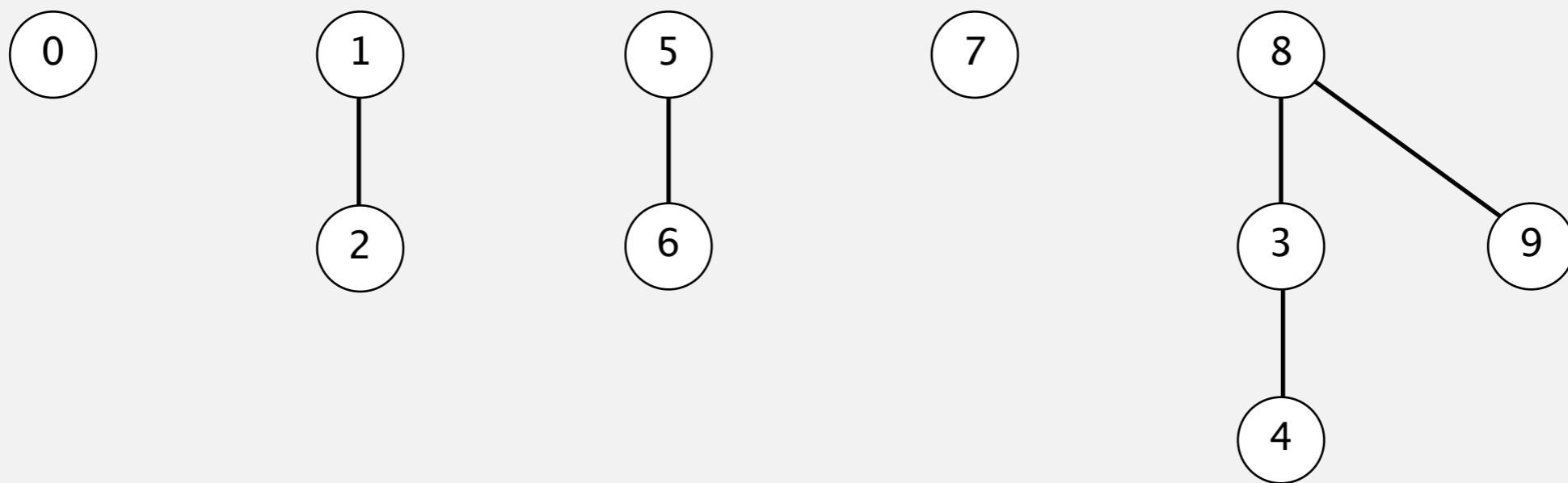


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

# Quick-union demo

---

**union(5, 0)**

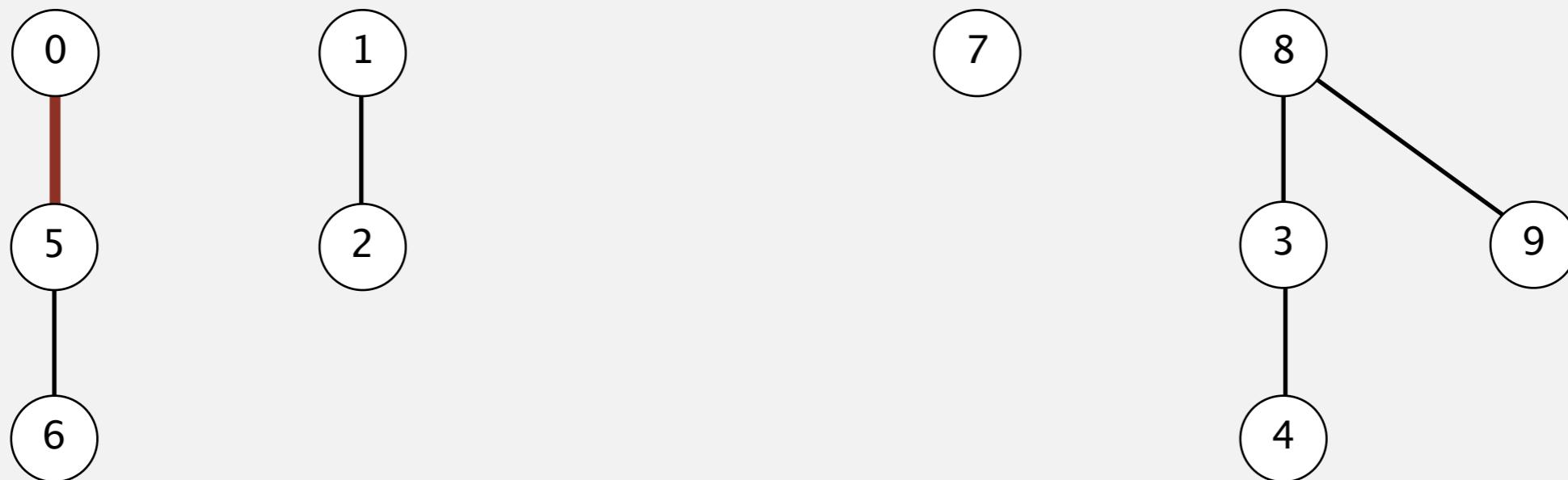


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

# Quick-union demo

---

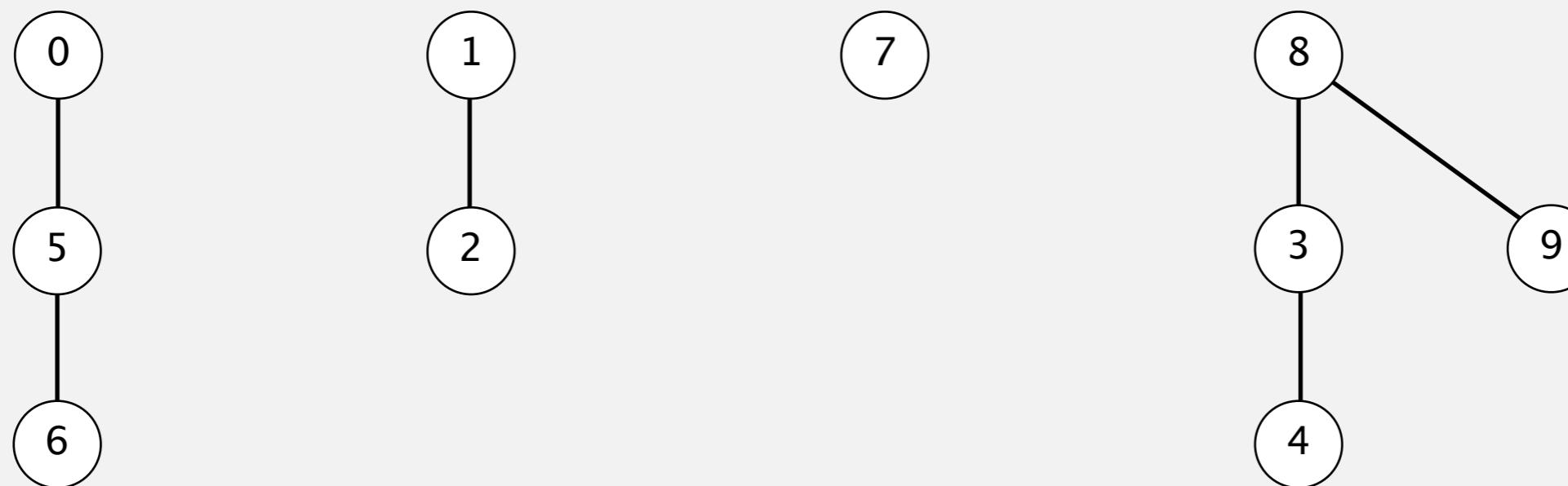
**union(5, 0)**



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

# Quick-union demo

---

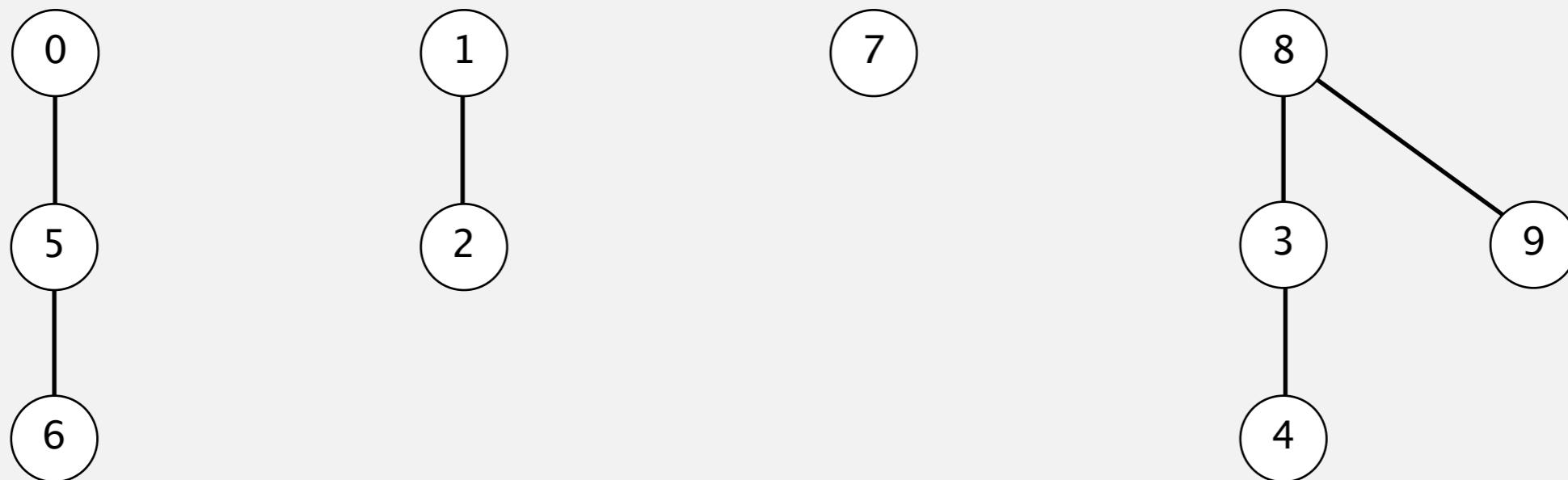


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8

# Quick-union demo

---

**union(7, 2)**

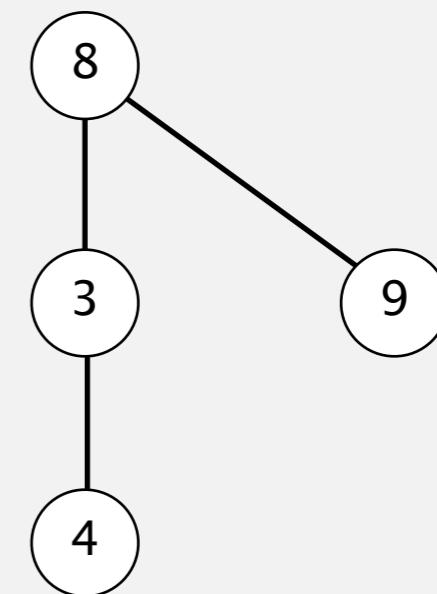
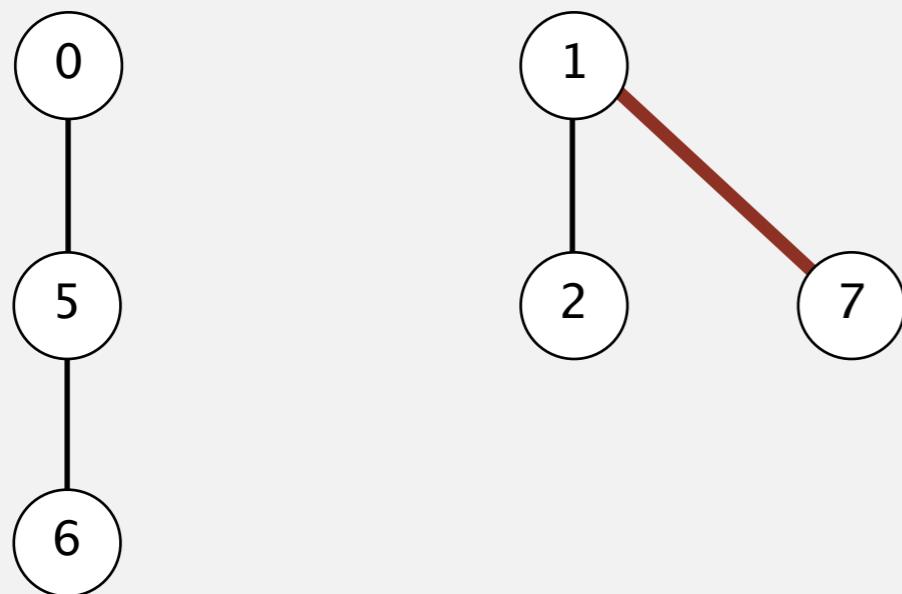


