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?

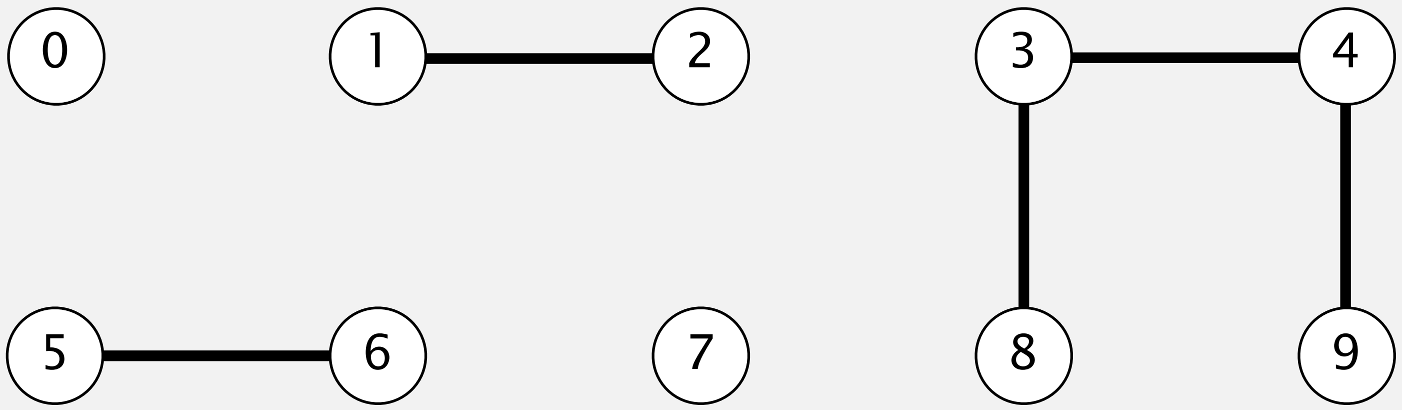
connect 4 and 3

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

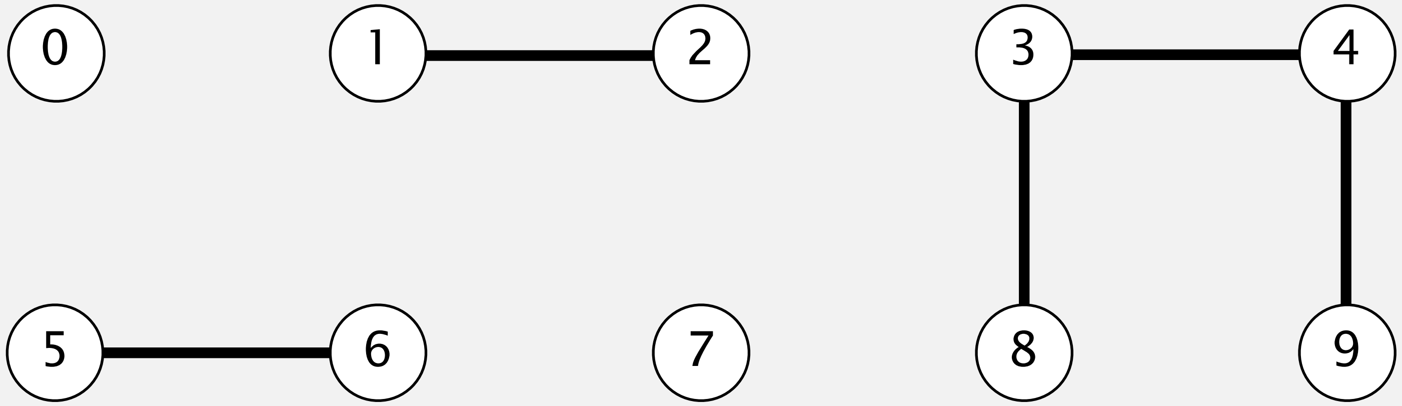
connect 3 and 8

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

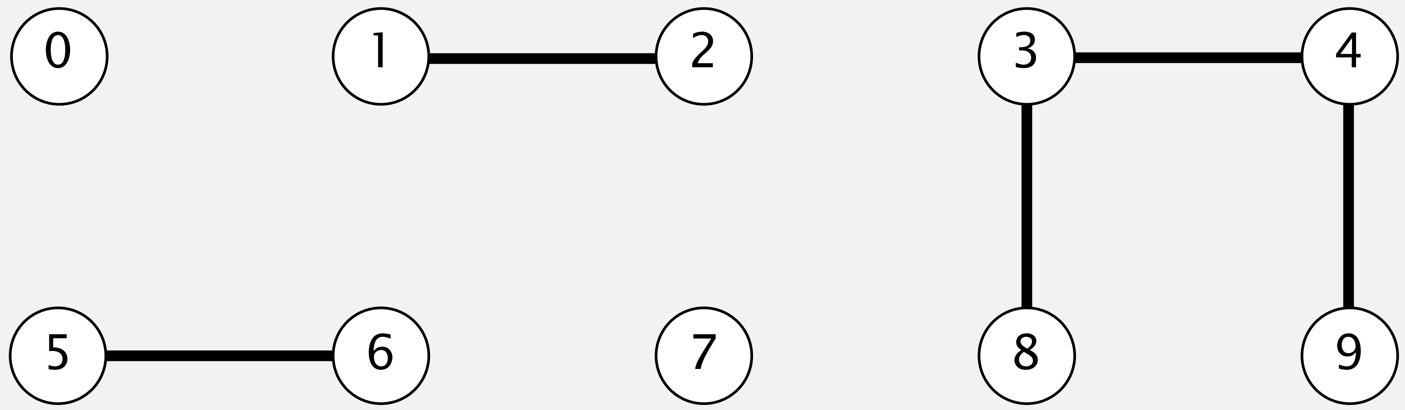
connect 3 and 8

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