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8

# Quick-union demo

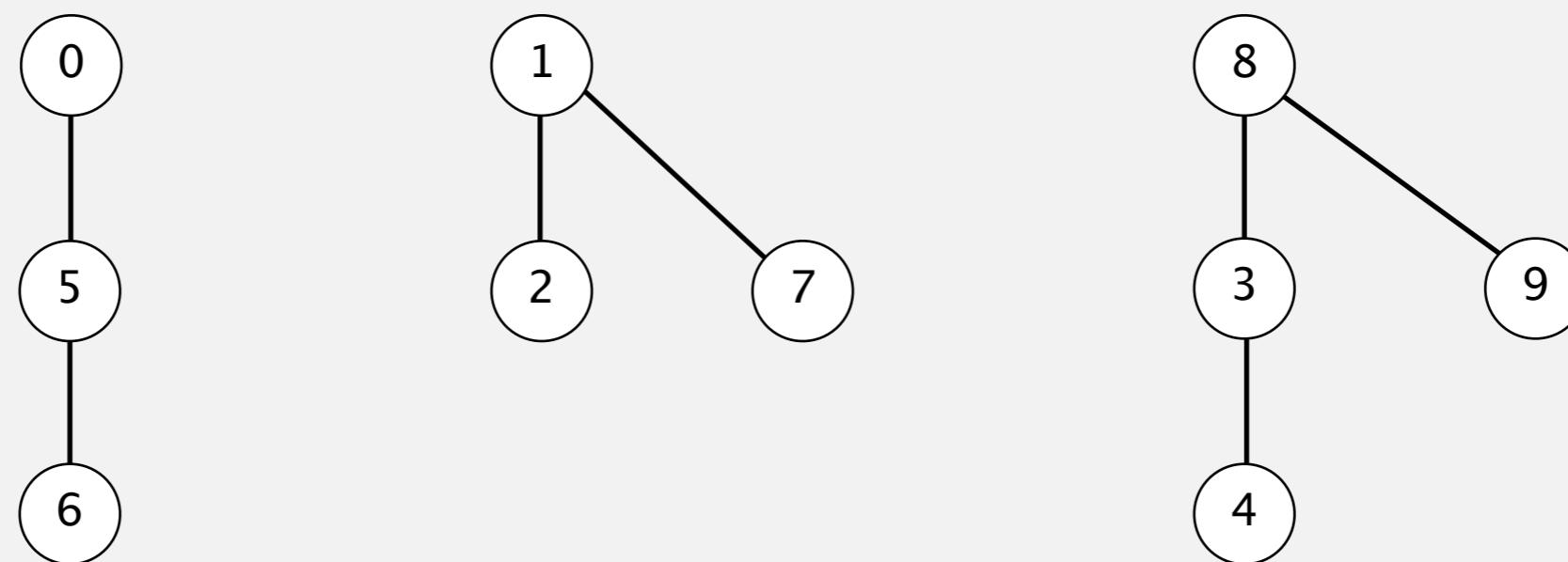
---

**union(7, 2)**



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

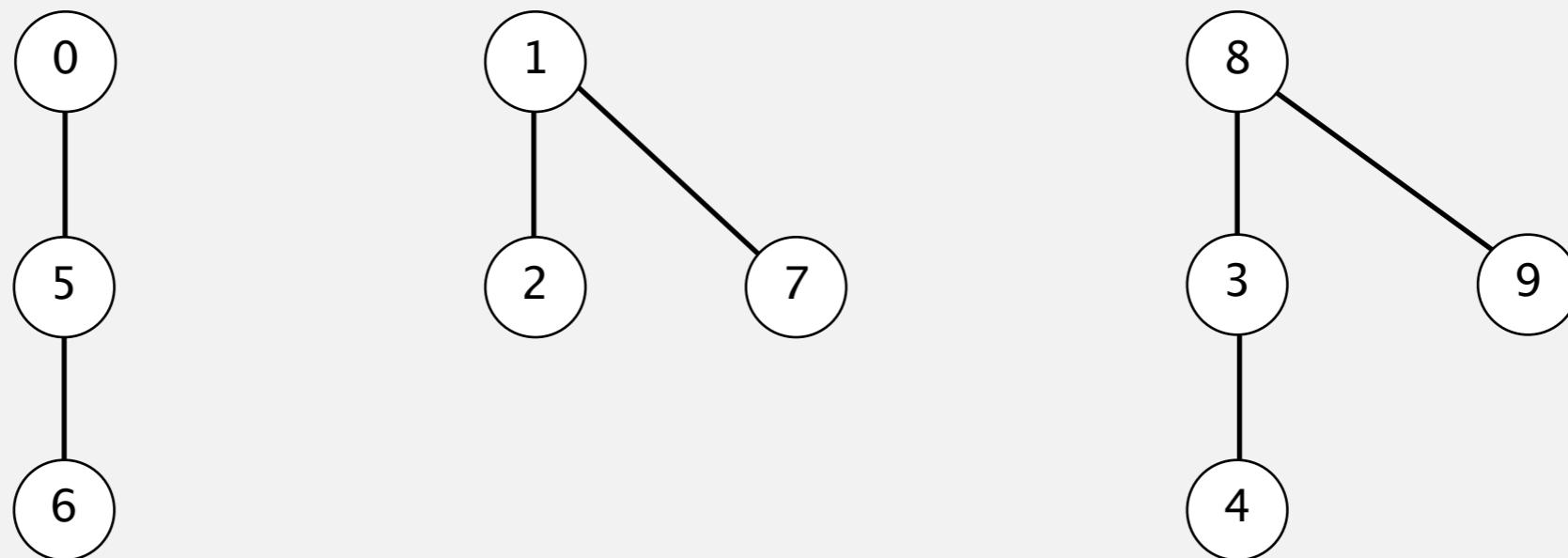
# Quick-union demo



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	3	0	5	1	8	8

# Quick-union demo

**union(6, 1)**

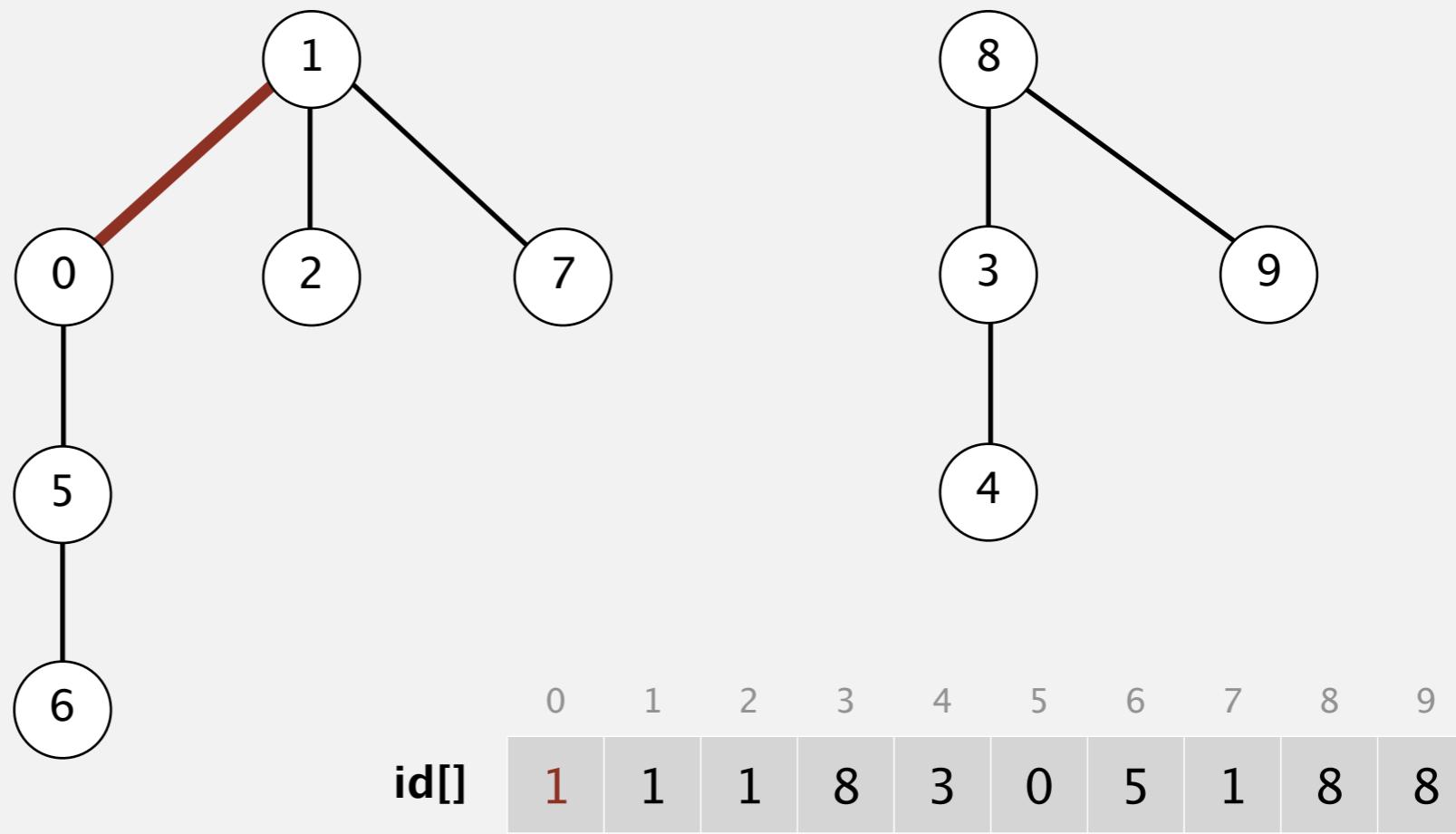


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	3	0	5	1	8	8

# Quick-union demo

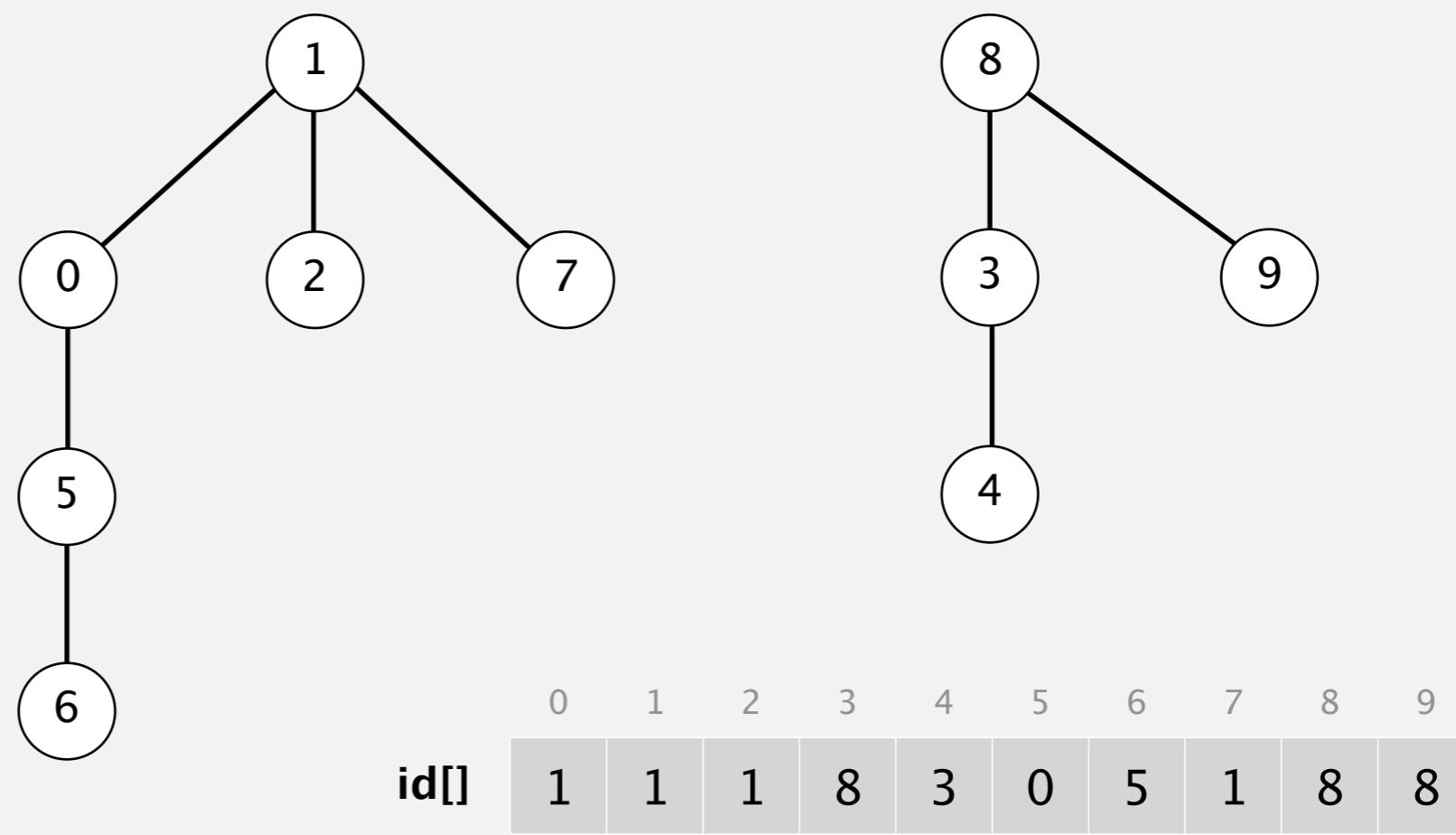
---

**union(6, 1)**



# Quick-union demo

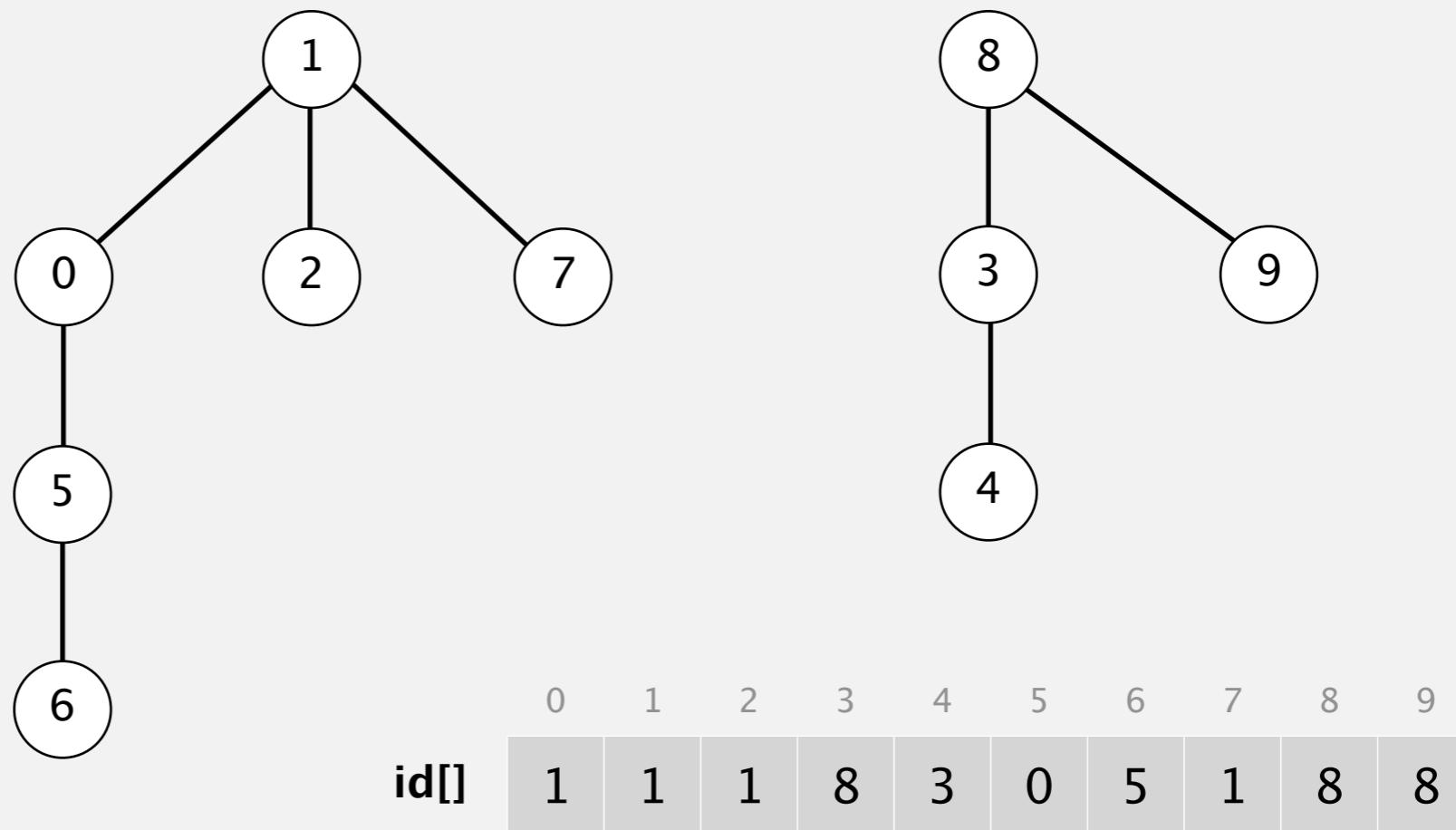
---



# Quick-union demo

---

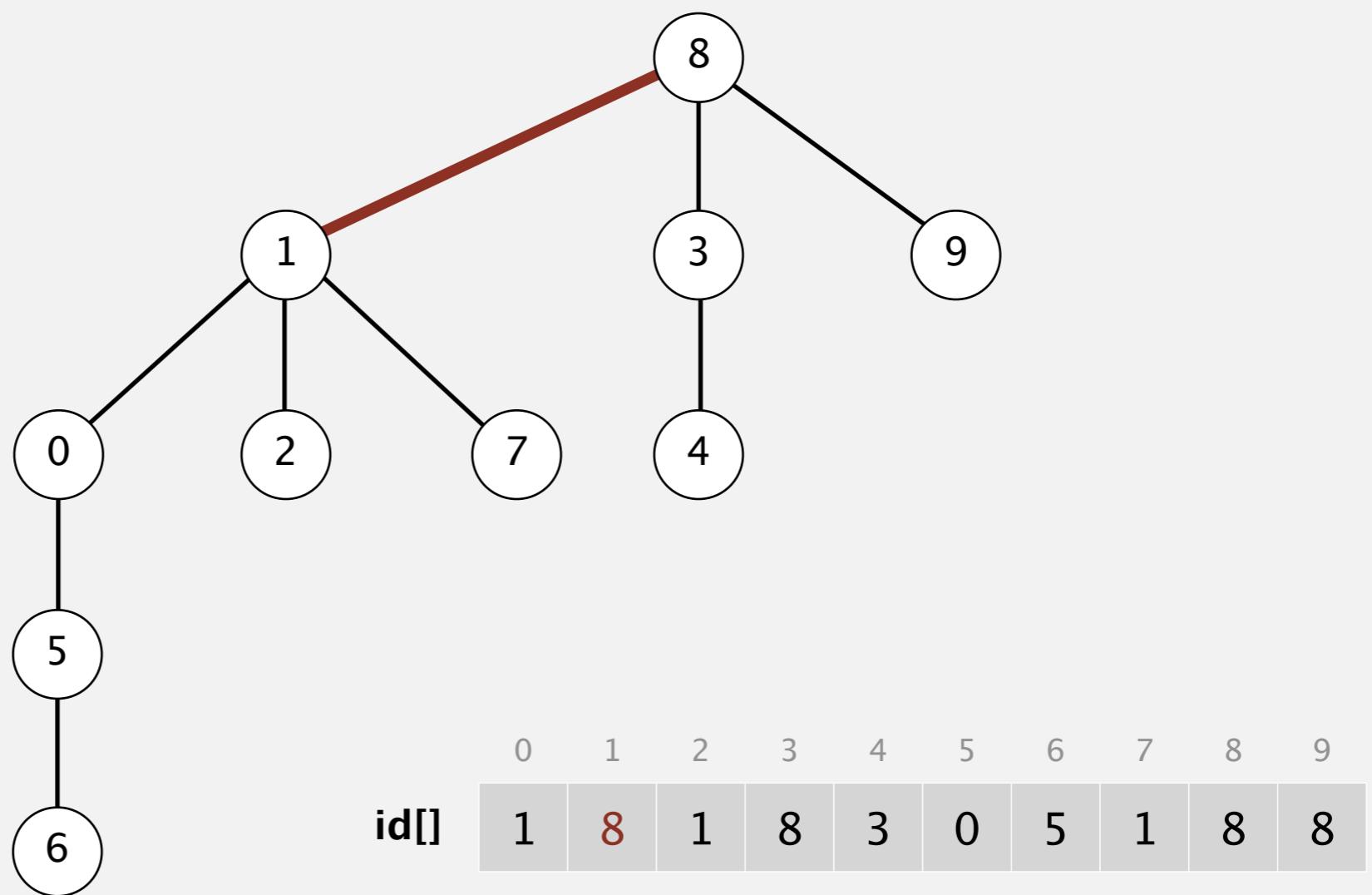
**union(7, 3)**



# Quick-union demo

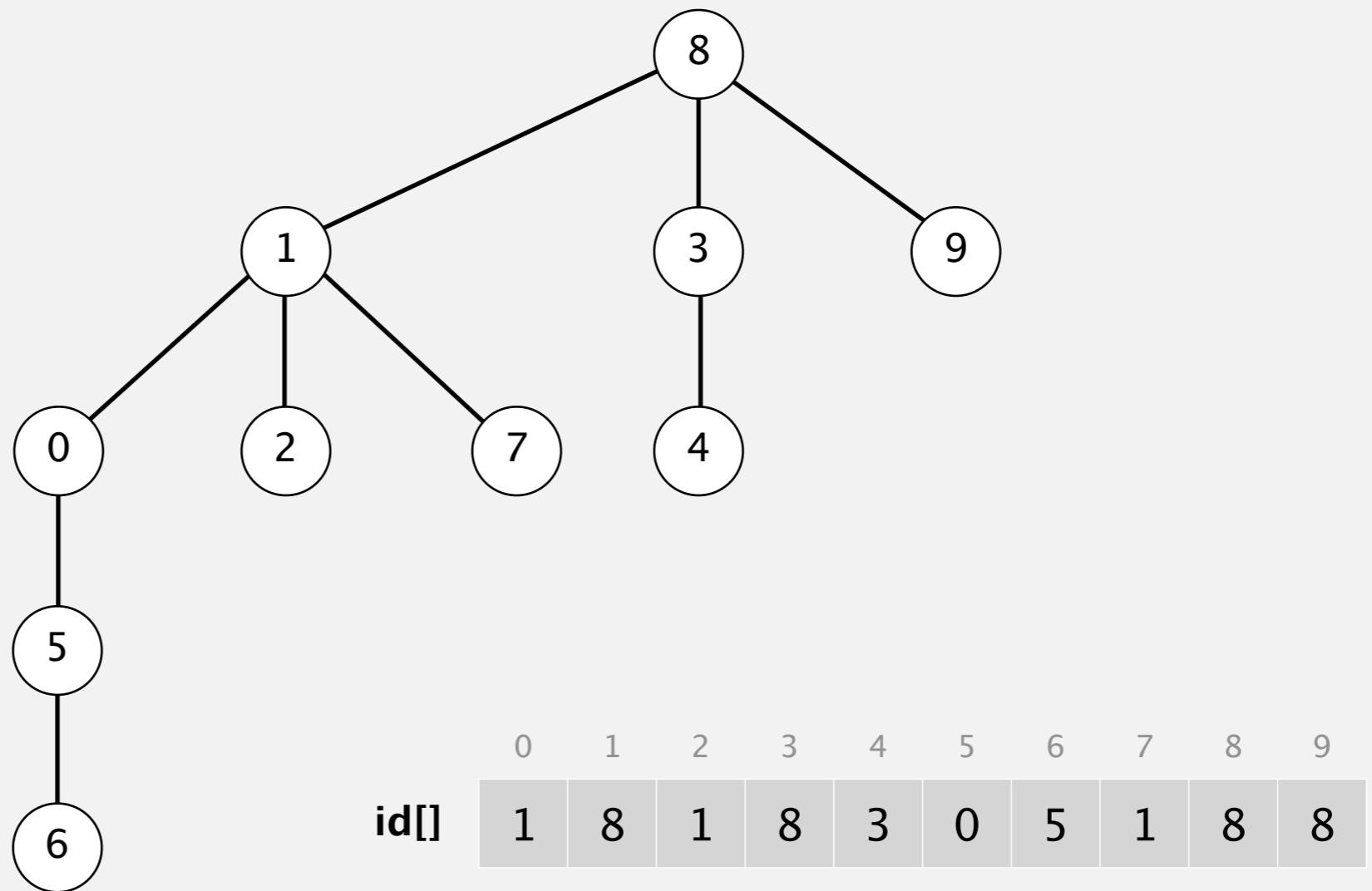
---

**union(7, 3)**



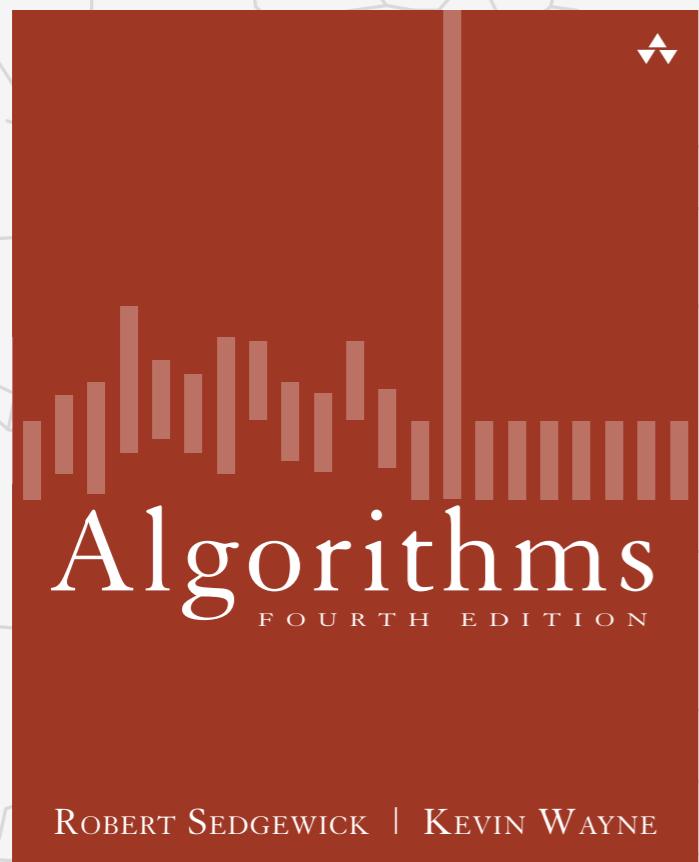
# Quick-union demo

---



# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

## 1.5 WEIGHTED QUICK-UNION DEMO

---

# Weighted quick-union demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Weighted quick-union demo

---