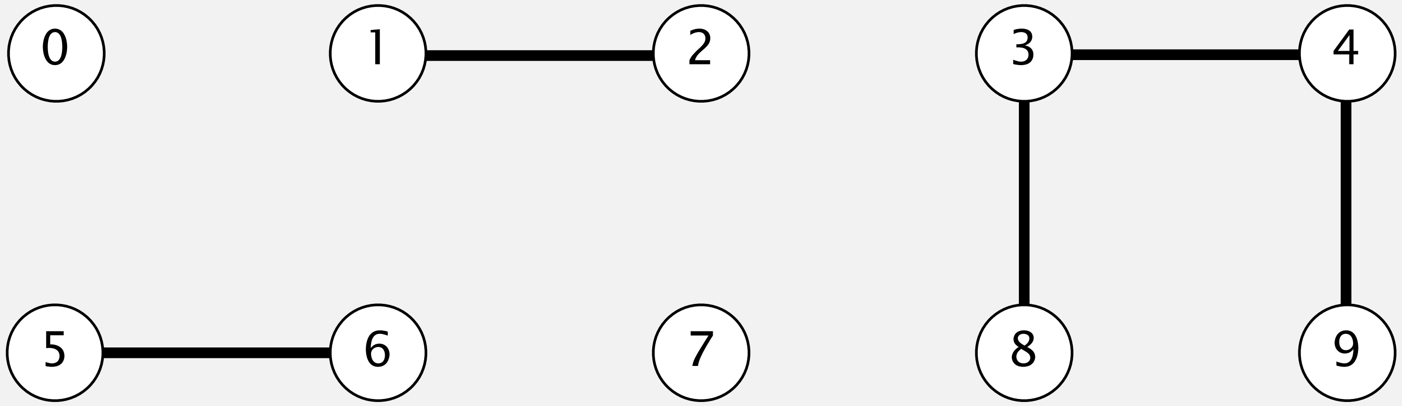
connect 6 and 5

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

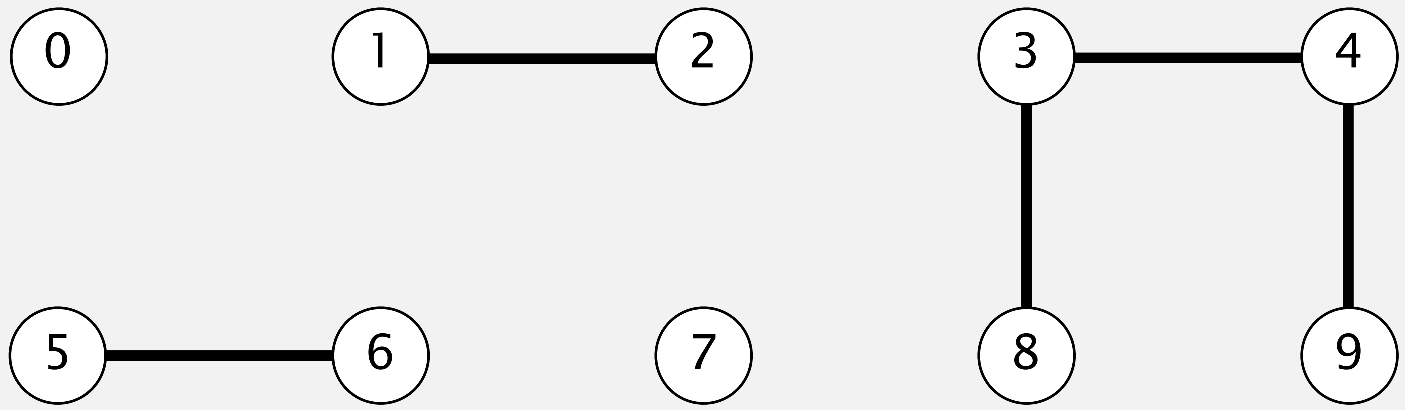
connect 6 and 5

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

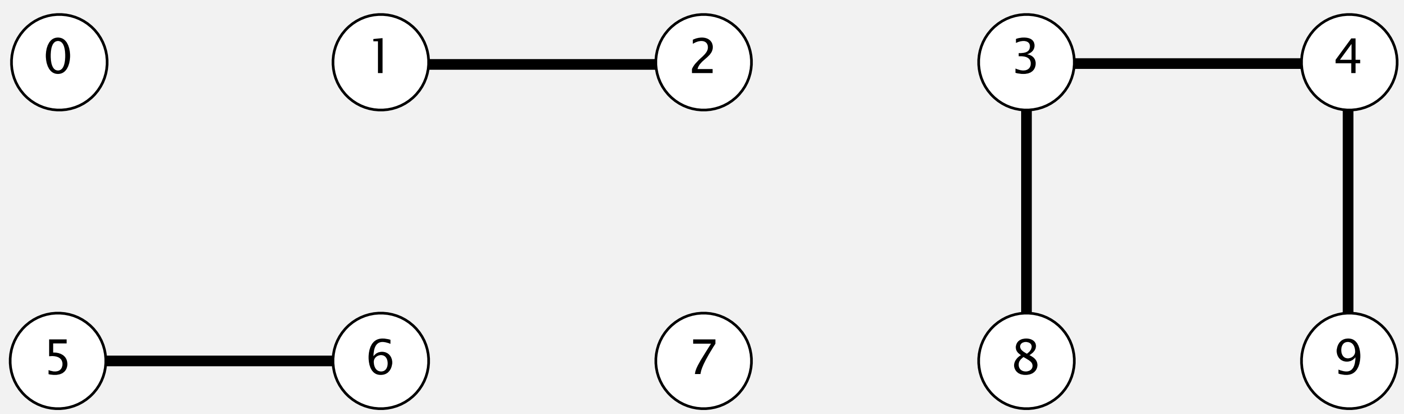
connect 9 and 4

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

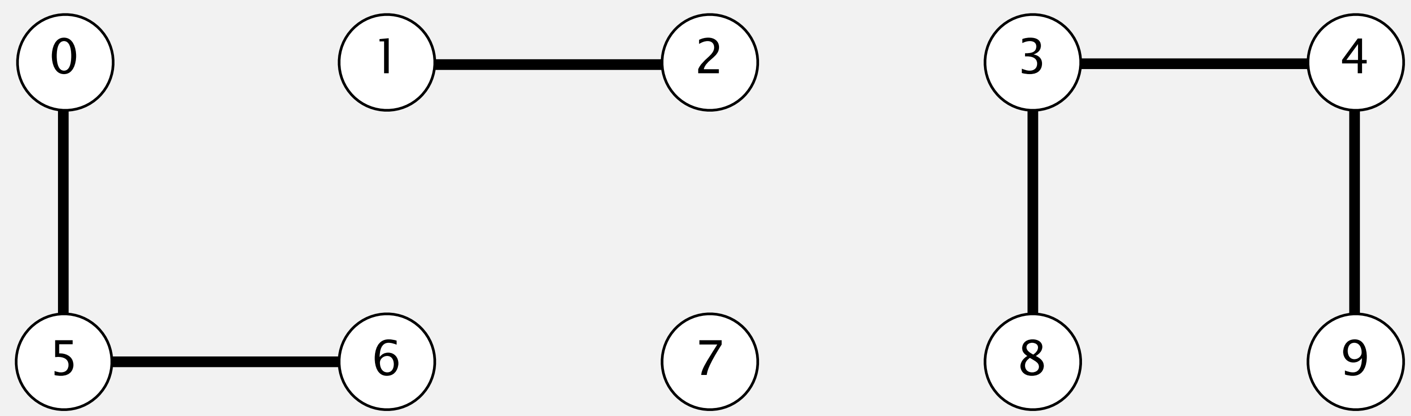
connect 9 and 4