**union(4, 3)**

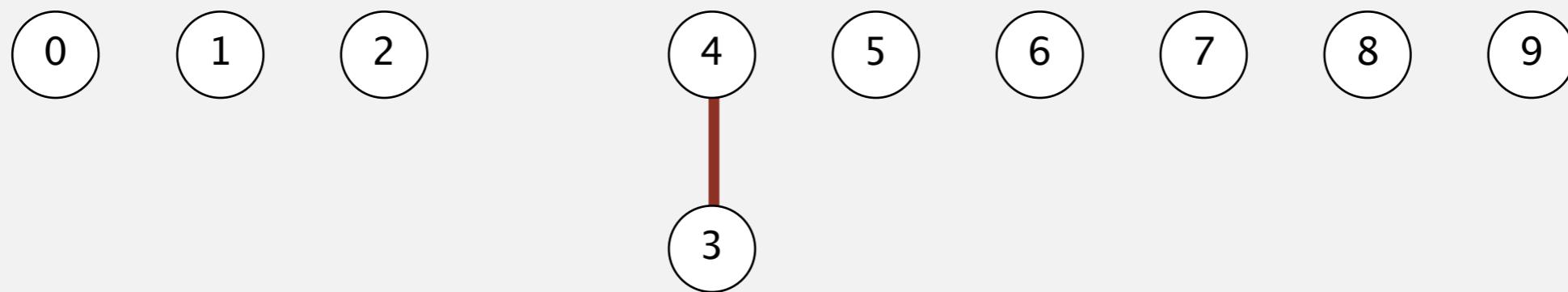


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Weighted quick-union demo

---

**union(4, 3)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	<b>4</b>	4	5	6	7	8	9

# Weighted quick-union demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	8	9

# Weighted quick-union demo

---

**union(3, 8)**

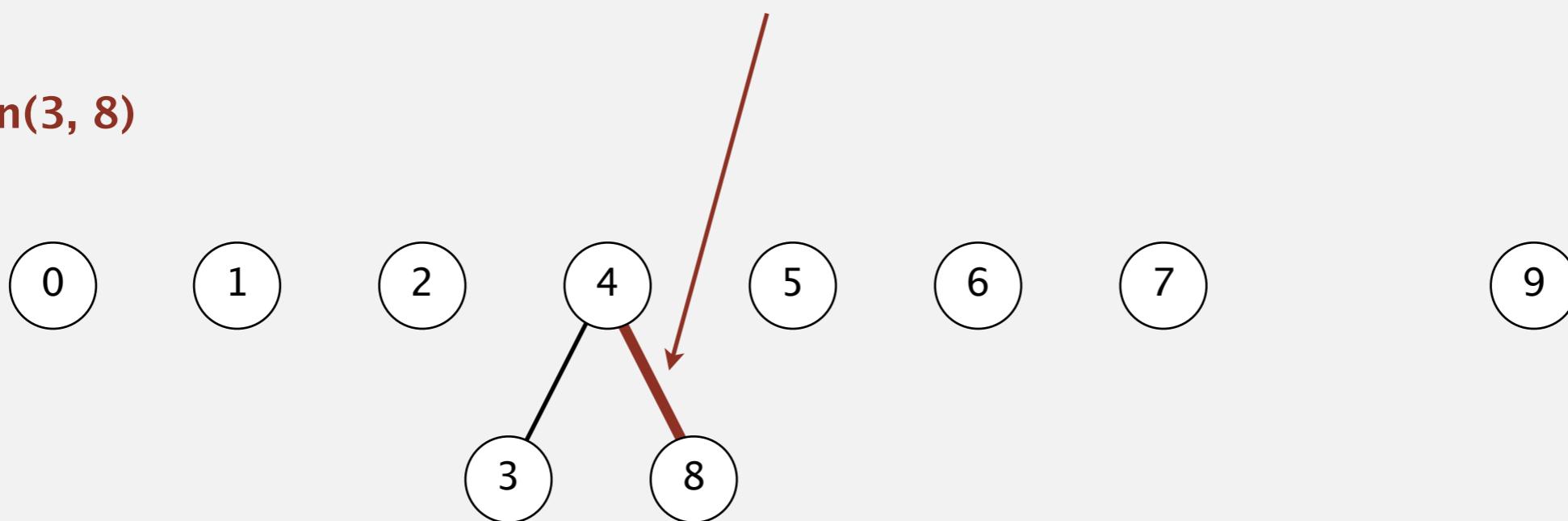


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	5	6	7	8	9

# Weighted quick-union demo

**union(3, 8)**

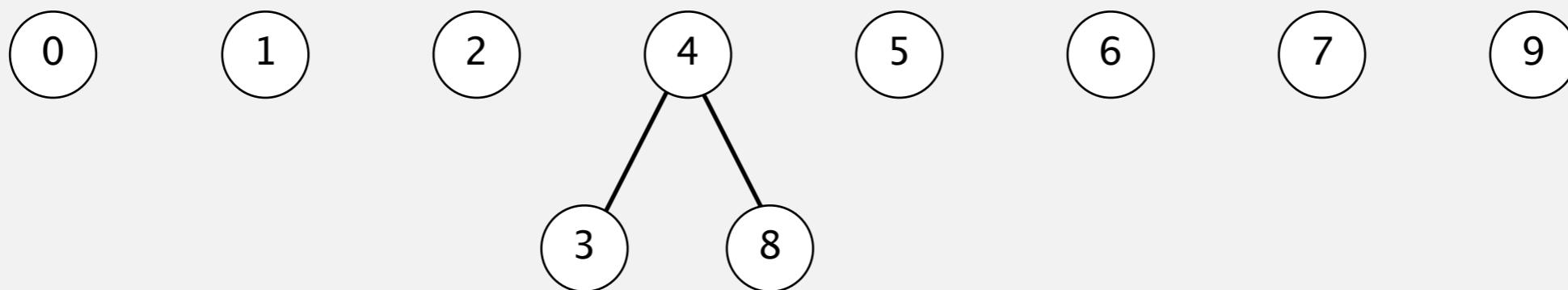
weighting: make 8 point to 4 (instead of 4 to 8)



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union demo

---

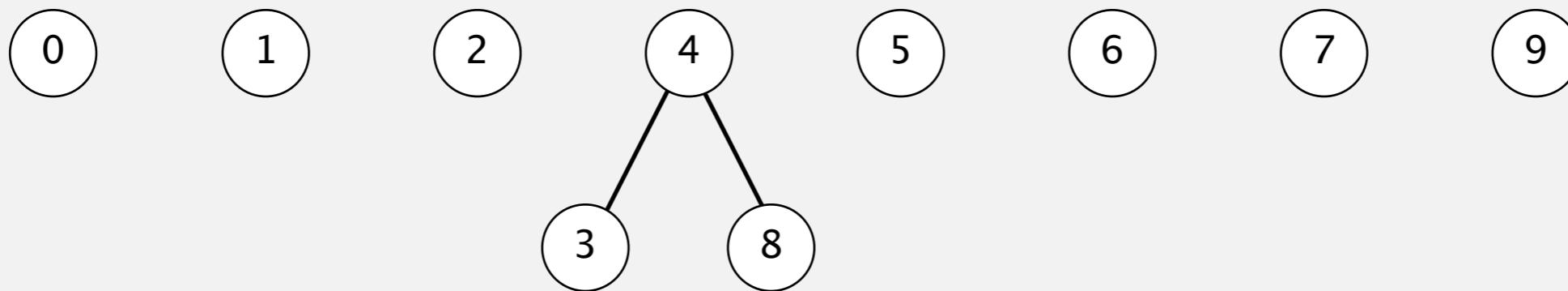


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union demo

---

**union(6, 5)**

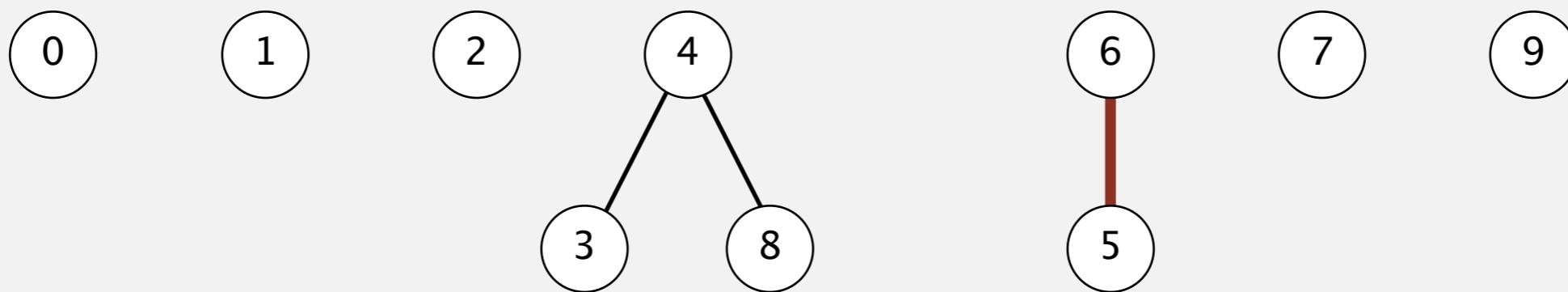


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union demo

---

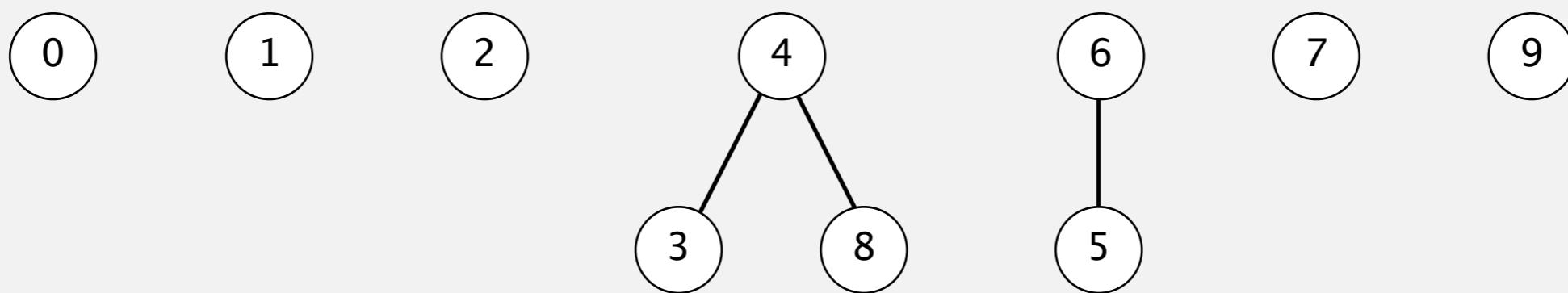
**union(6, 5)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	<b>6</b>	6	7	4	9