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

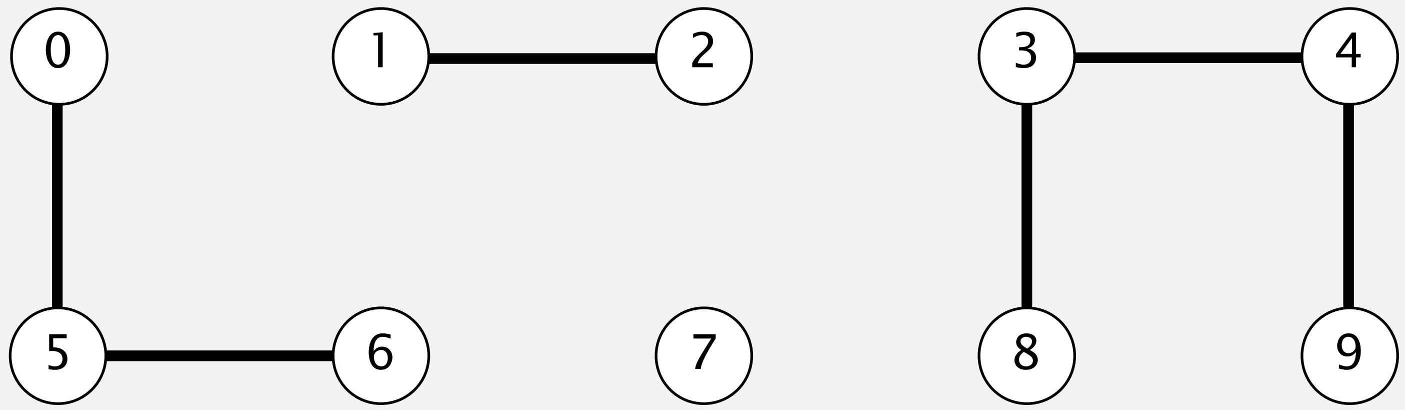
connect 2 and 1

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

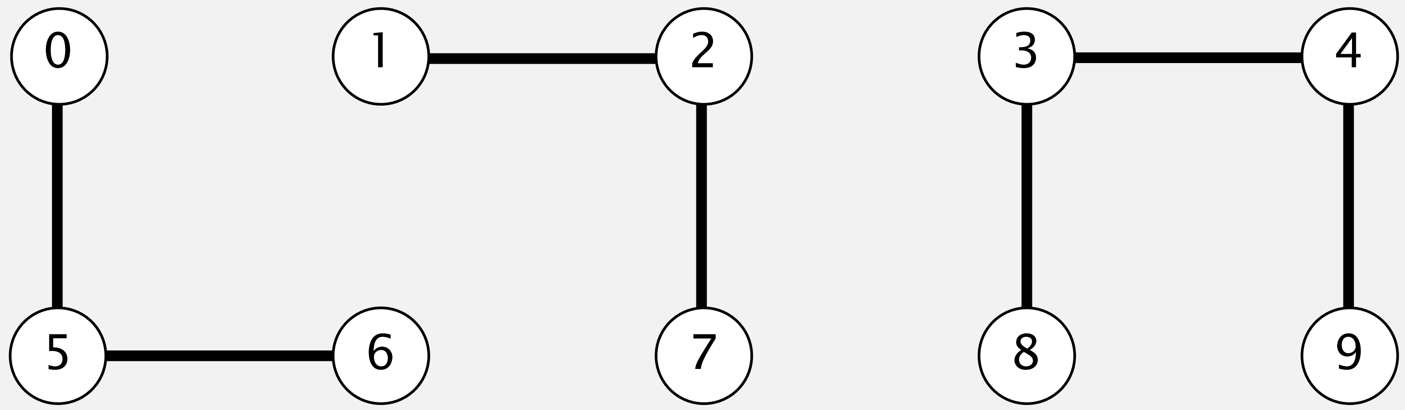
connect 2 and 1

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

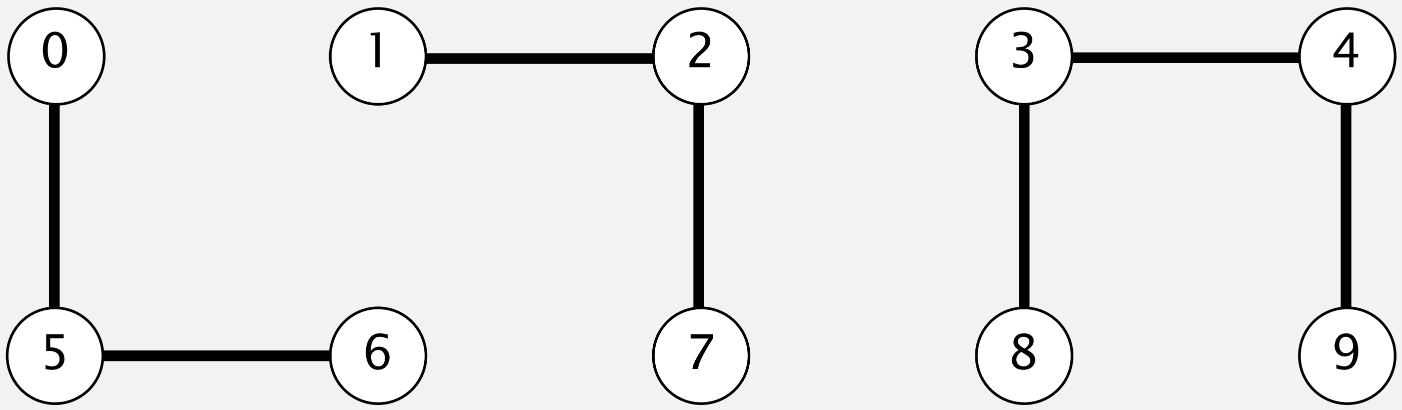
are 0 and 7 connected? ✗

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