# Weighted quick-union demo

---

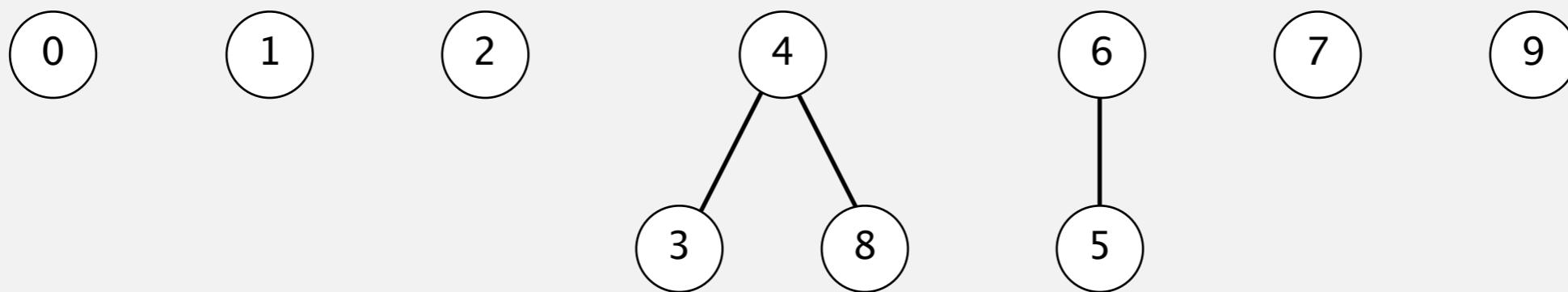


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	6	6	7	4	9

# Weighted quick-union demo

---

**union(9, 4)**

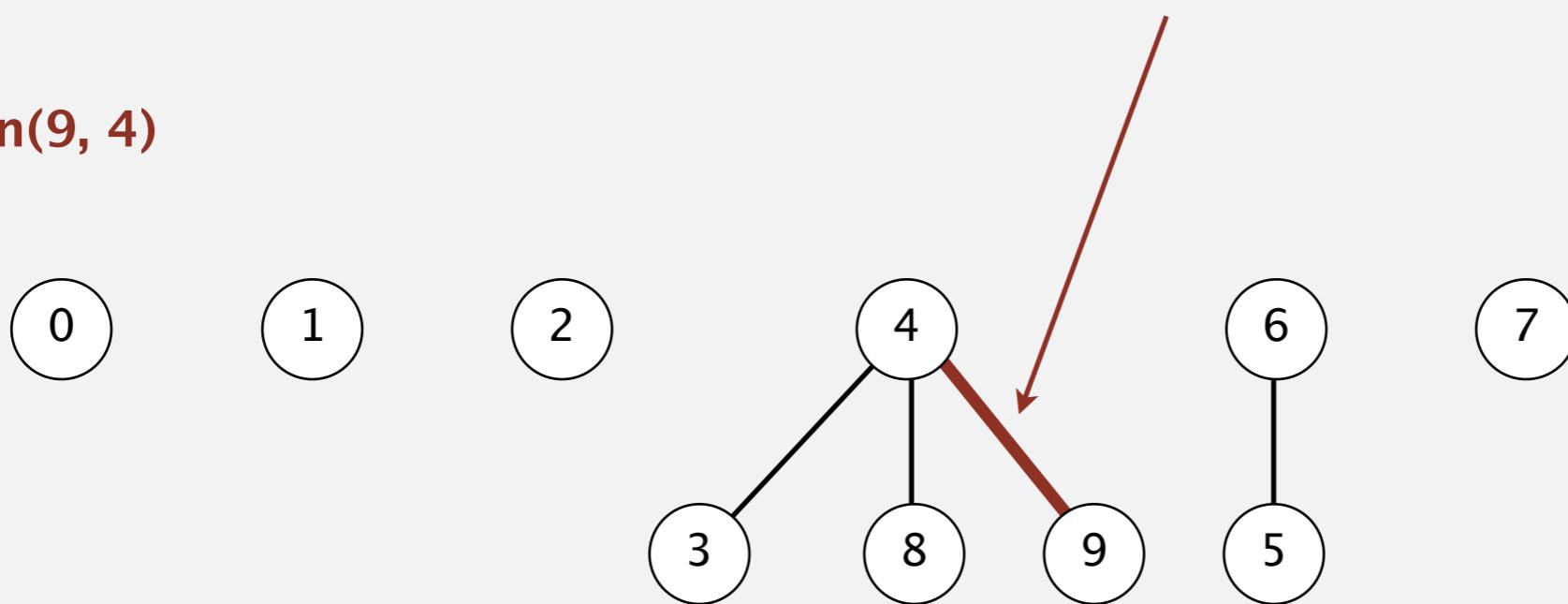


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	6	6	7	4	9

# Weighted quick-union demo

**union(9, 4)**

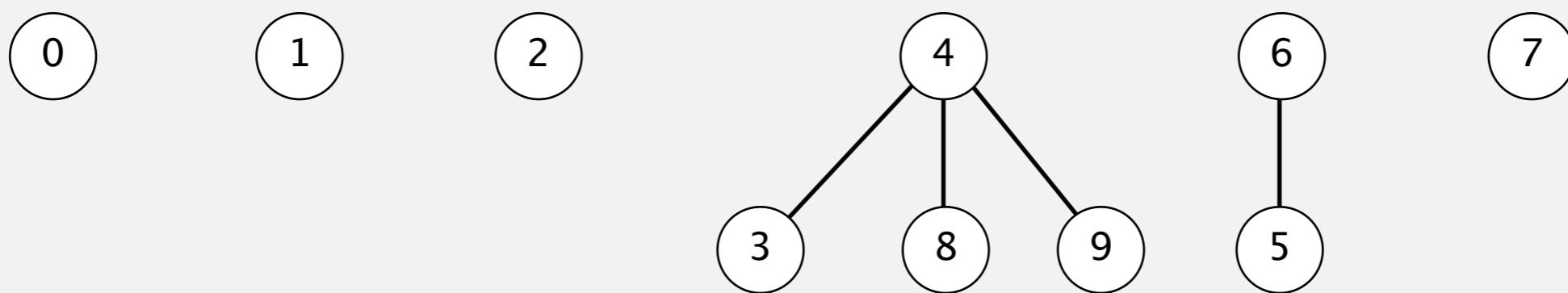
weighting: make 9 point to 4



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

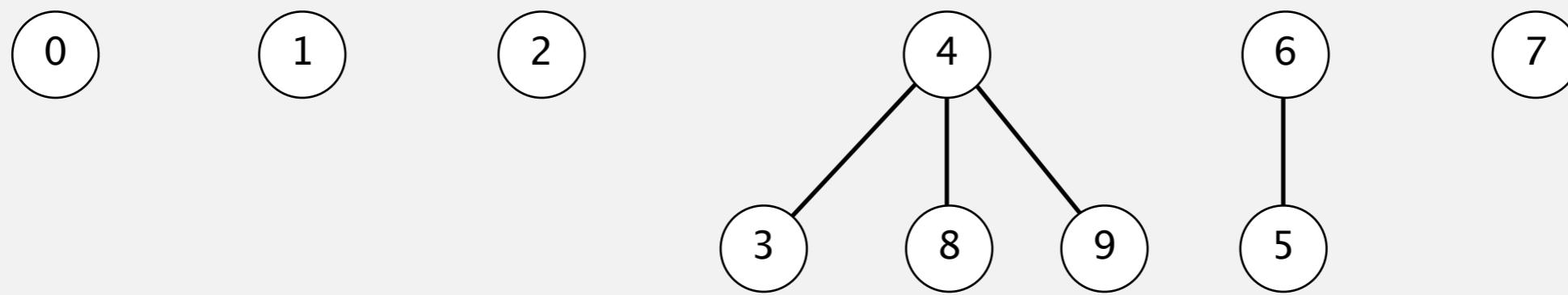


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

**union(2, 1)**

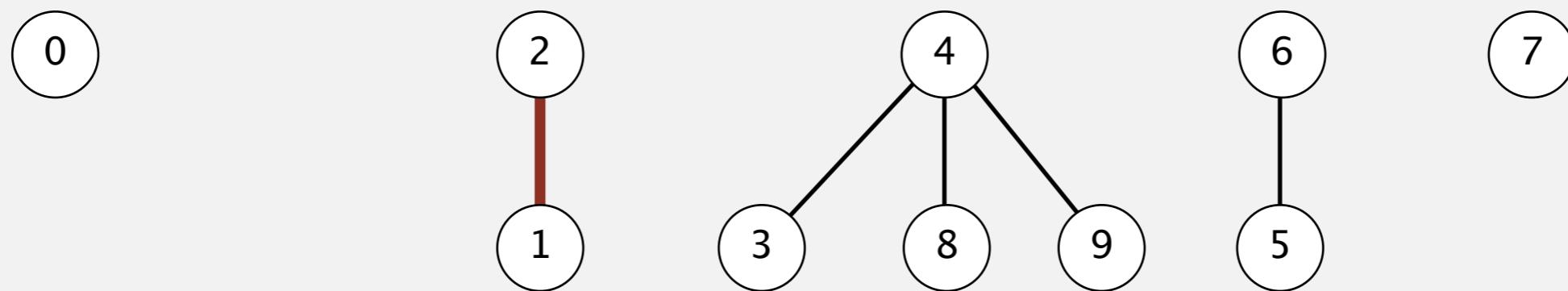


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

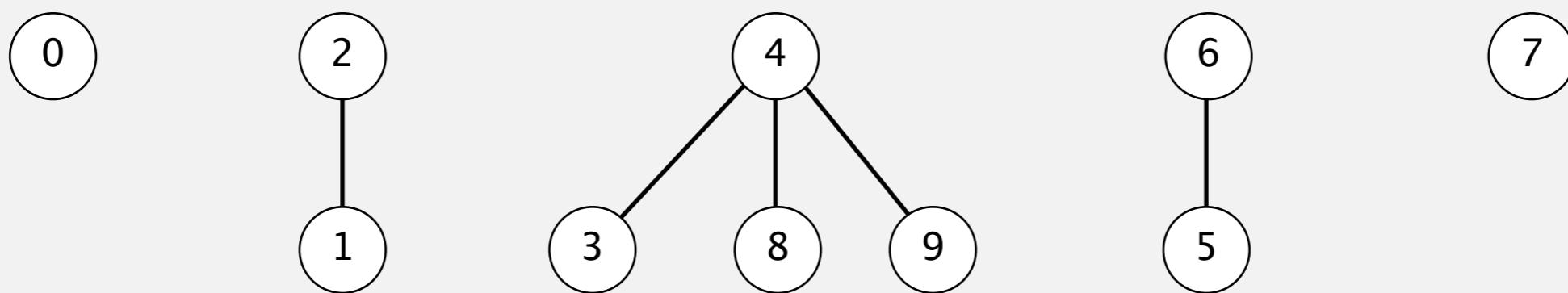
**union(2, 1)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

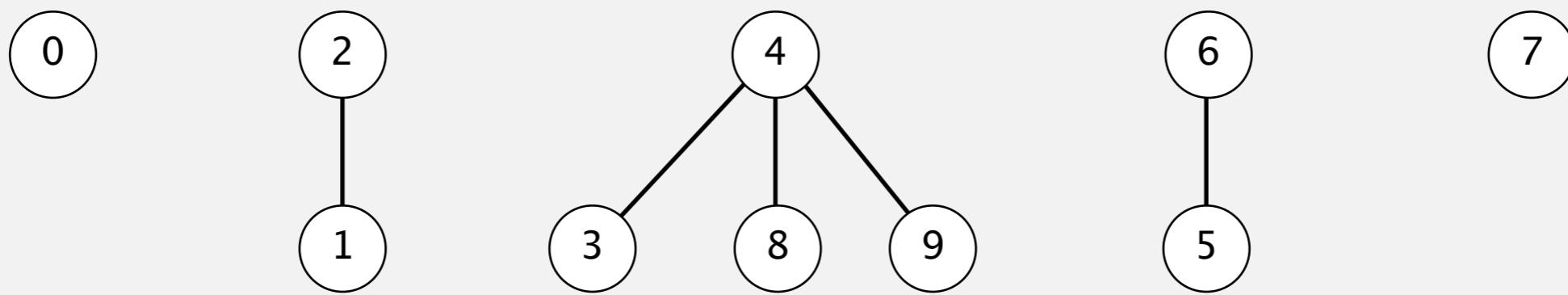


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

**union(5, 0)**

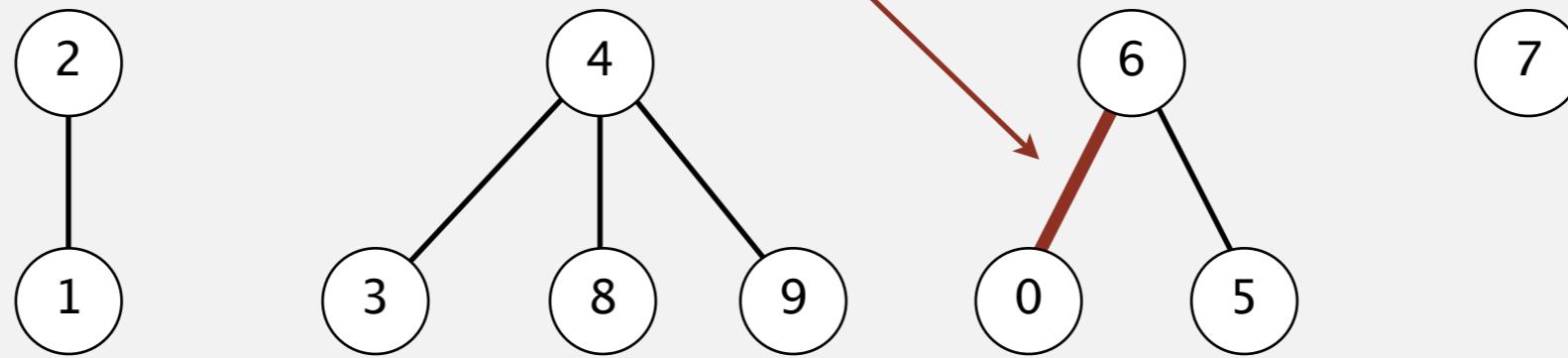


id[]	0	2	2	4	4	6	6	7	4	4
0	0	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

union(5, 0)

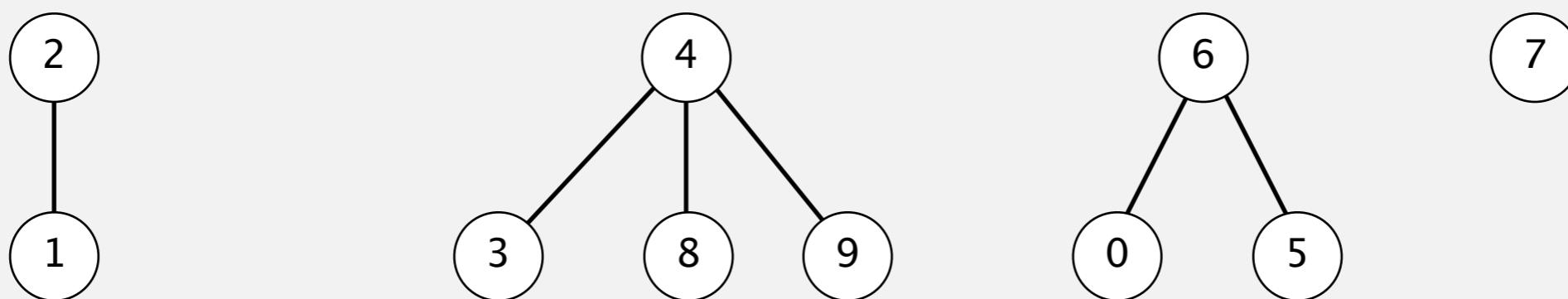
weighting: make 0 point to 6 (instead of 6 to 0)



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

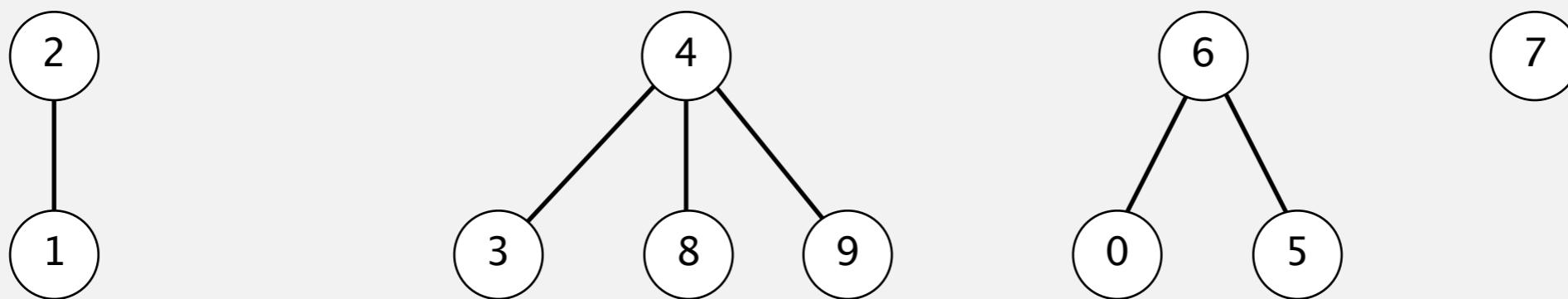


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

**union(7, 2)**

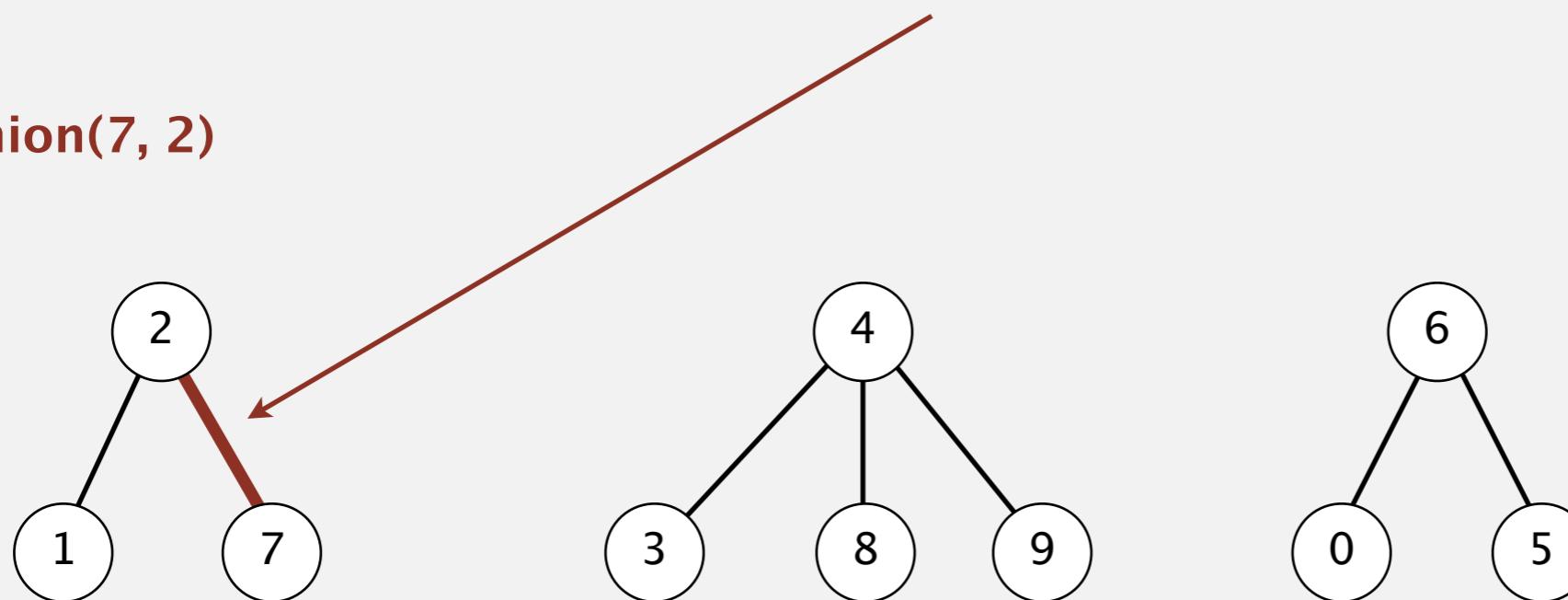


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

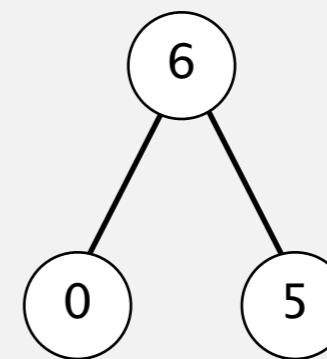
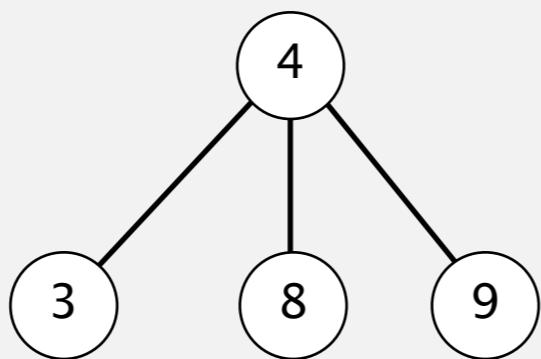
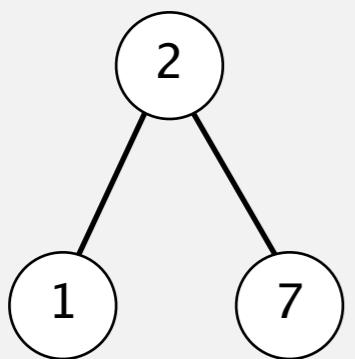
weighting: make 7 point to 2  
union(7, 2)



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

# Weighted quick-union demo

---

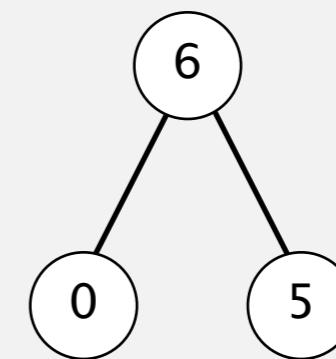
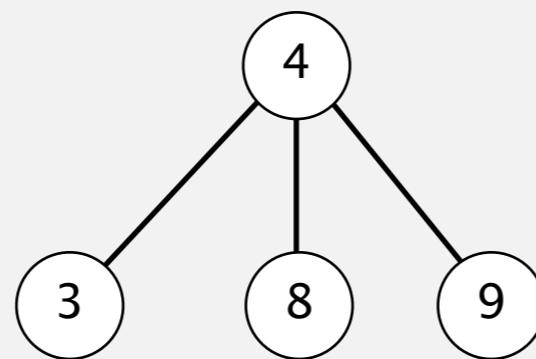
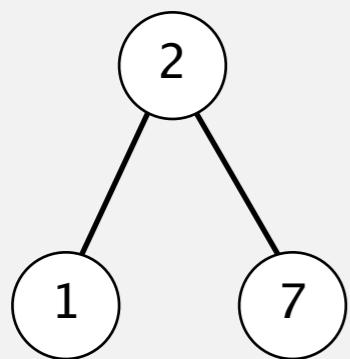


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	2	4	4

# Weighted quick-union demo

---

**union(6, 1)**

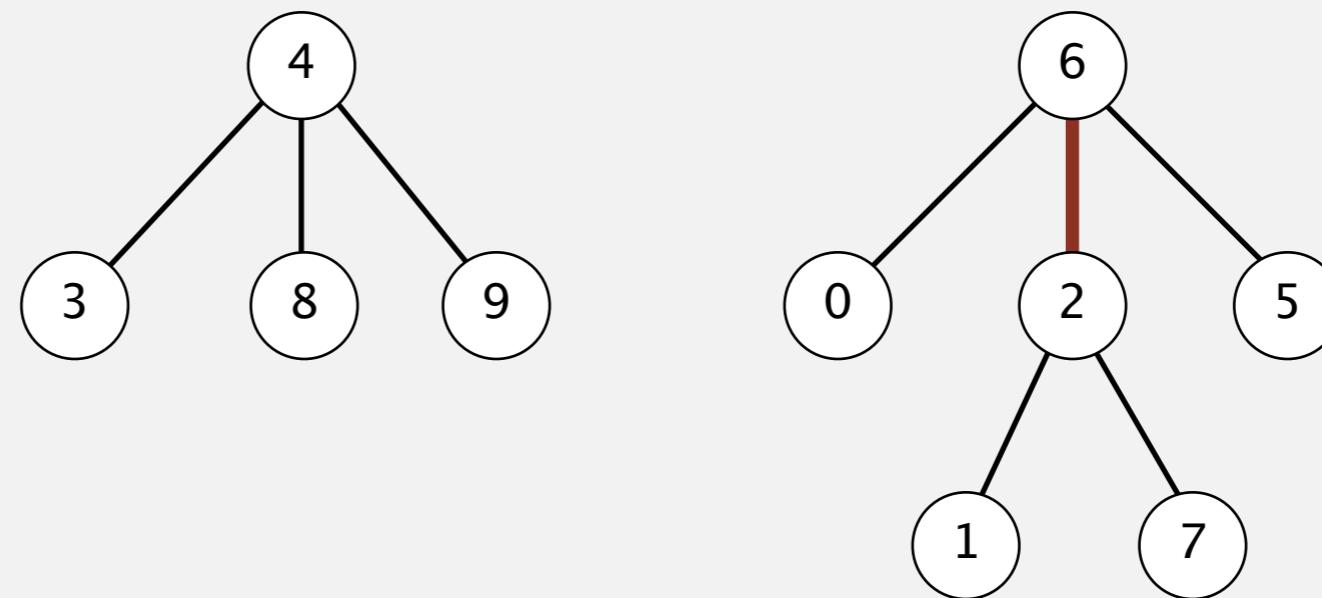


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	2	4	4

# Weighted quick-union demo

---

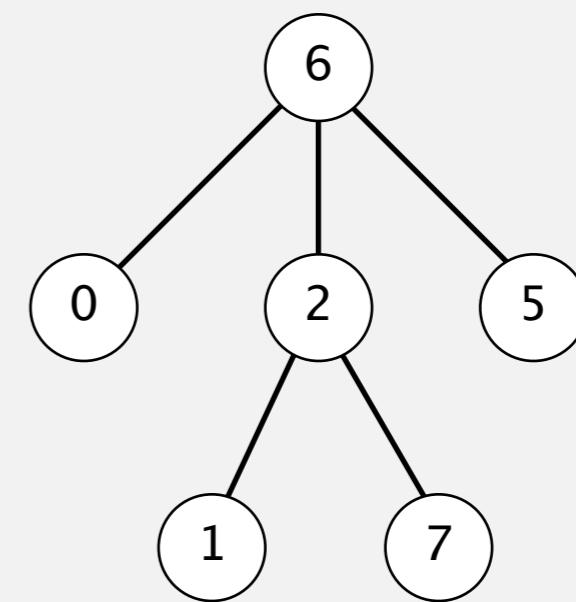
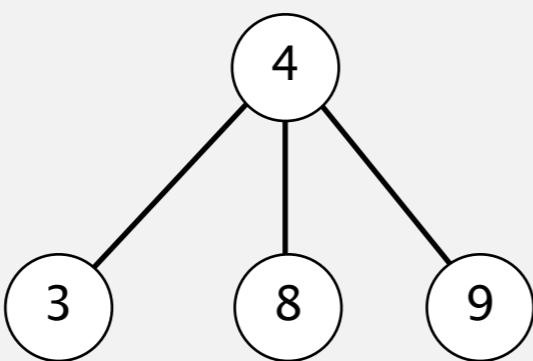
**union(6, 1)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	<b>6</b>	4	4	6	6	2	4	4

# Weighted quick-union demo

---

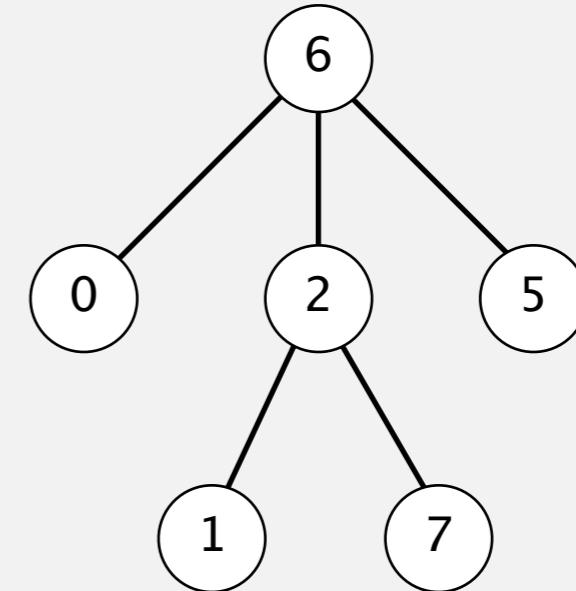
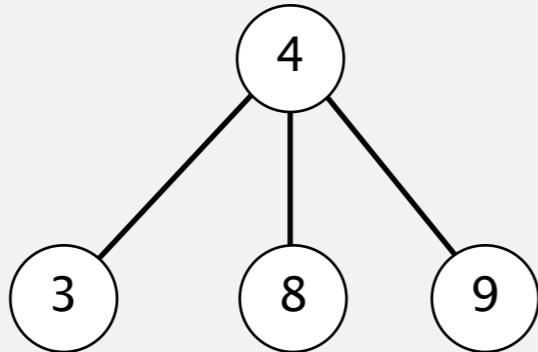


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	4	6	6	2	4	4

# Weighted quick-union demo

---

**union(7, 3)**



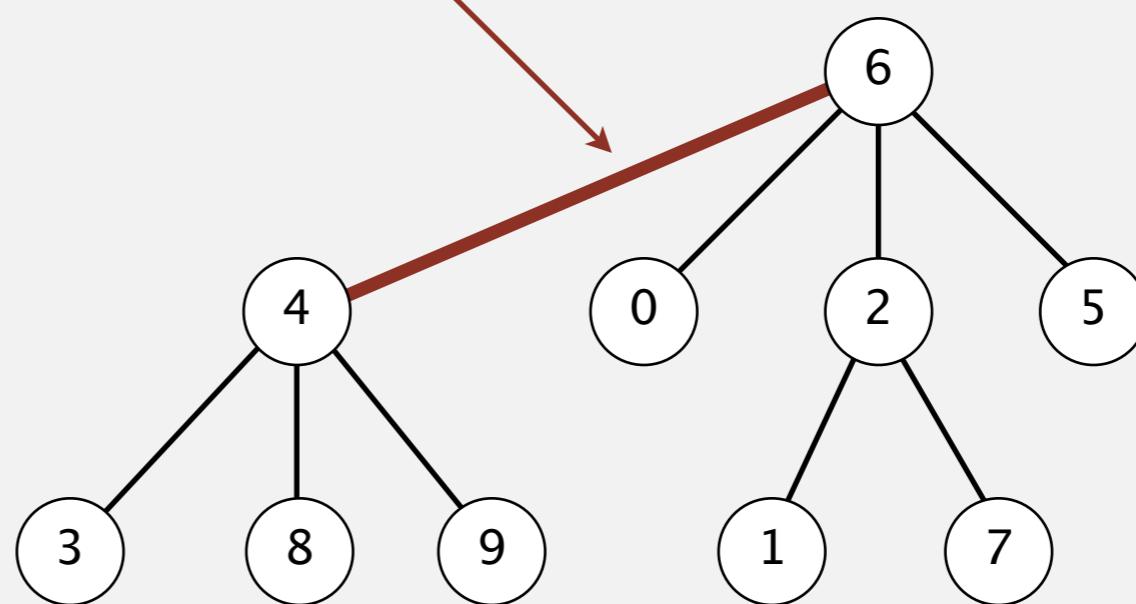
0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	4	6	6	2	4	4

# Weighted quick-union demo

---

union(7, 3)

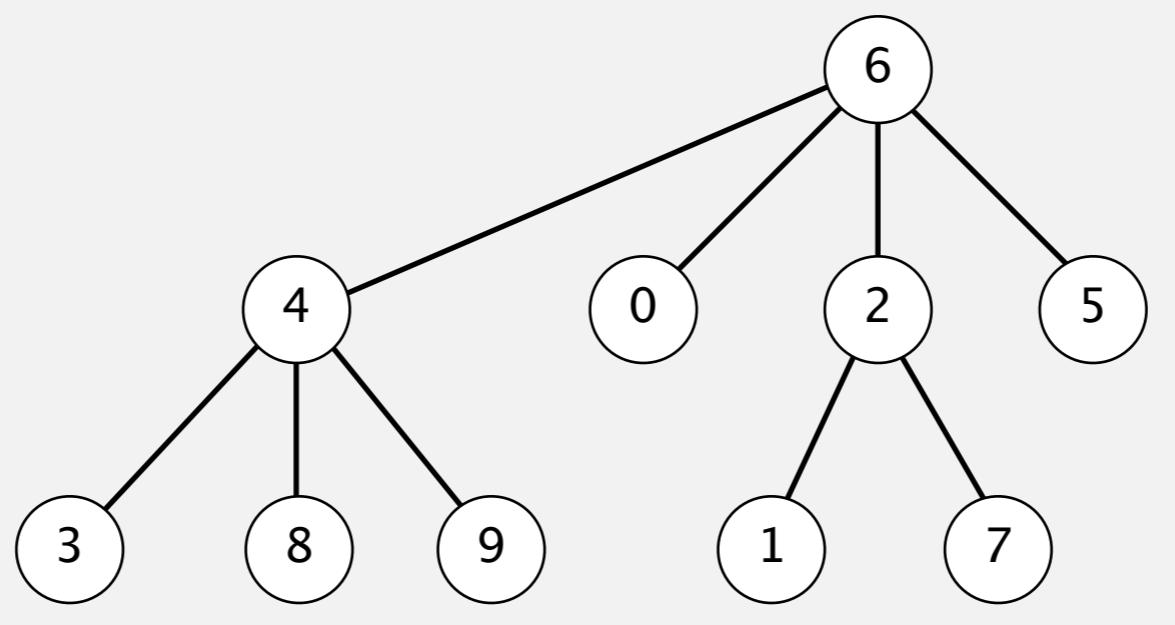
weighting: make 4 point to 6 (instead of 6 to 4)



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	6	6	6	2	4	4

# Weighted quick-union demo

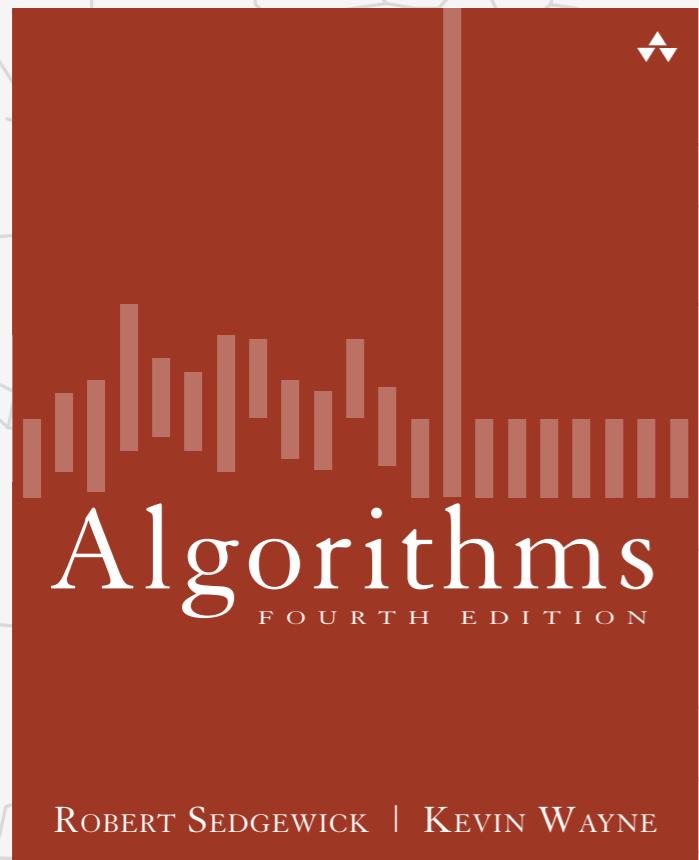
---



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	6	6	6	2	4	4

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



## 1.5 PATH COMPRESSION DEMO

---

**click to begin demo**

<http://algs4.cs.princeton.edu>

# Weighted quick-union with path compression demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Weighted quick-union with path compression demo

---

**union(4, 3)**

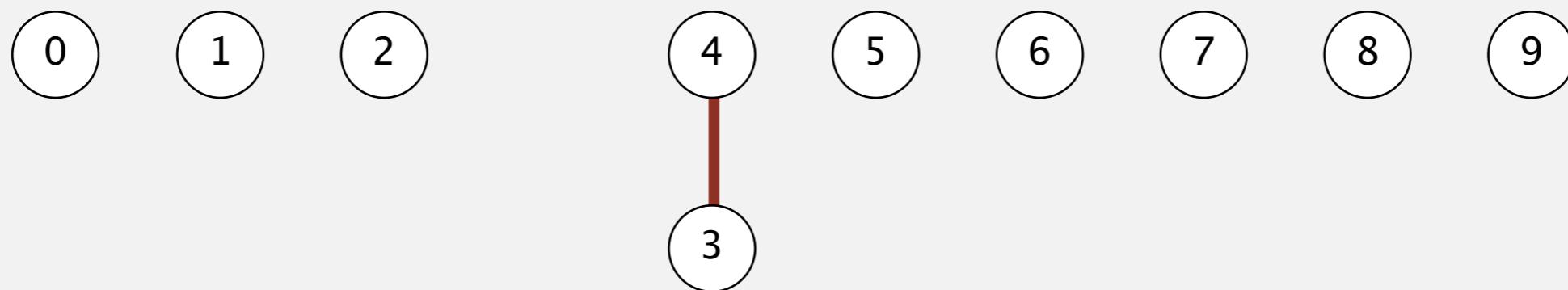


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Weighted quick-union with path compression demo

---

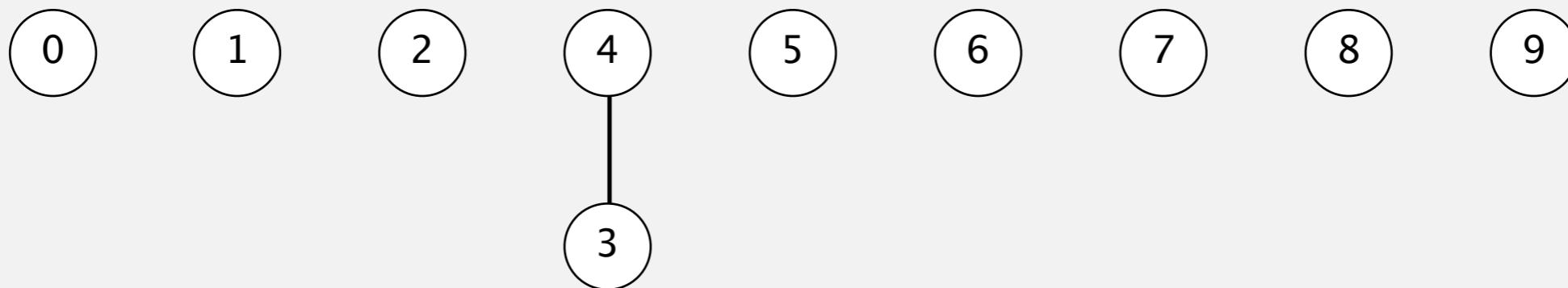
**union(4, 3)**



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

# Weighted quick-union with path compression demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	8	9

# Weighted quick-union with path compression demo

---

**union(3, 8)**

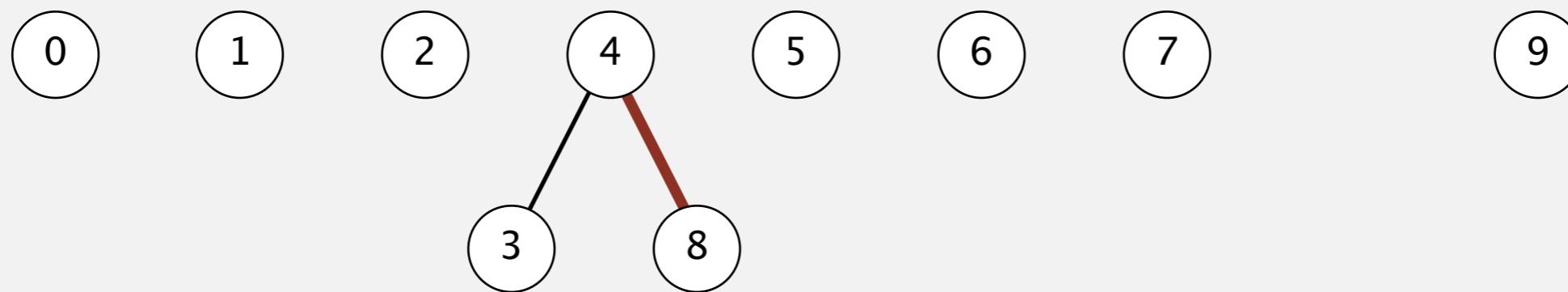


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	5	6	7	8	9

# Weighted quick-union with path compression demo

---

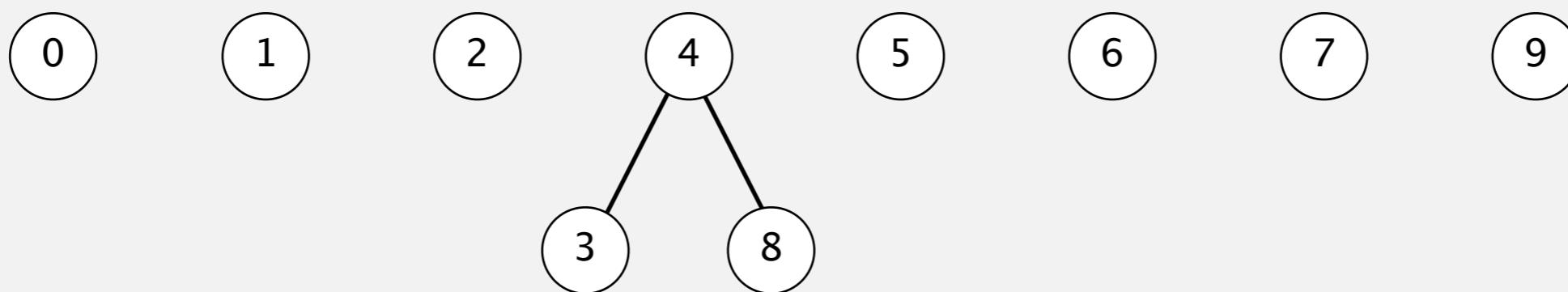
**union(3, 8)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union with path compression demo