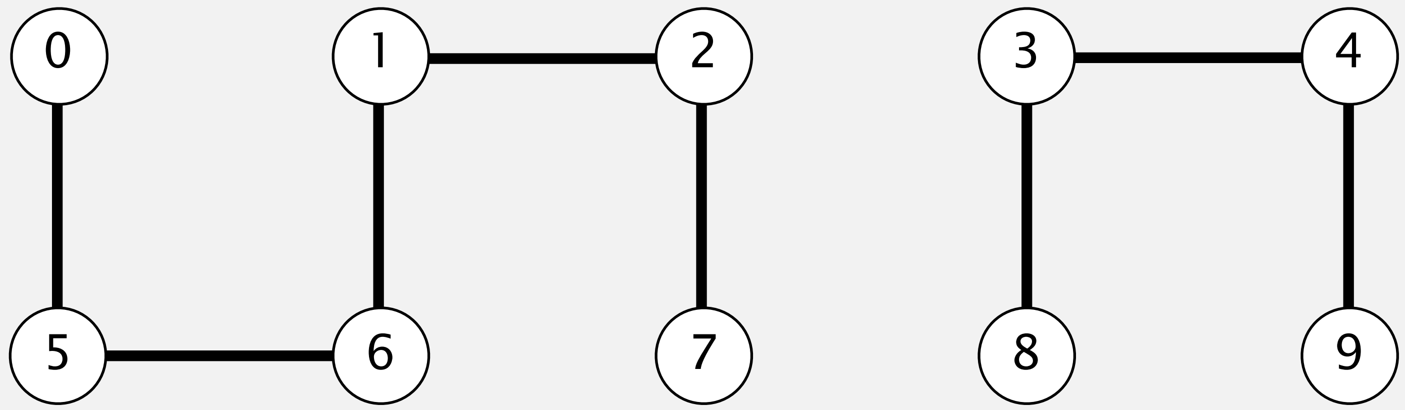
are 0 and 7 connected? ✗

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? ✗

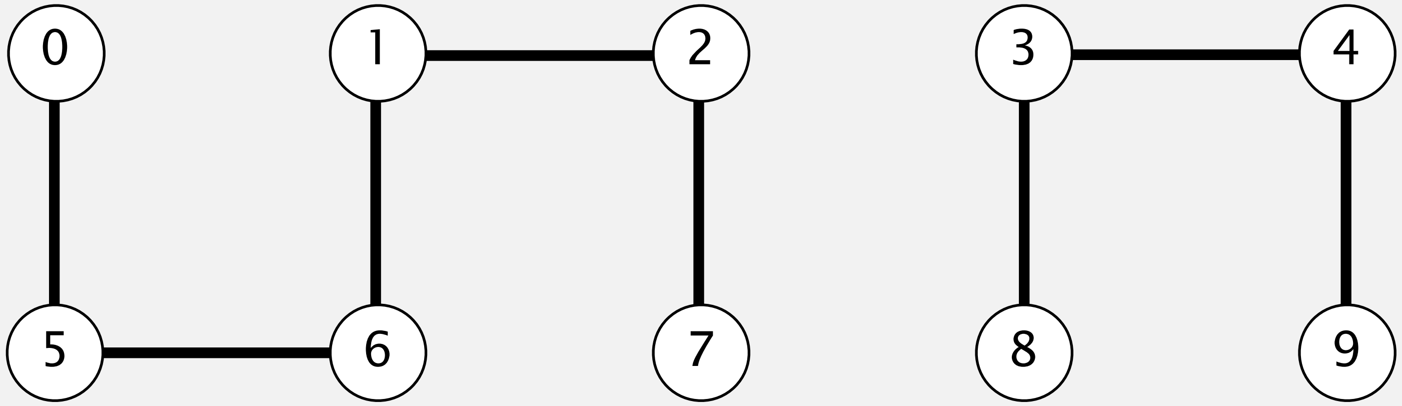
are 8 and 9 connected? ✔

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? ✗

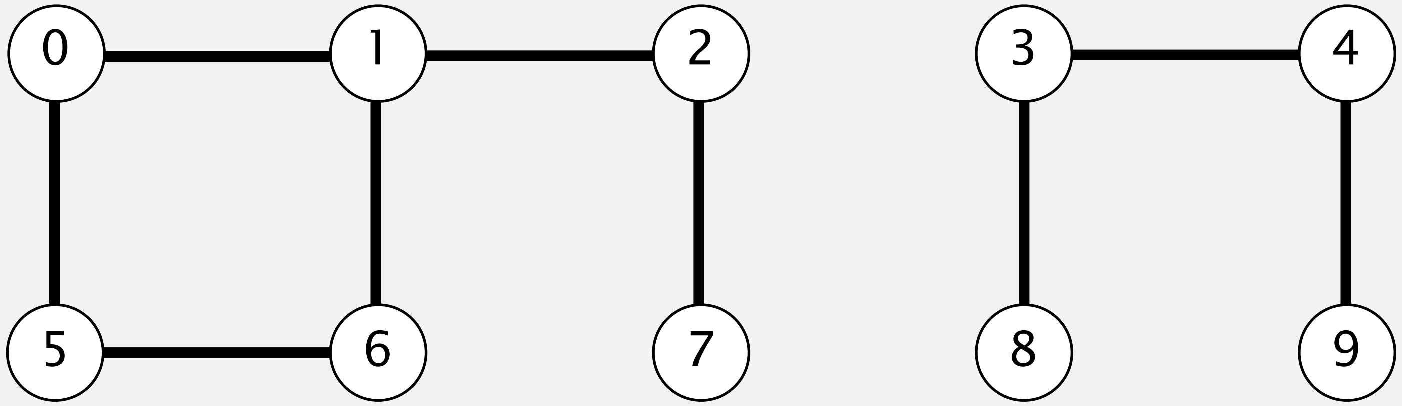
are 8 and 9 connected? ✔

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? ✔

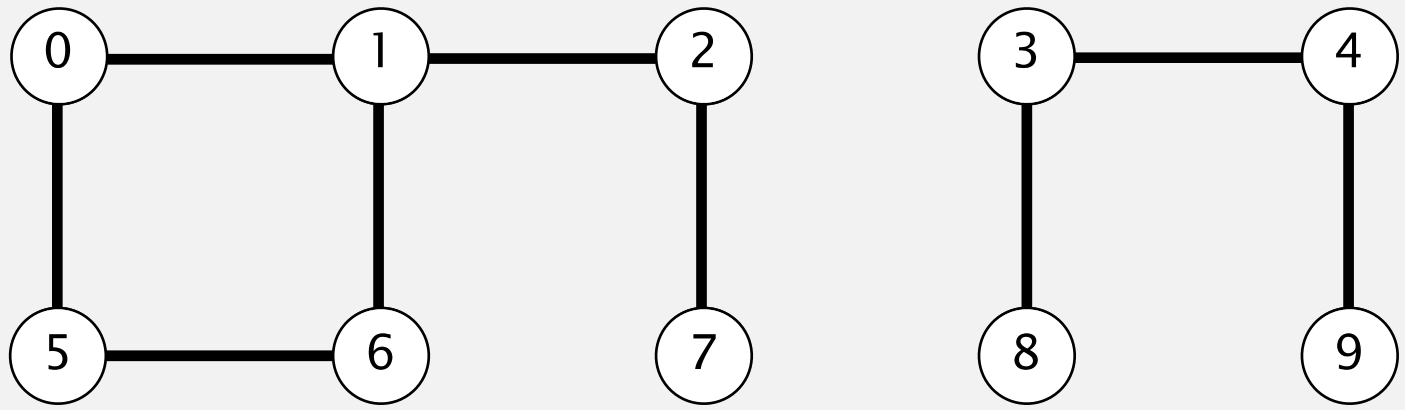
connect 5 and 0

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? ✔

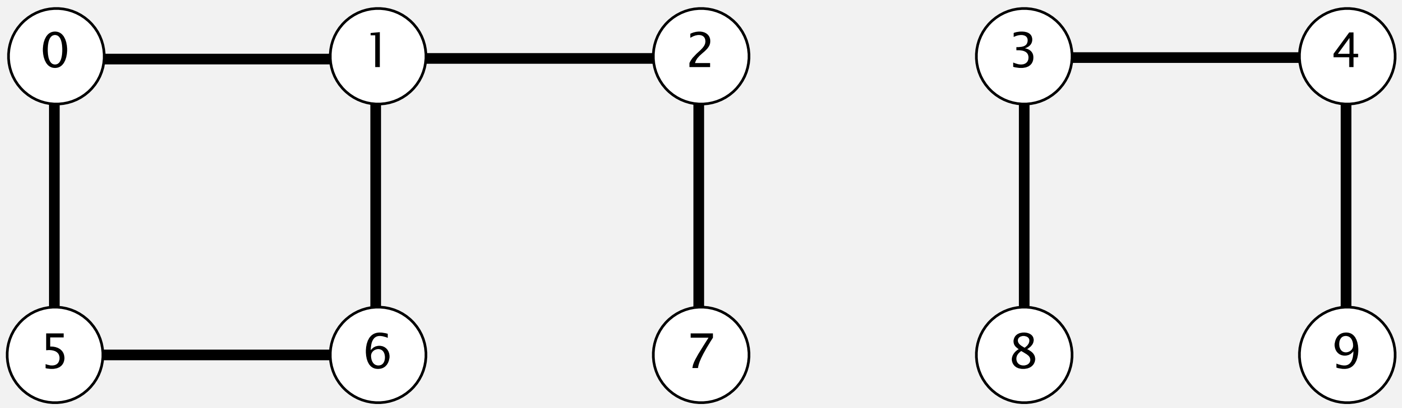
connect 5 and 0

connect 7 and 2

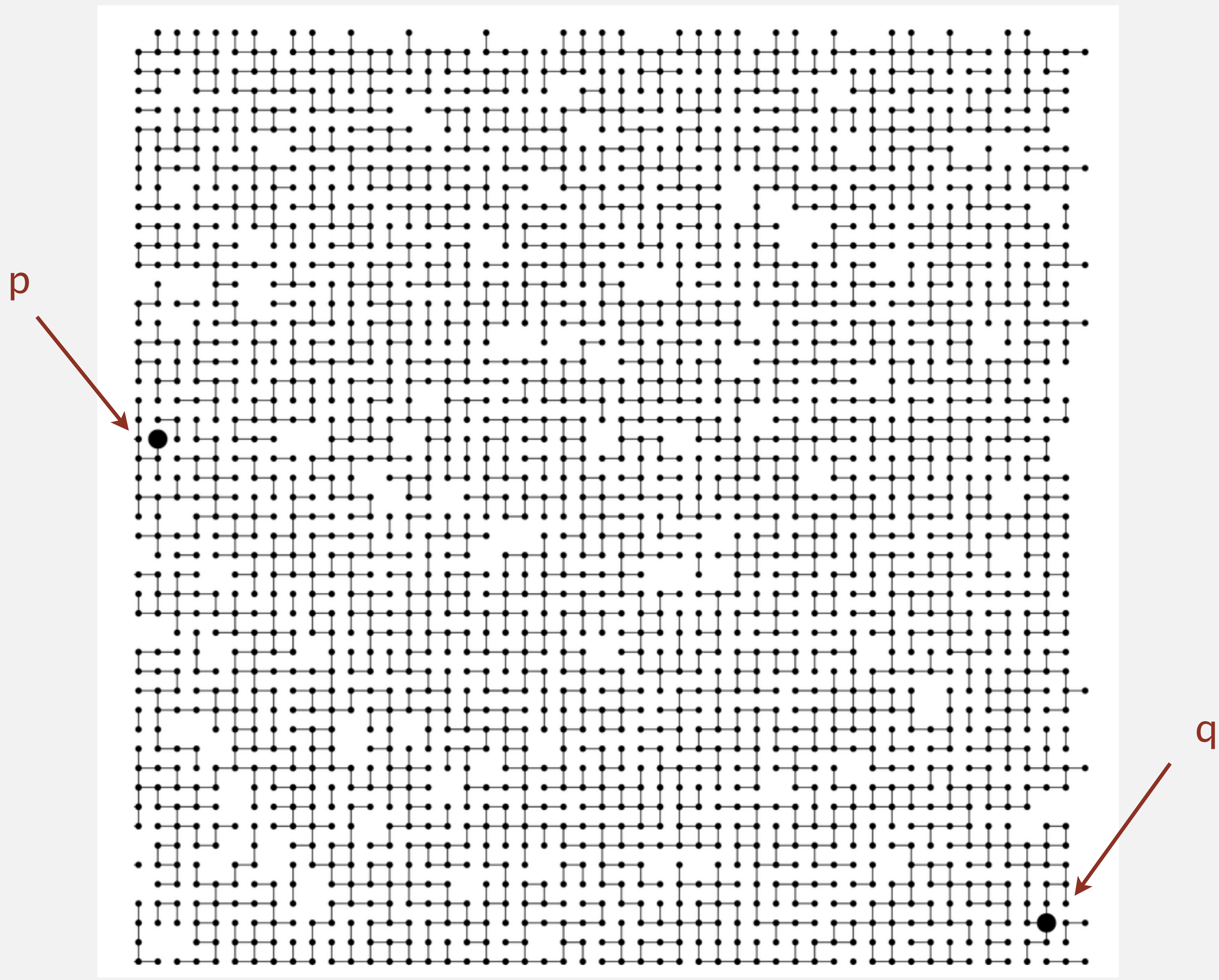
connect 6 and 1

connect 1 and 0

are 0 and 7 connected? ✔

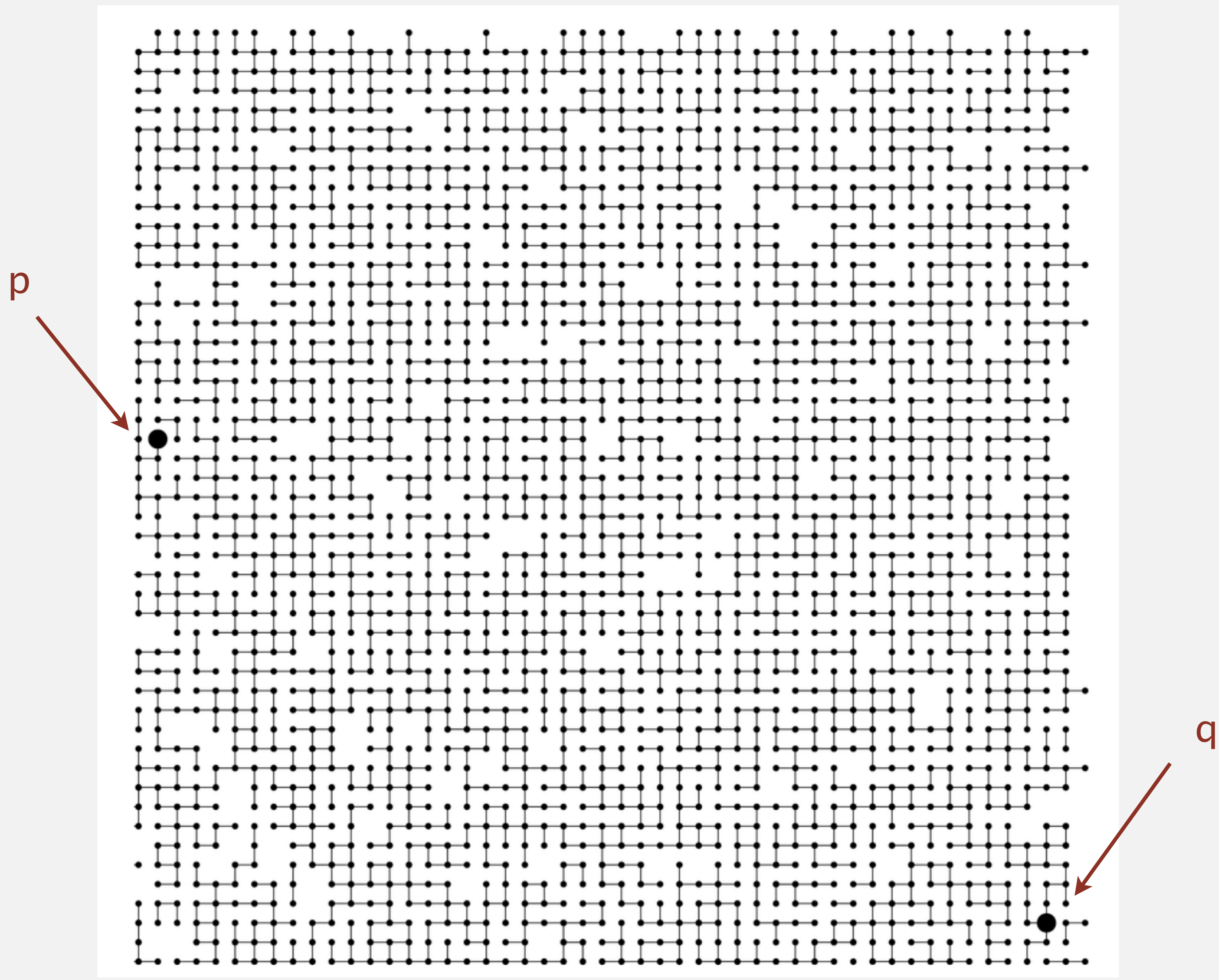


A larger connectivity example



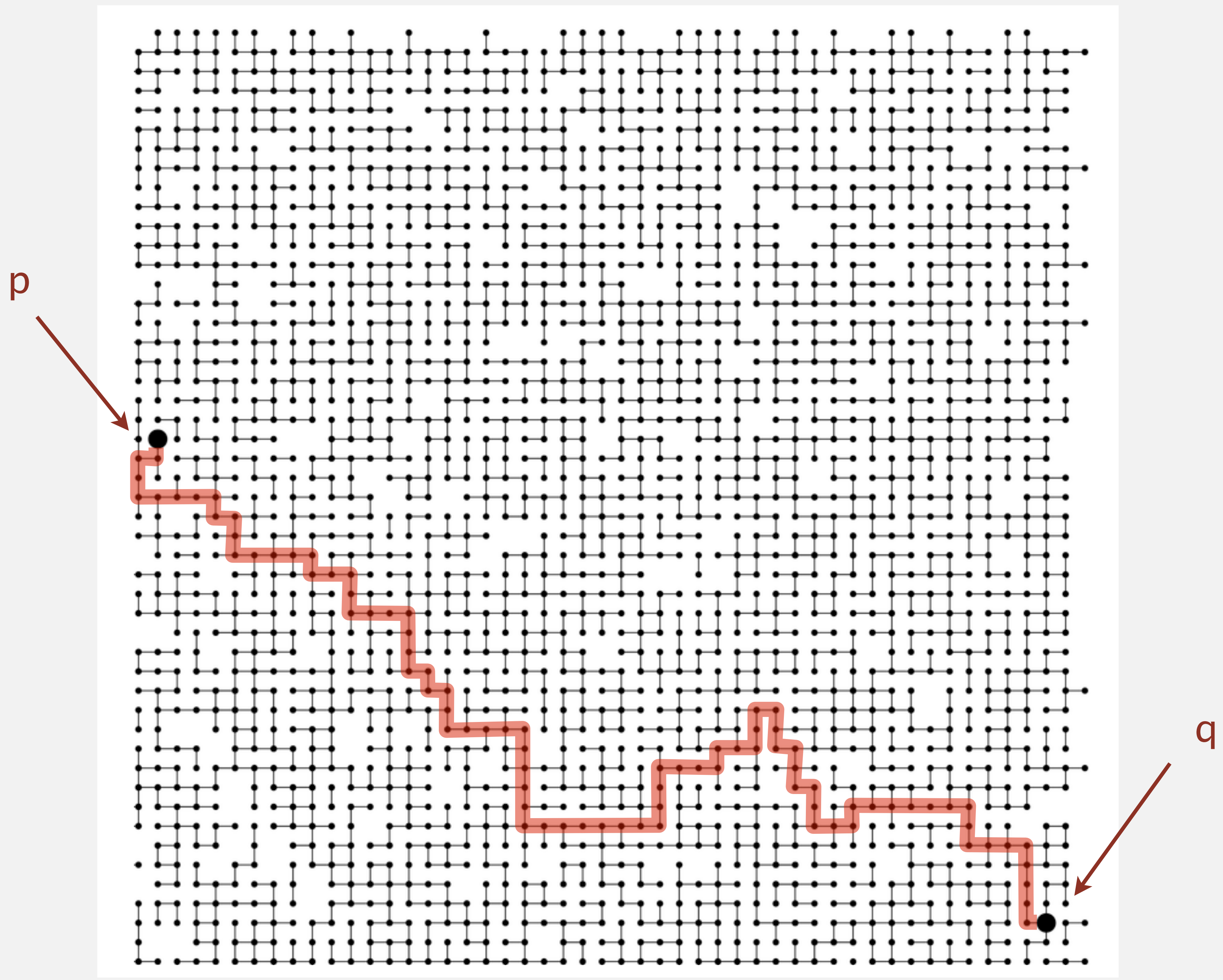
A larger connectivity example