---

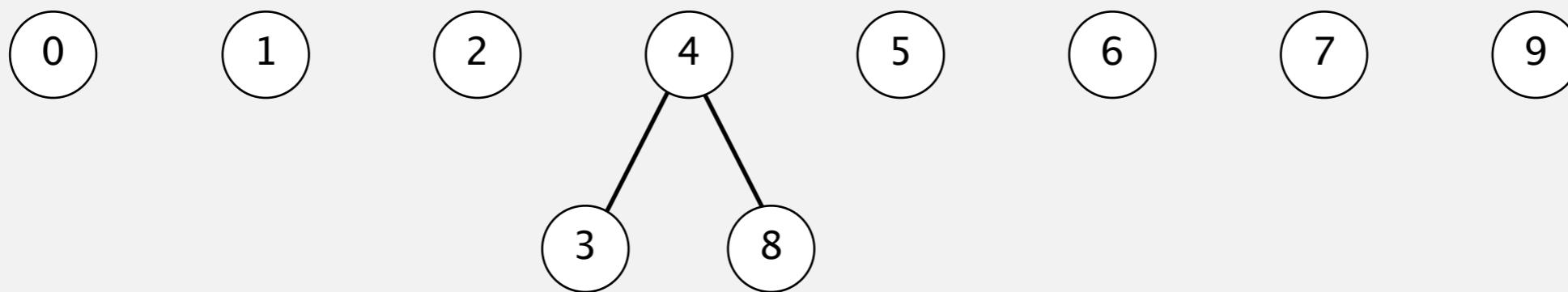


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union with path compression demo

---

**union(6, 5)**

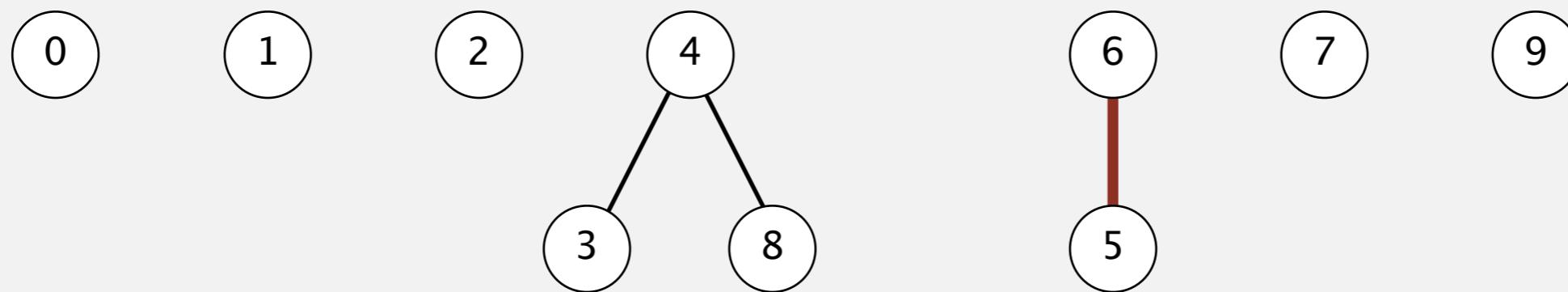


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union with path compression demo

---

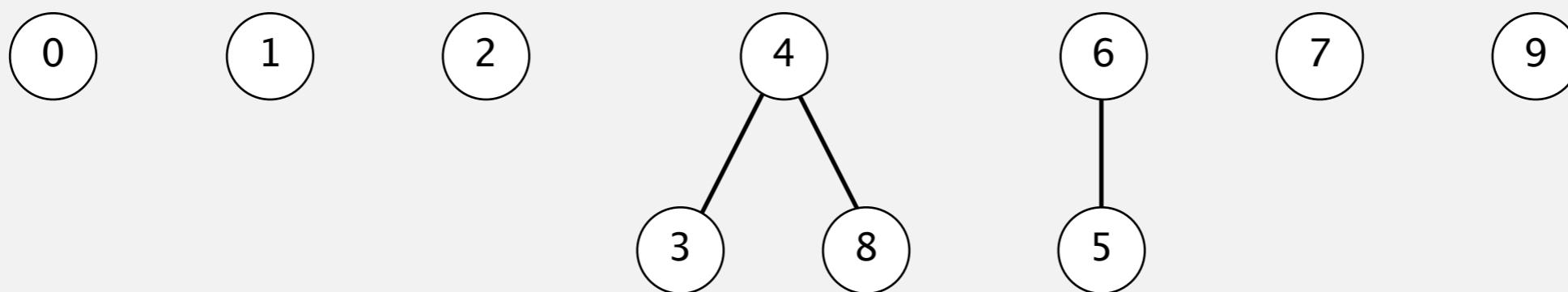
**union(6, 5)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	<b>6</b>	6	7	4	9

# Weighted quick-union with path compression demo

---

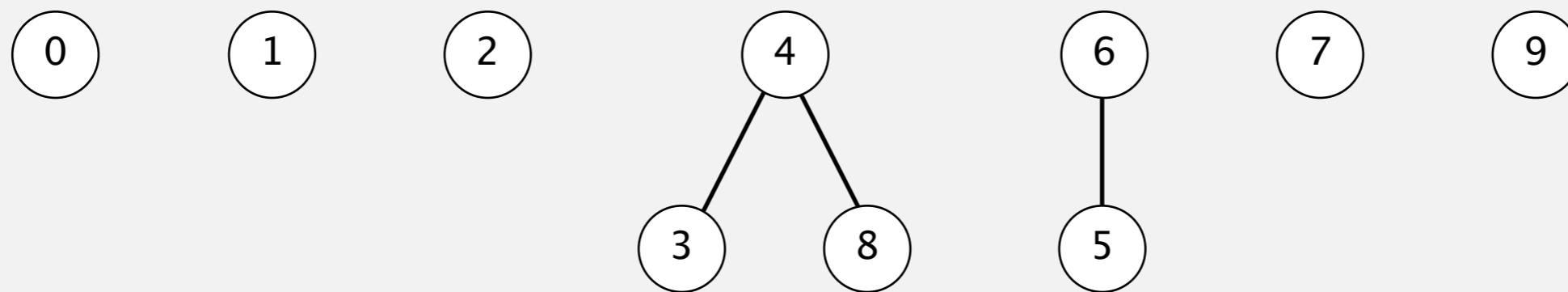


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	6	6	7	4	9

# Weighted quick-union with path compression demo

---

**union(9, 4)**

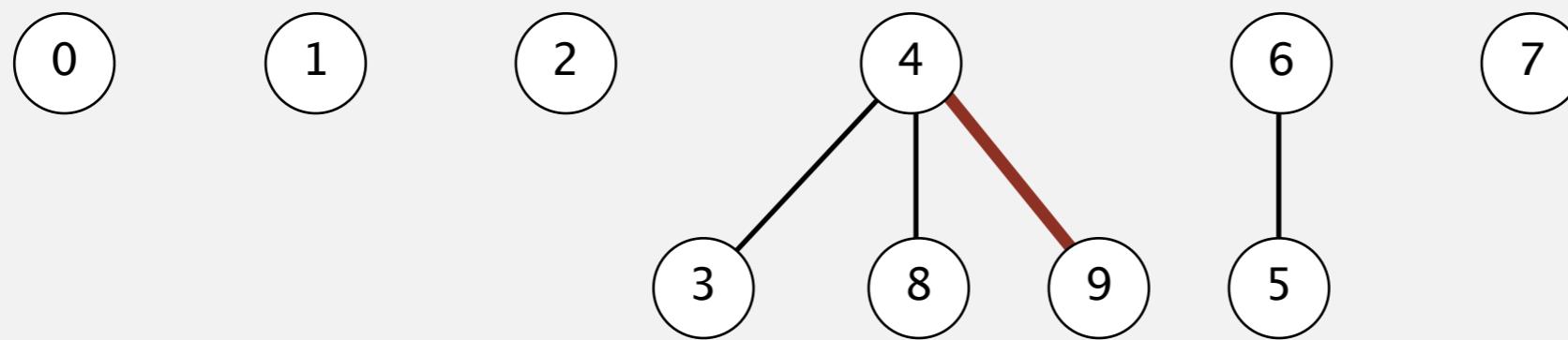


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	6	6	7	4	9

# Weighted quick-union with path compression demo

---

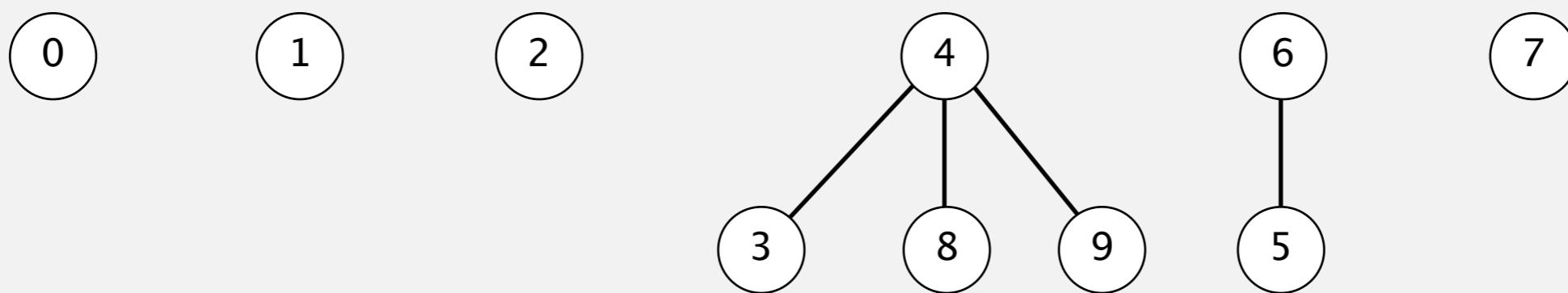
**union(9, 4)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	1	2	4	4	6	6	7	4	<b>4</b>

# Weighted quick-union with path compression demo

---

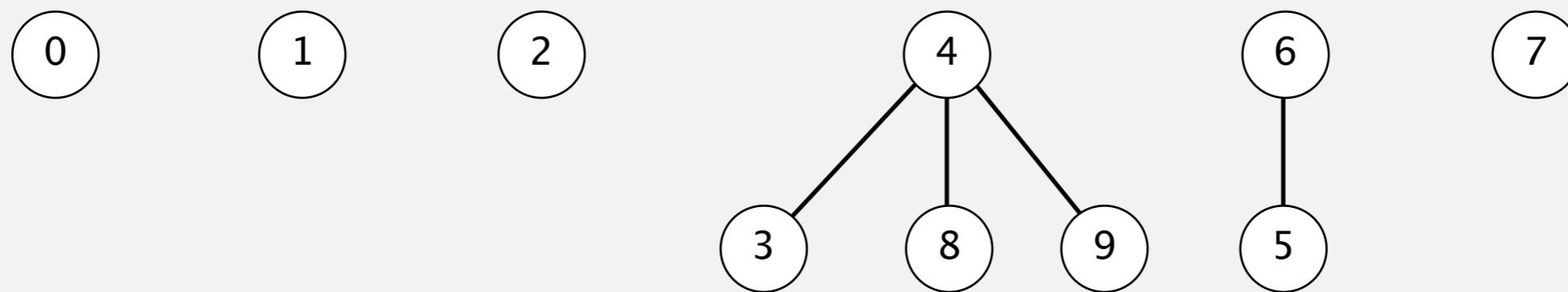


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union with path compression demo

---

**union(2, 1)**

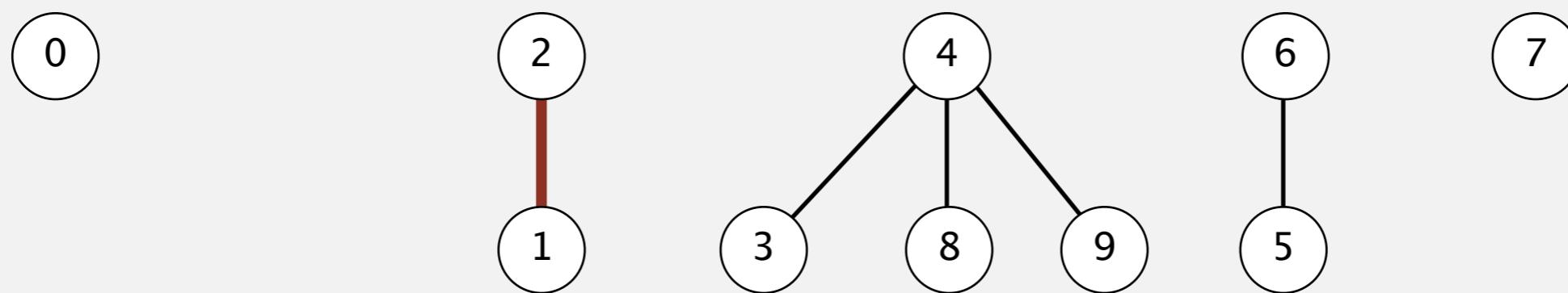


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union with path compression demo

---

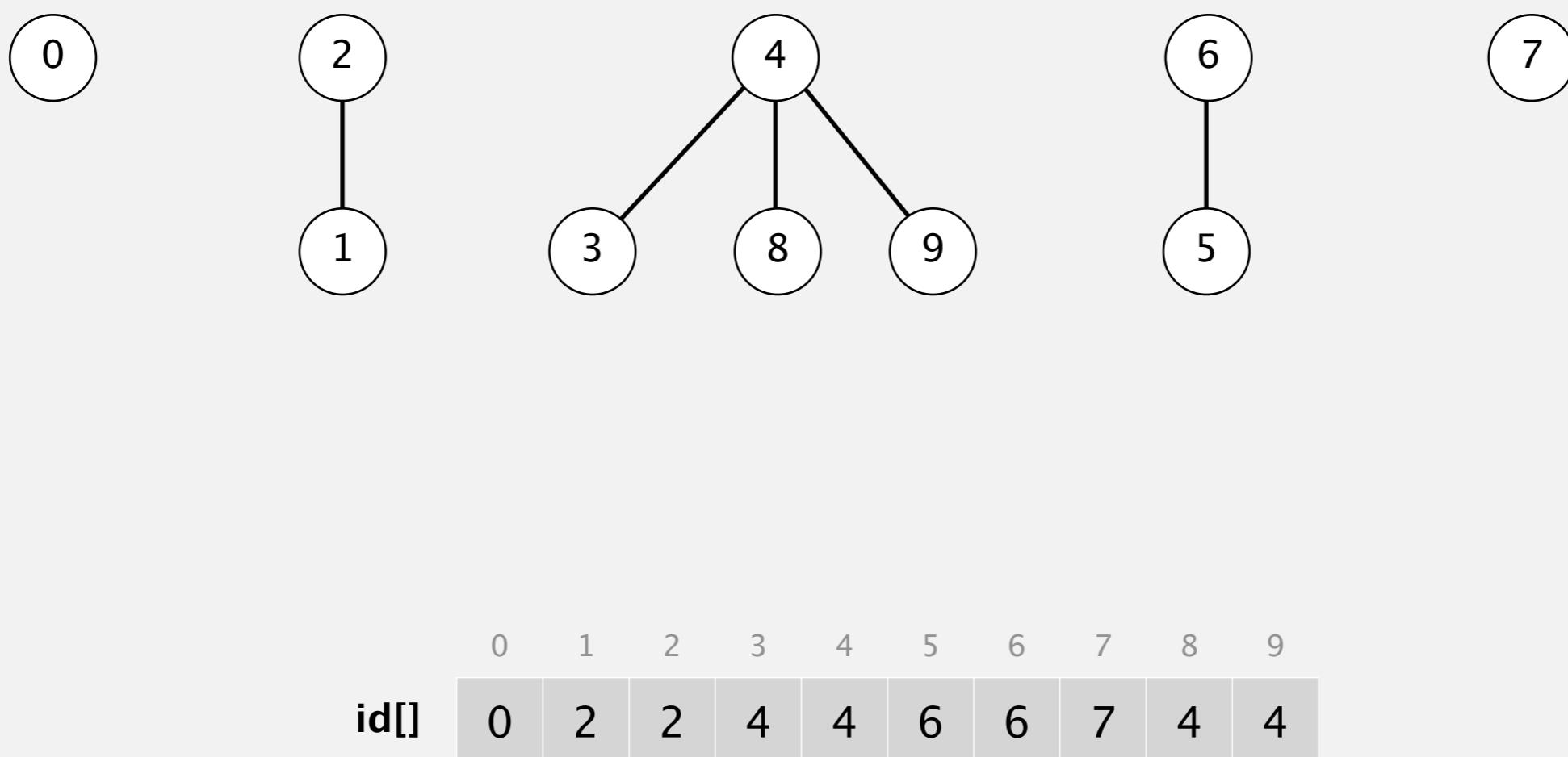
**union(2, 1)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	2	2	4	4	6	6	7	4	4