Q. Is there a path connecting p and q ?



A larger connectivity example

Q. Is there a path connecting p and q ?

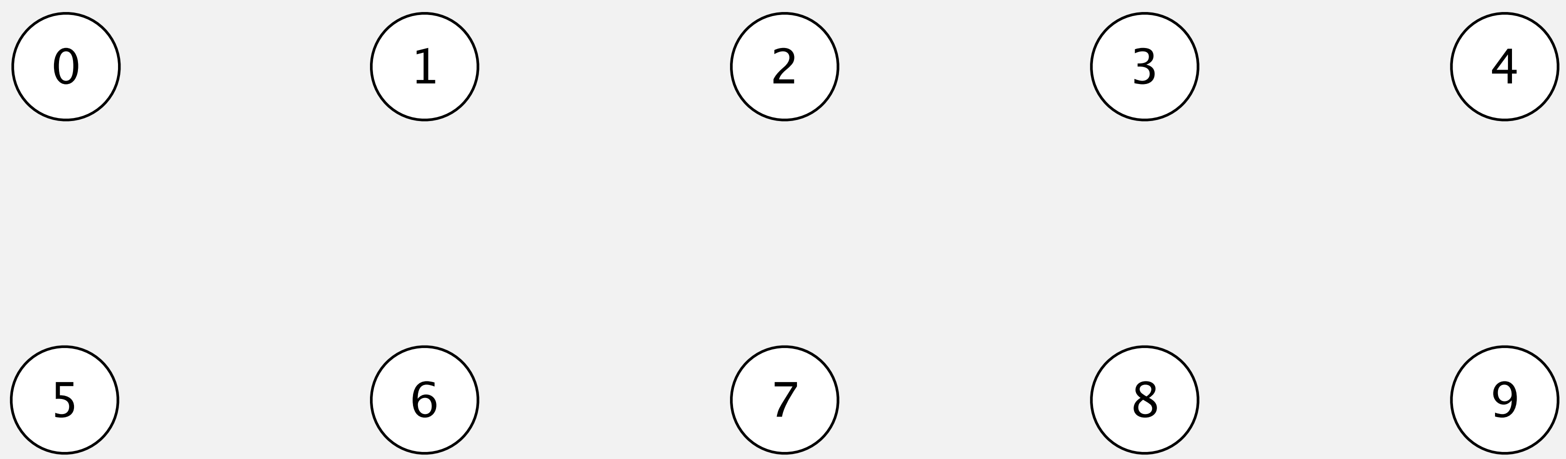


A. Yes.

http://algs4.cs.princeton.edu

1.5 QUICK-FIND DEMO

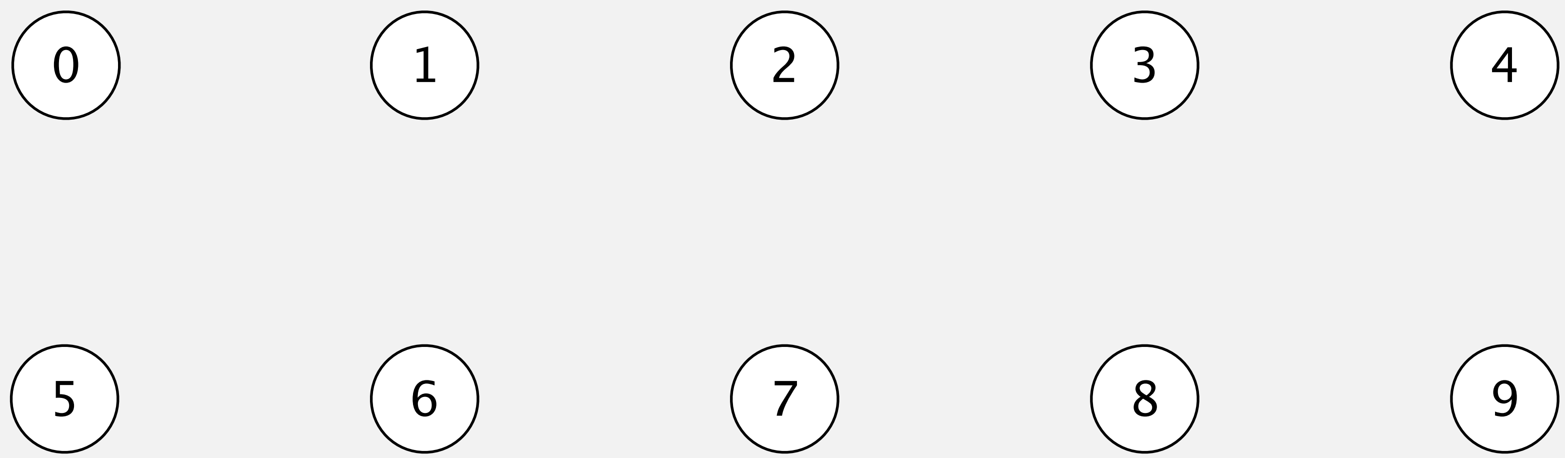
Quick-find demo



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

Quick-find demo

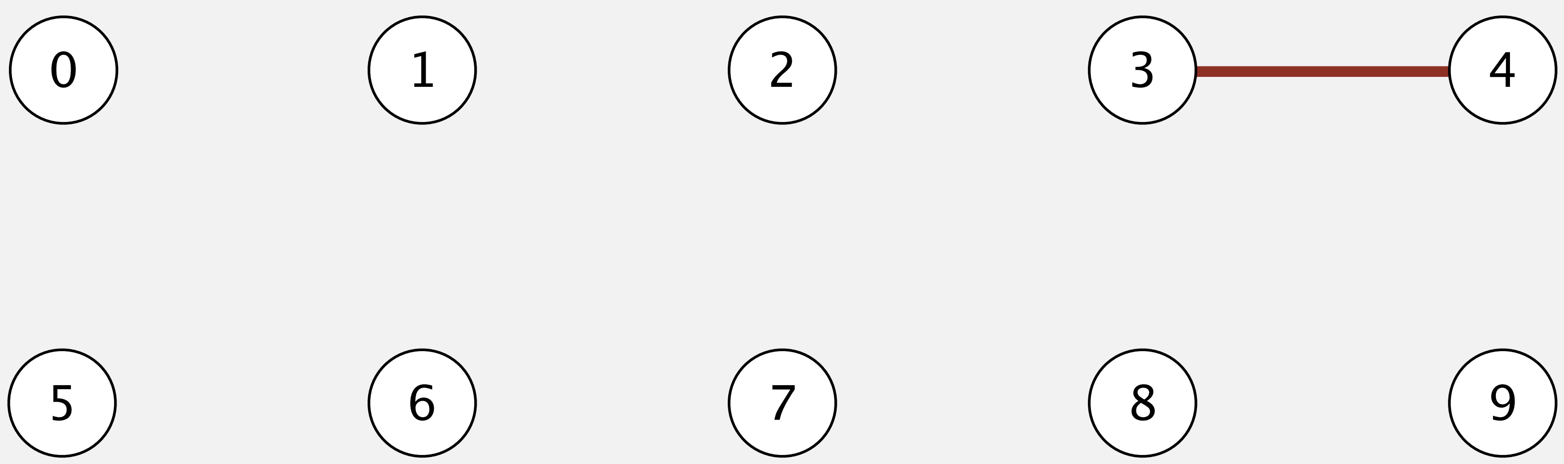
union(4, 3)



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

Quick-find demo

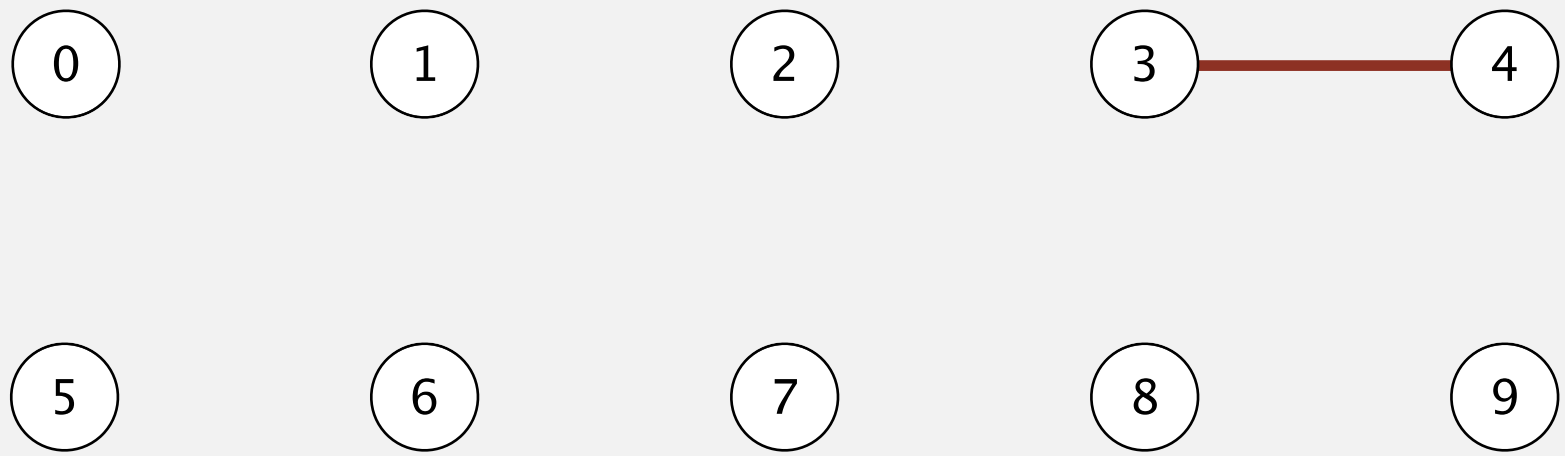
union(4, 3)



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

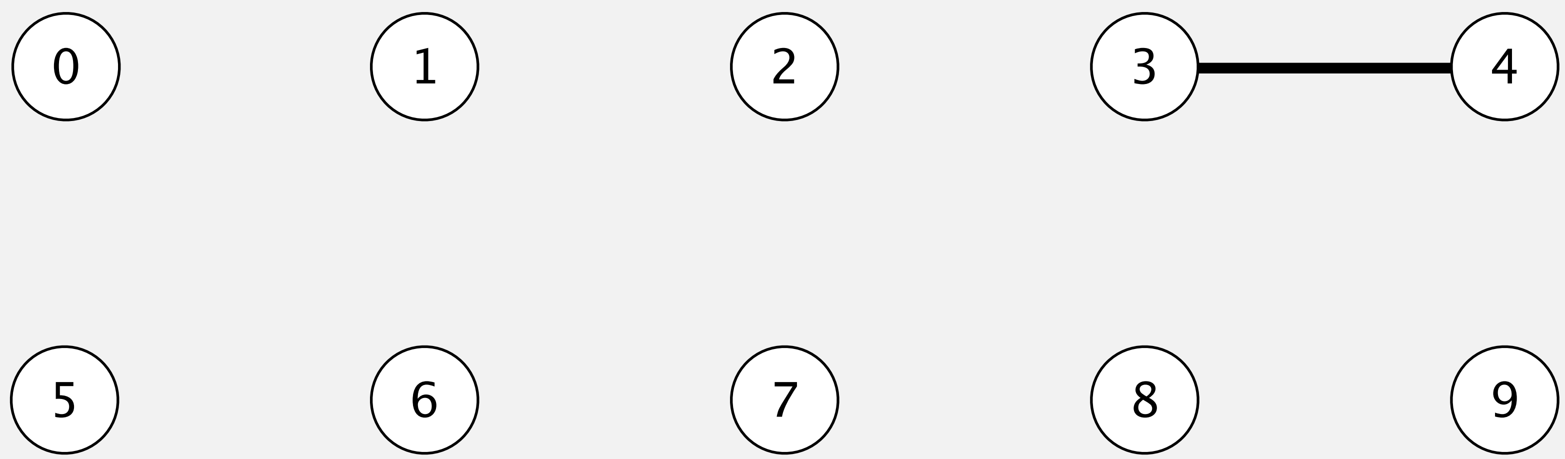
Quick-find demo

union(4, 3)



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

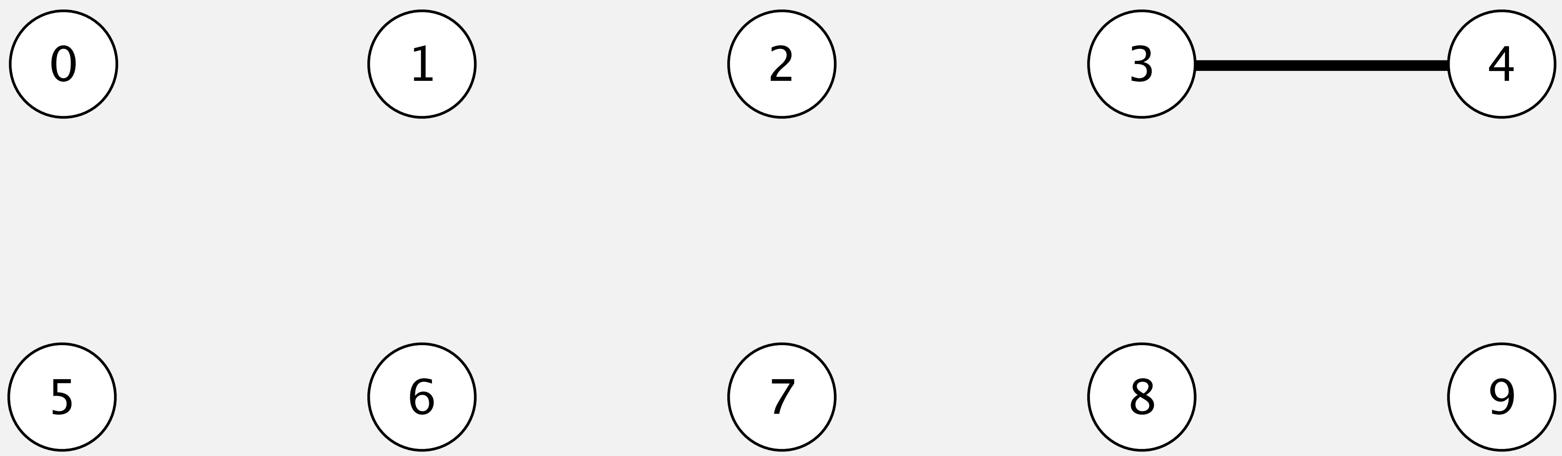
Quick-find demo



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

Quick-find demo

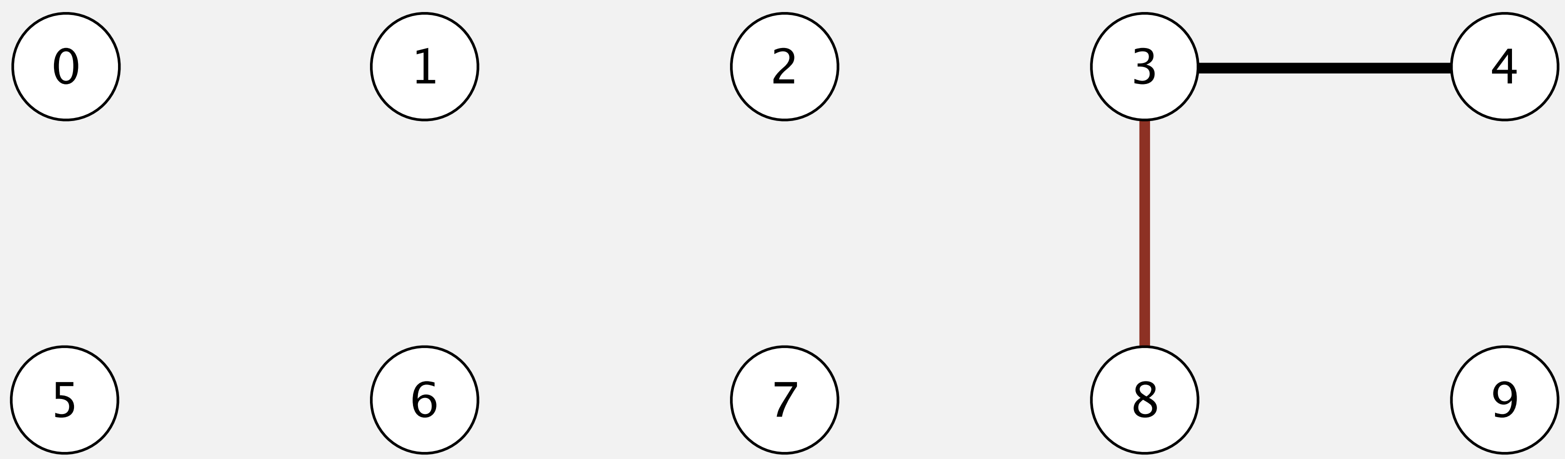
union(3, 8)