# Weighted quick-union with path compression demo

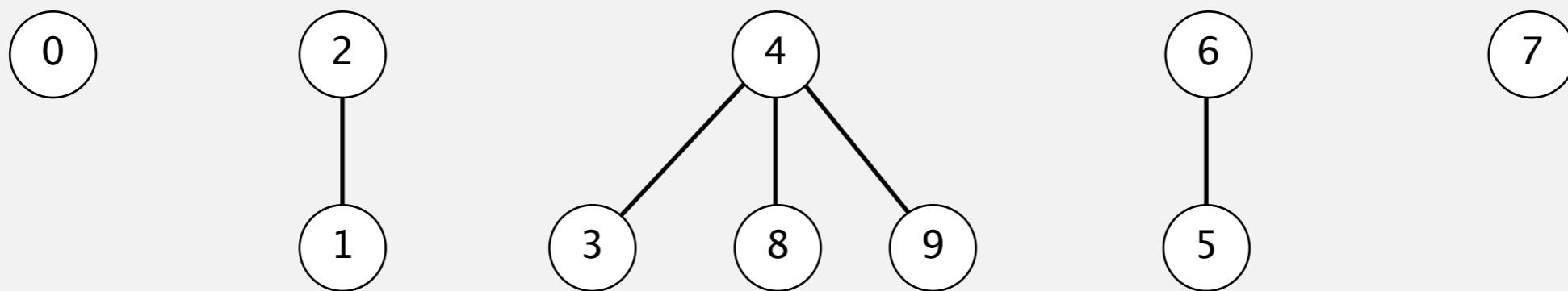
---



# Weighted quick-union with path compression demo

---

**union(5, 0)**

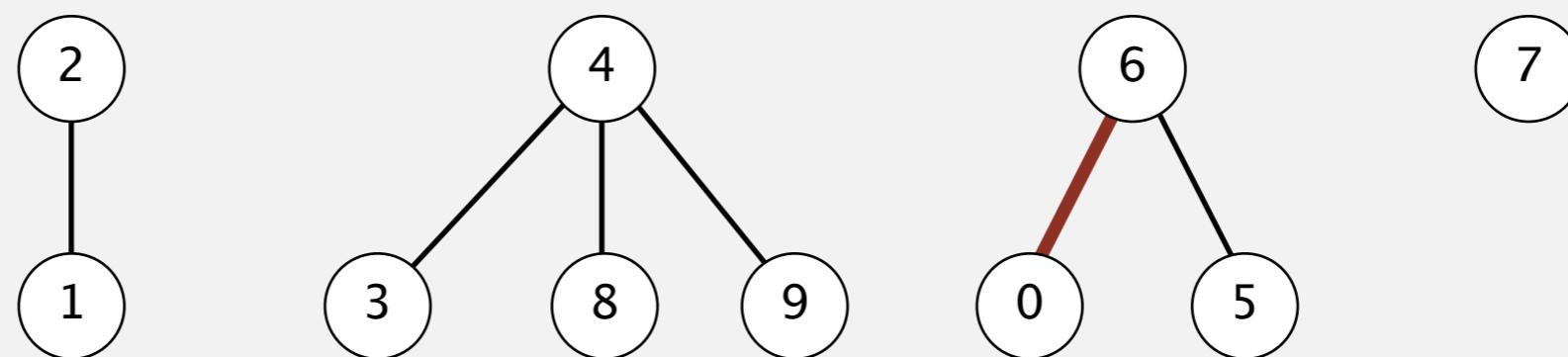


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	0	2	2	4	4	6	6	7	4	4

# Weighted quick-union with path compression demo

---

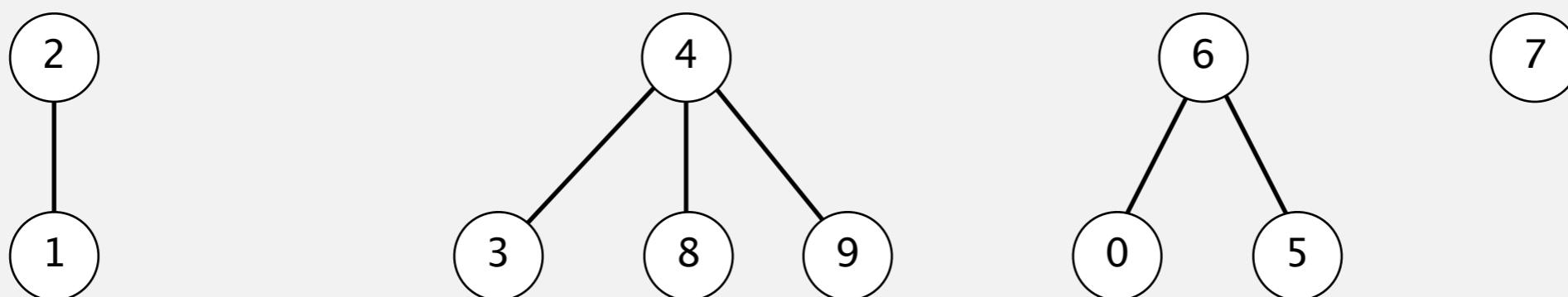
**union(5, 0)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	7	4	4

# Weighted quick-union with path compression demo

---

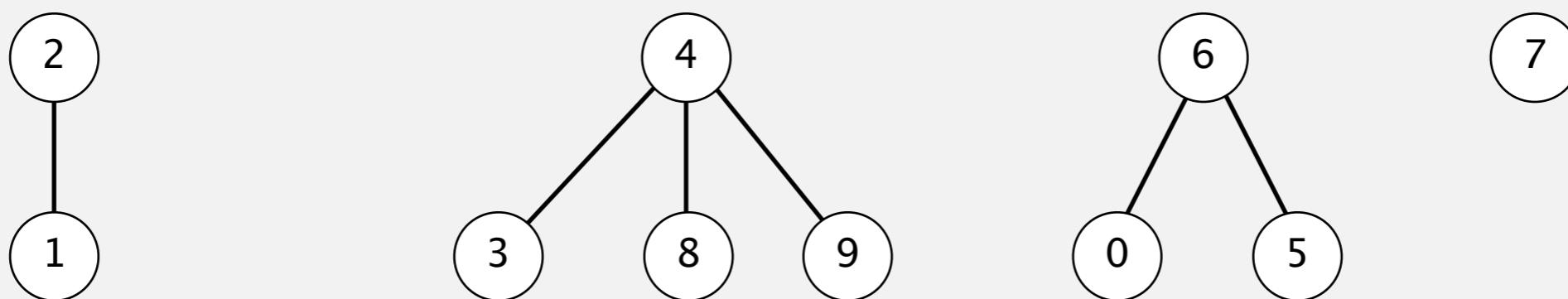


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	7	4	4

# Weighted quick-union with path compression demo

---

**union(7, 2)**

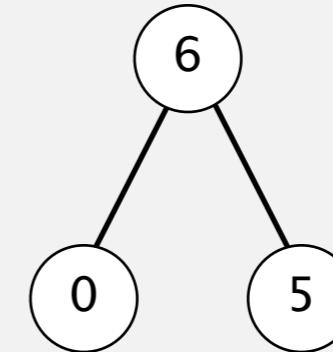
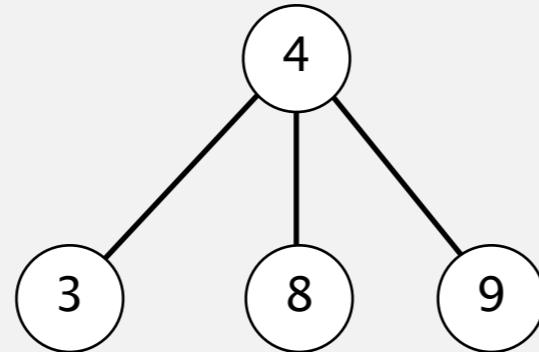
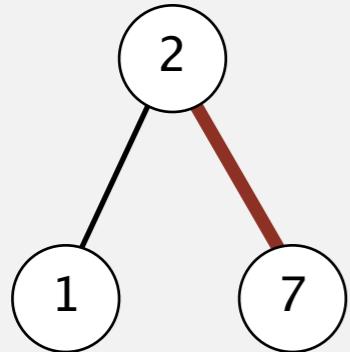


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	7	4	4

# Weighted quick-union with path compression demo

---

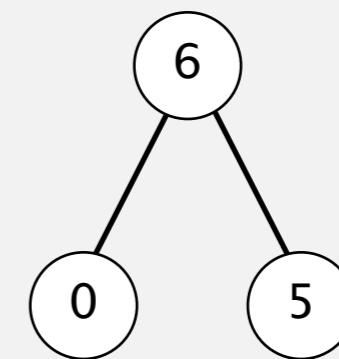
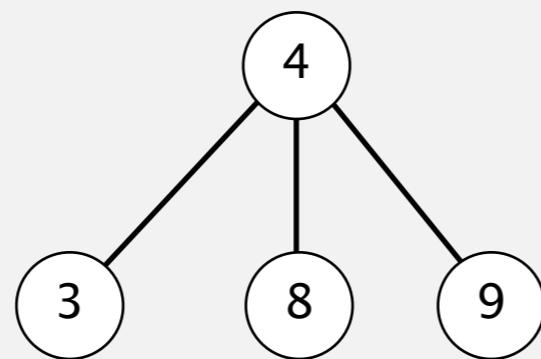
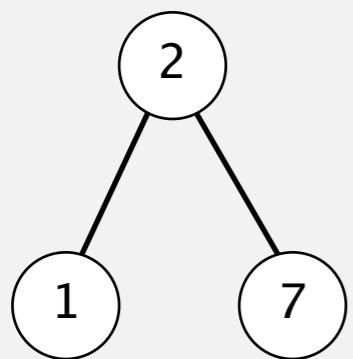
**union(7, 2)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	2	4	4

# Weighted quick-union with path compression demo

---

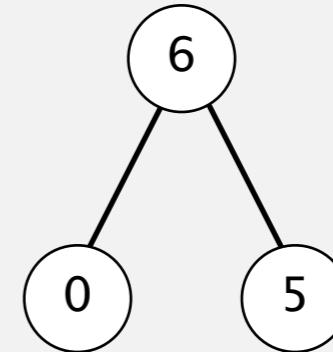
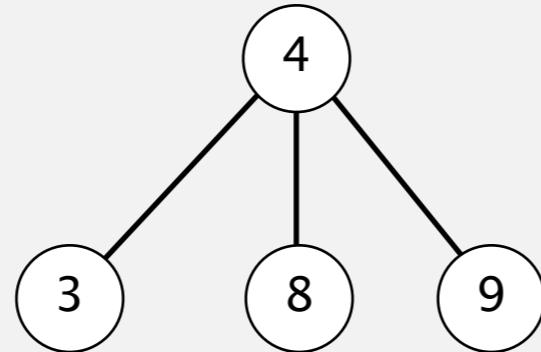
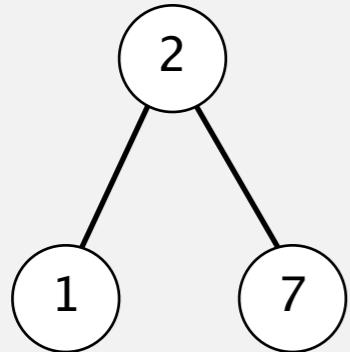


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	2	4	4

# Weighted quick-union with path compression demo

---

**union(6, 1)**

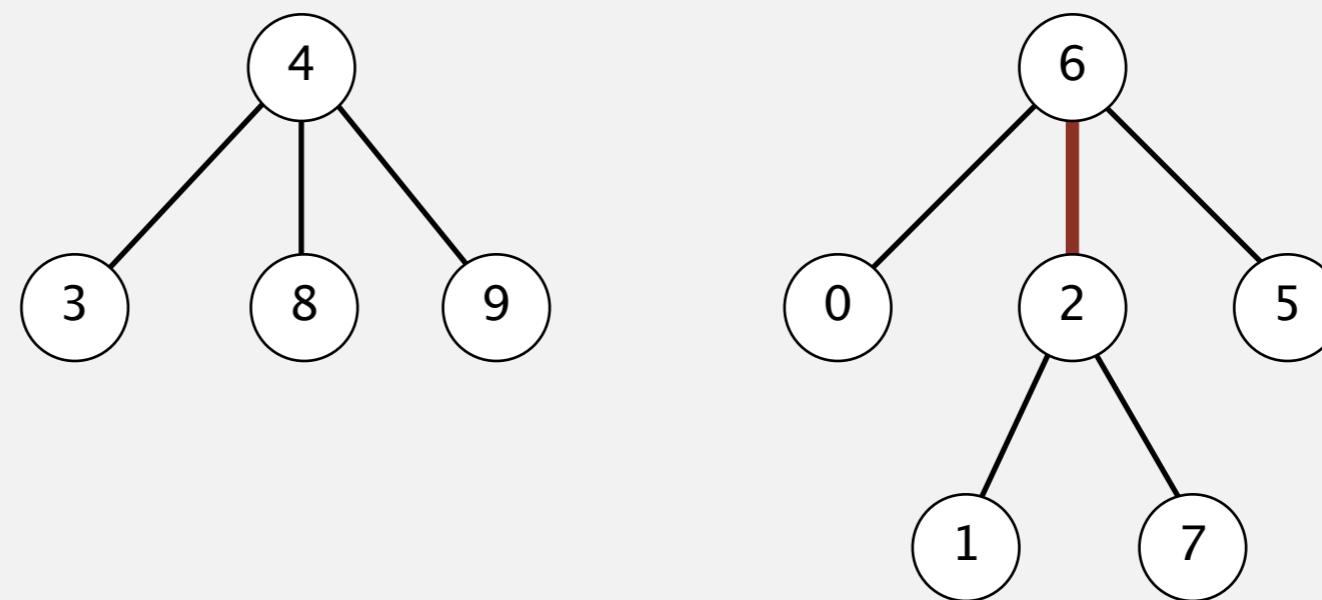


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	6	2	2	4	4	6	6	2	4	4

# Weighted quick-union with path compression demo

---

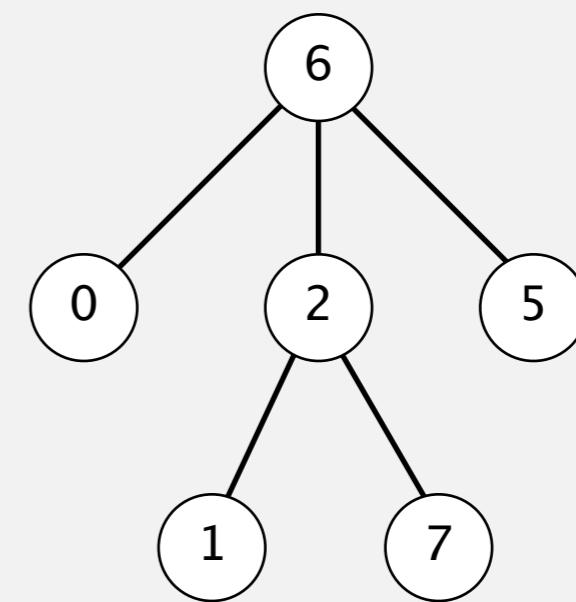
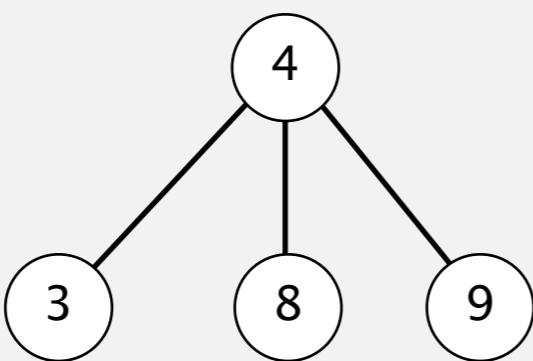
**union(6, 1)**



0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	<b>6</b>	4	4	6	6	2	4	4

# Weighted quick-union with path compression demo

---

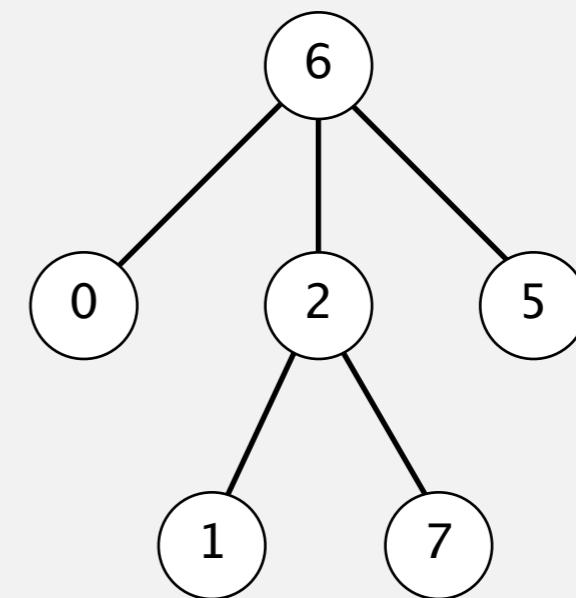
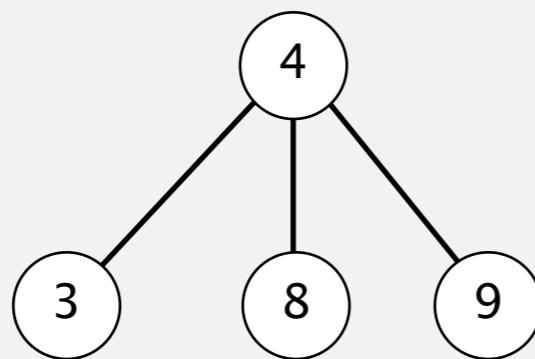


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	4	6	6	2	4	4

# Weighted quick-union with path compression demo

---

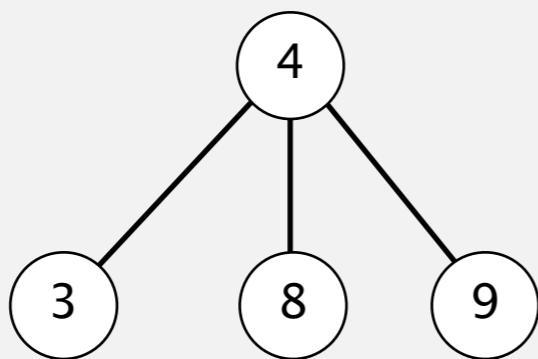
**union(7, 3)**



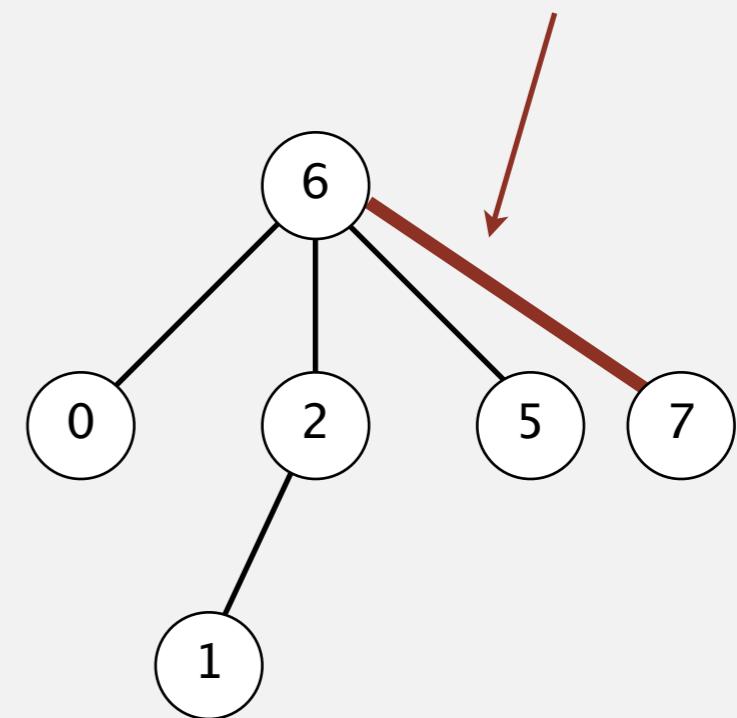
0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	4	6	6	2	4	4

# Weighted quick-union with path compression demo

union(7, 3)



path compression:  
make 7 point to 6

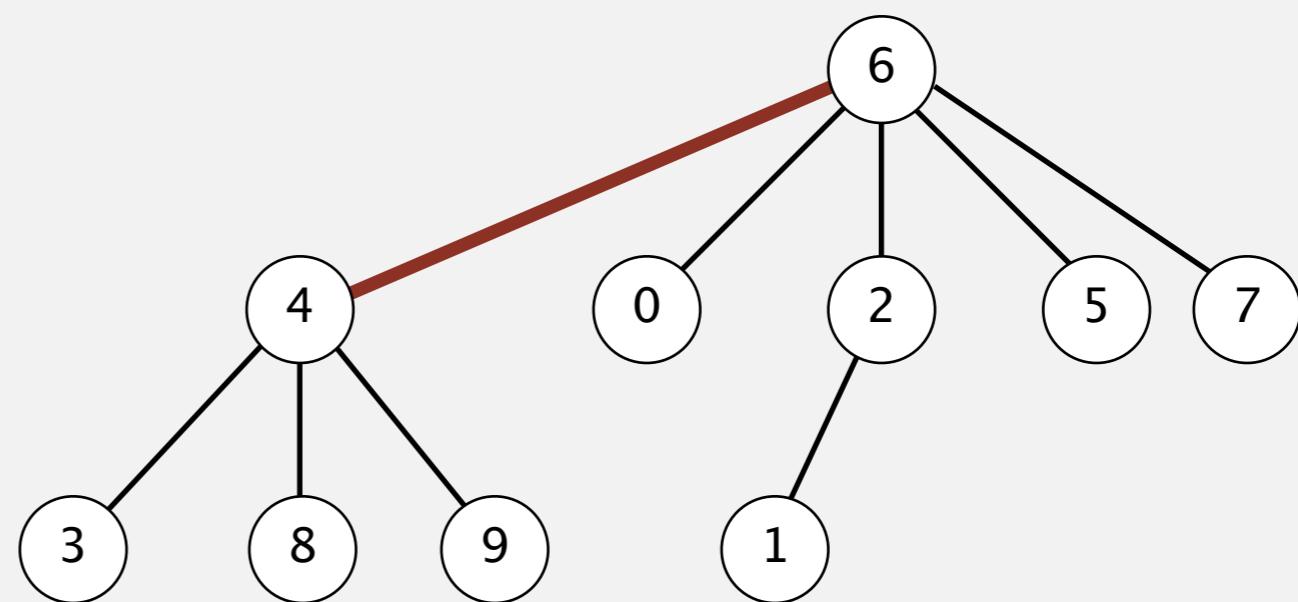


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	4	6	6	<b>6</b>	4	4

# Weighted quick-union with path compression demo

---

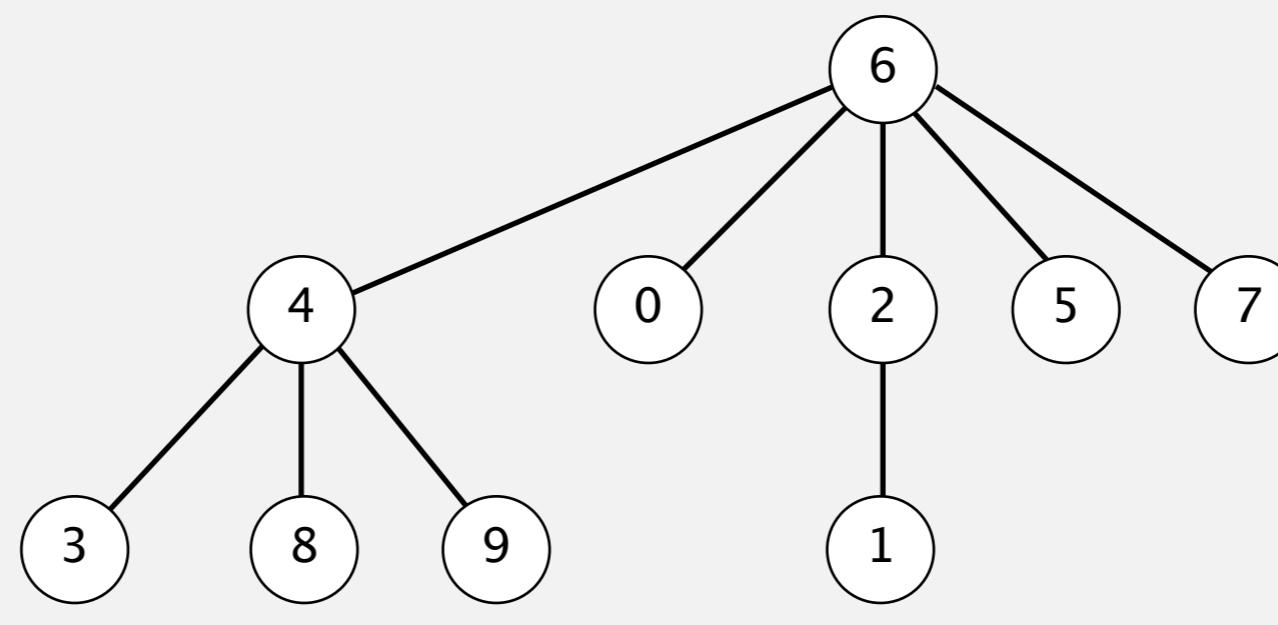
**union(7, 3)**



0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	6	2	6	4	6	6	6	4	4

# Weighted quick-union with path compression demo

---

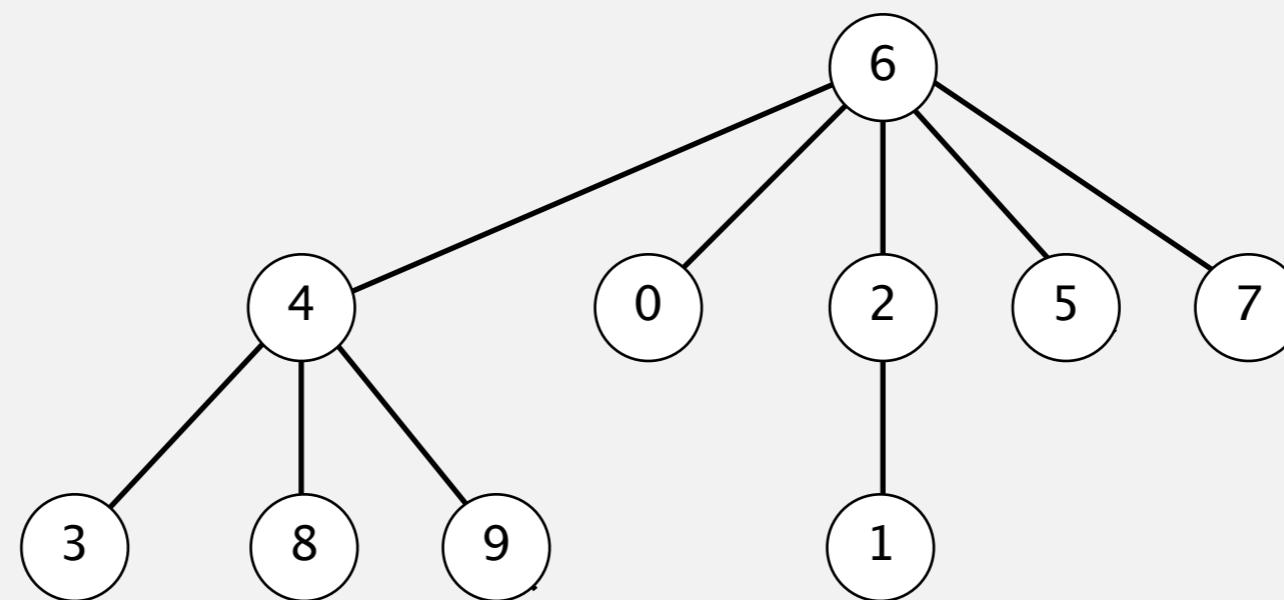


0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	6	2	6	4	6	6	6	4	4

# Weighted quick-union with path compression demo

---

connected(9, 1)



0 1 2 3 4 5 6 7 8 9

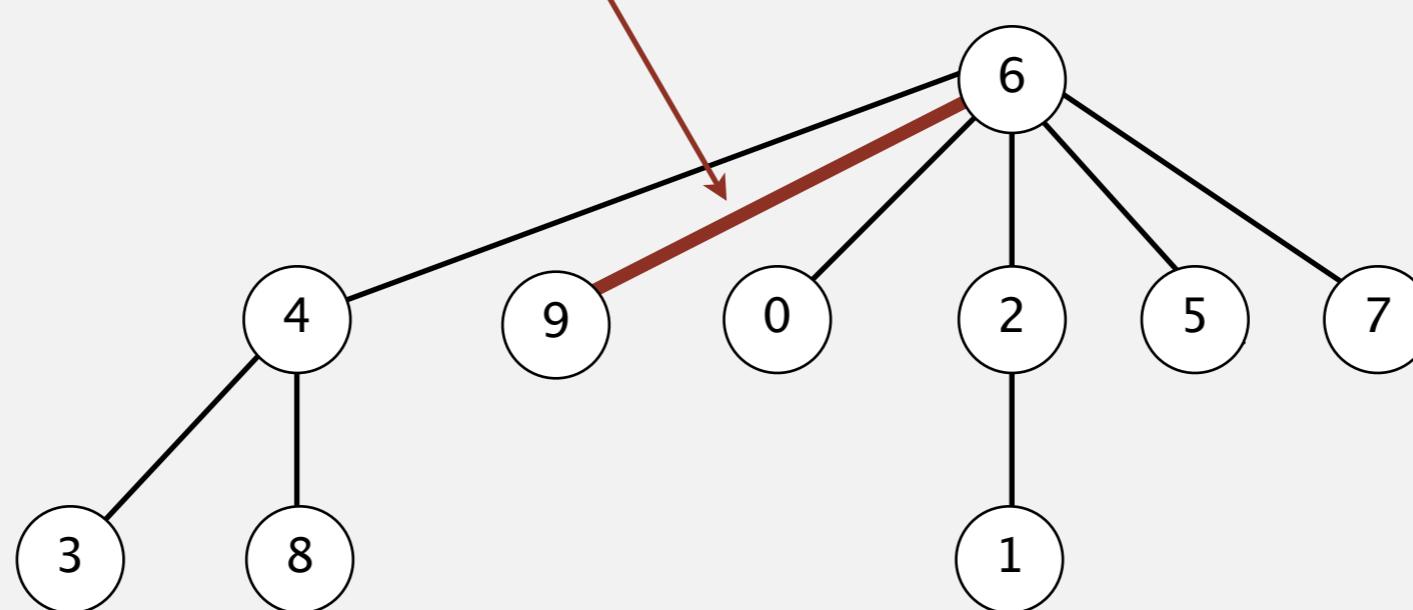
id[]	6	2	6	4	6	6	6	4	4
------	---	---	---	---	---	---	---	---	---

# Weighted quick-union with path compression demo

connected(9, 1)

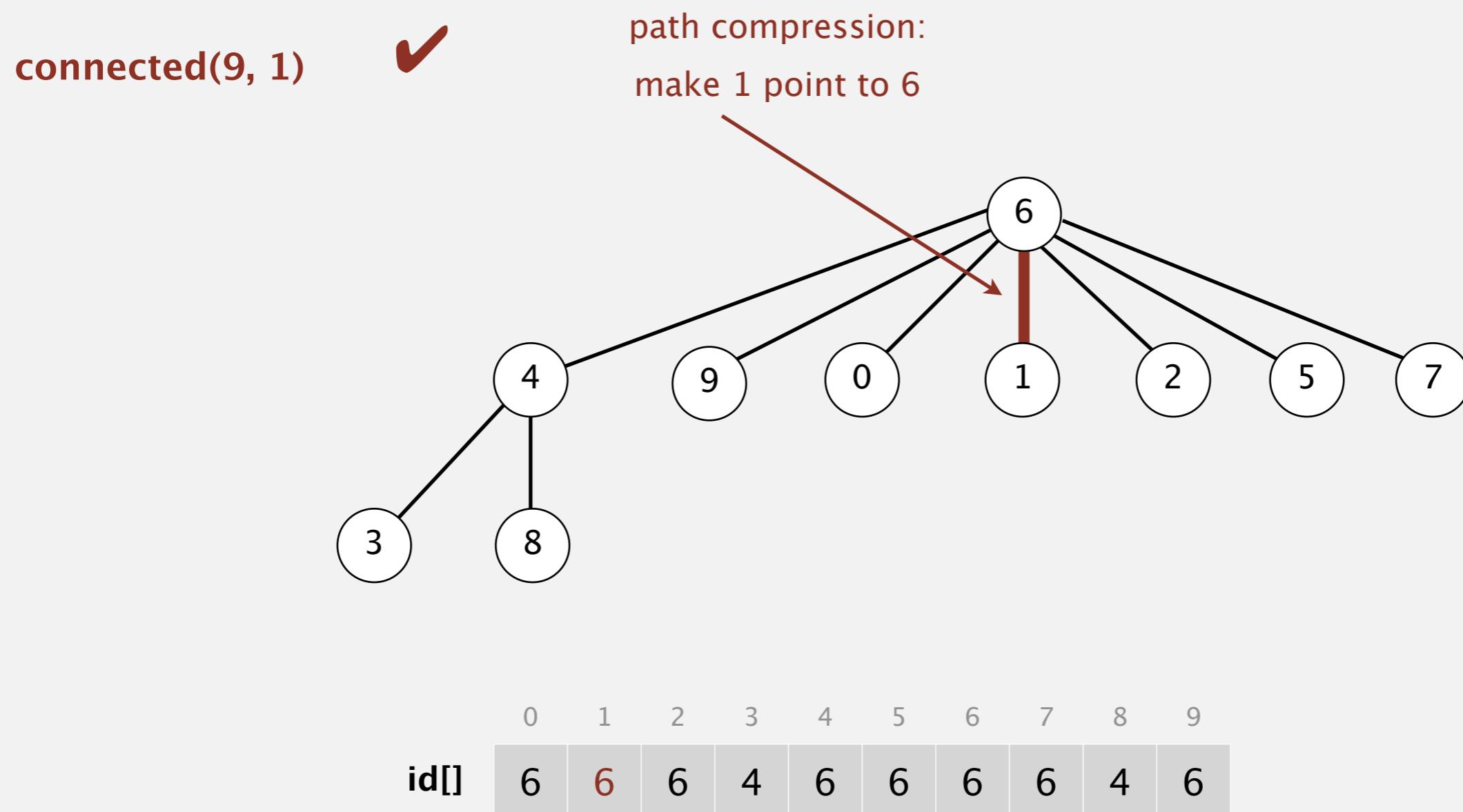


path compression:  
make 9 point to 6



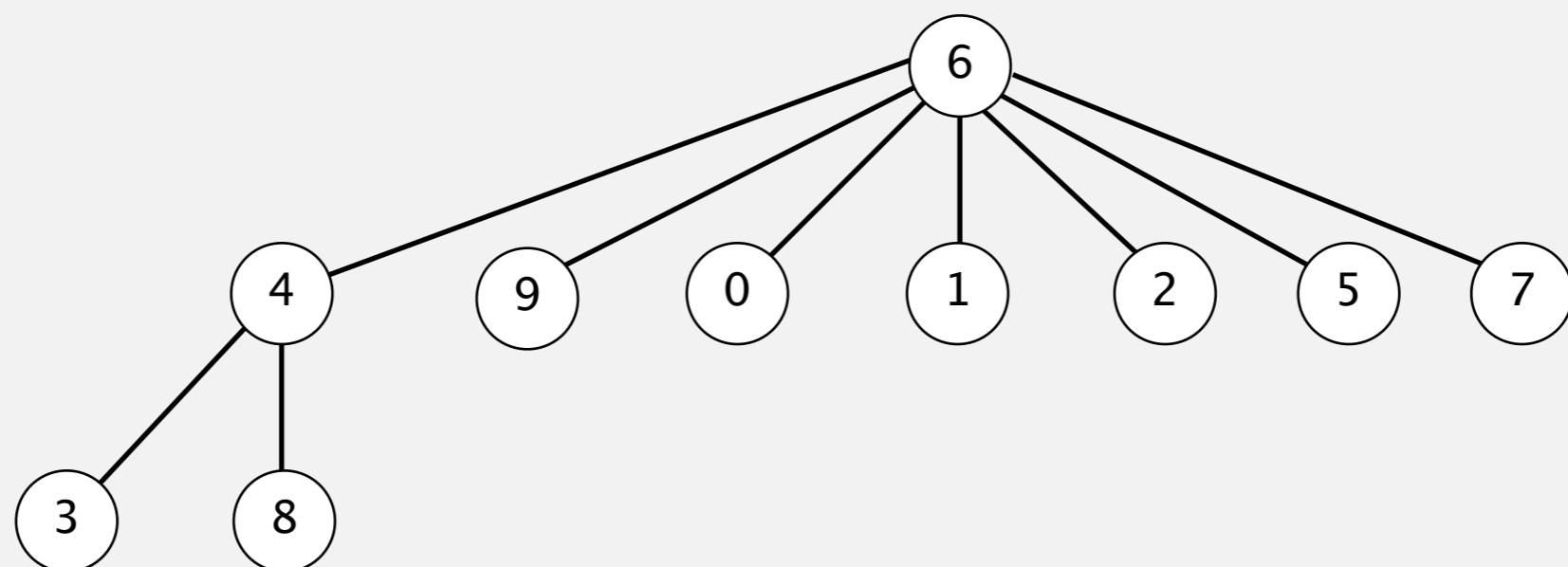
0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	6	2	6	4	6	6	6	4	<b>6</b>

# Weighted quick-union with path compression demo



# Weighted quick-union with path compression demo

---



0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	6	6	6	4	6	6	6	4	6

## Kruskal's algorithm: running time

---

**Proposition.** Kruskal's algorithm computes MST in time proportional to  $E \log E$  (in the worst case).

## Kruskal's algorithm: running time

---

**Proposition.** Kruskal's algorithm computes MST in time proportional to  $E \log E$  (in the worst case).

Pf.

operation	frequency	time per op
<b>build pq</b>	1	$E$
<b>delete-min</b>	$E$	$\log E$
<b>union</b>	$V$	$\log^* V$ †
<b>connected</b>	$E$	$\log^* V$ †

† amortized bound using weighted quick union with path compression

## Kruskal's algorithm: running time

---

**Proposition.** Kruskal's algorithm computes MST in time proportional to  $E \log E$  (in the worst case).

Pf.

operation	frequency	time per op
<b>build pq</b>	1	$E$
<b>delete-min</b>	$E$	$\log E$
<b>union</b>	$V$	$\log^* V^\dagger$
<b>connected</b>	$E$	$\log^* V^\dagger$

† amortized bound using weighted quick union with path compression

recall:  $\log^* V \leq 5$  in this universe



**Remark.** If edges are already sorted, order of growth is  $E \log^* V$ .

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle

# Does a linear-time MST algorithm exist?

---

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	<i>optimal</i>	Pettie-Ramachandran

# Does a linear-time MST algorithm exist?

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	<i>optimal</i>	Pettie-Ramachandran
20xx	$E$	???



# Does a linear-time MST algorithm exist?

## deterministic compare-based MST algorithms

year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	<i>optimal</i>	Pettie-Ramachandran
20xx	$E$	???



Remark. Linear-time randomized MST algorithm (Karger-Klein-Tarjan 1995).