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

Quick-find demo

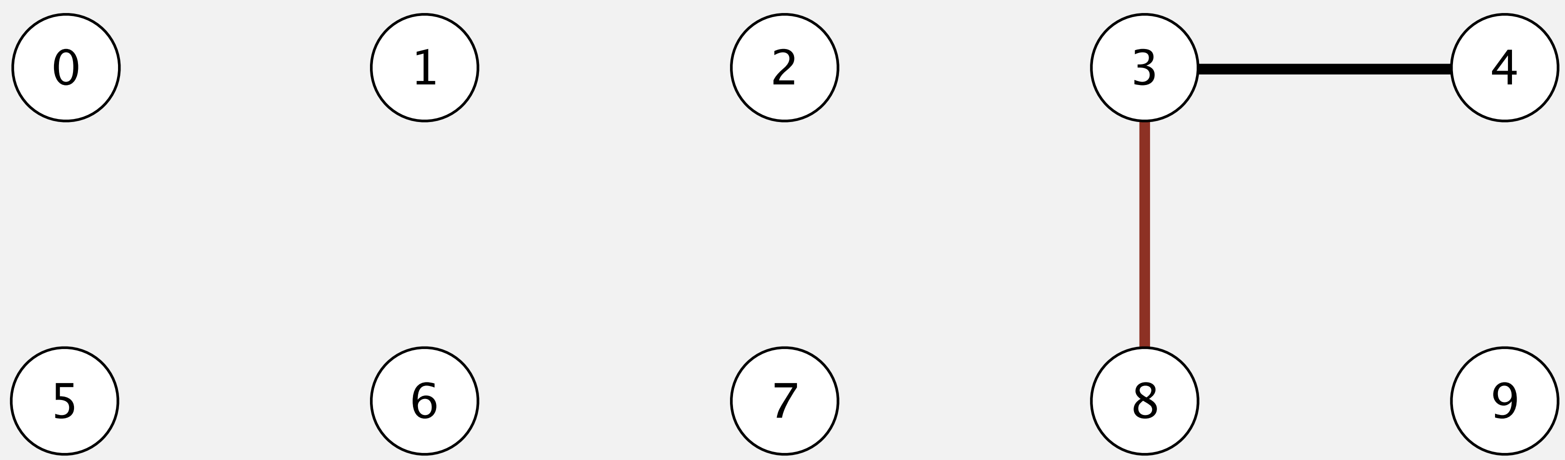
union(3, 8)



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

Quick-find demo

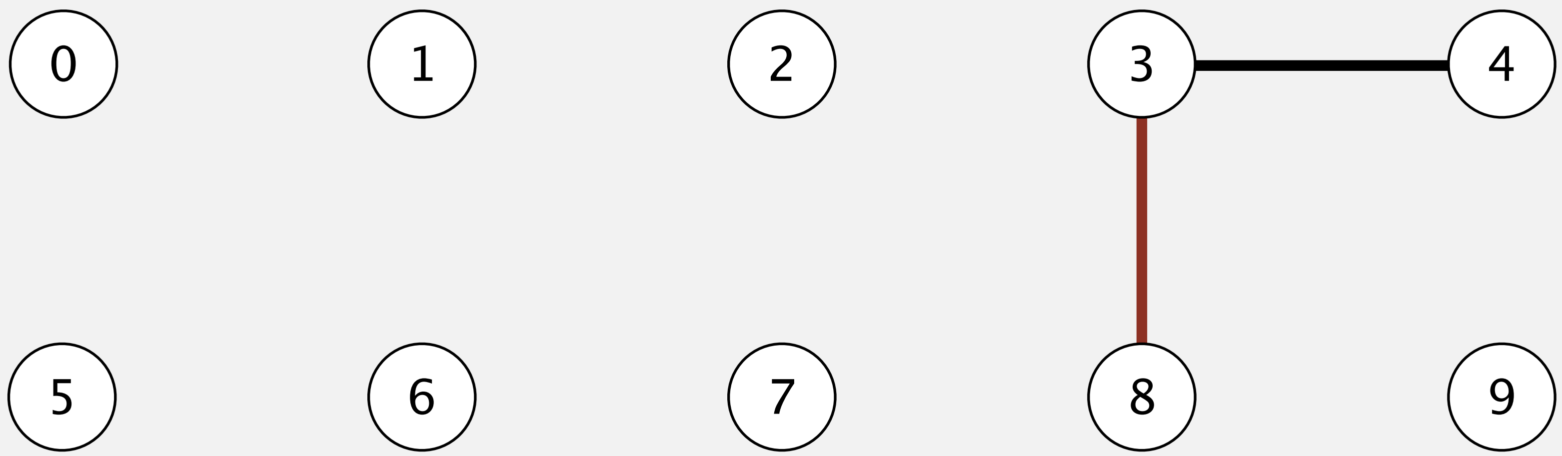
union(3, 8)



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

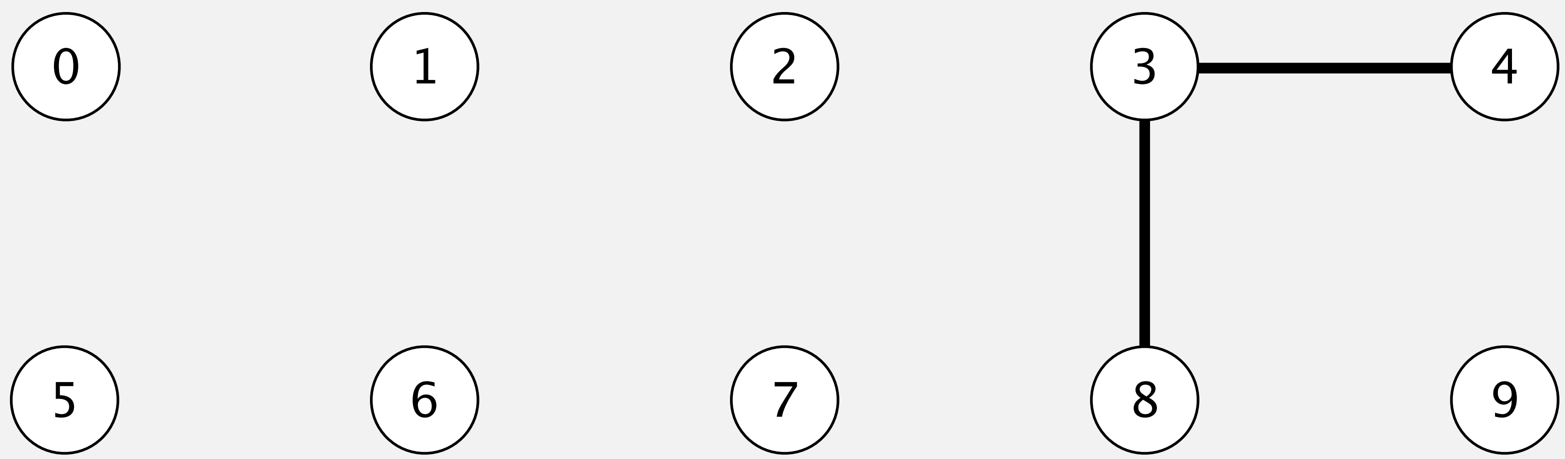
Quick-find demo

union(3, 8)



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

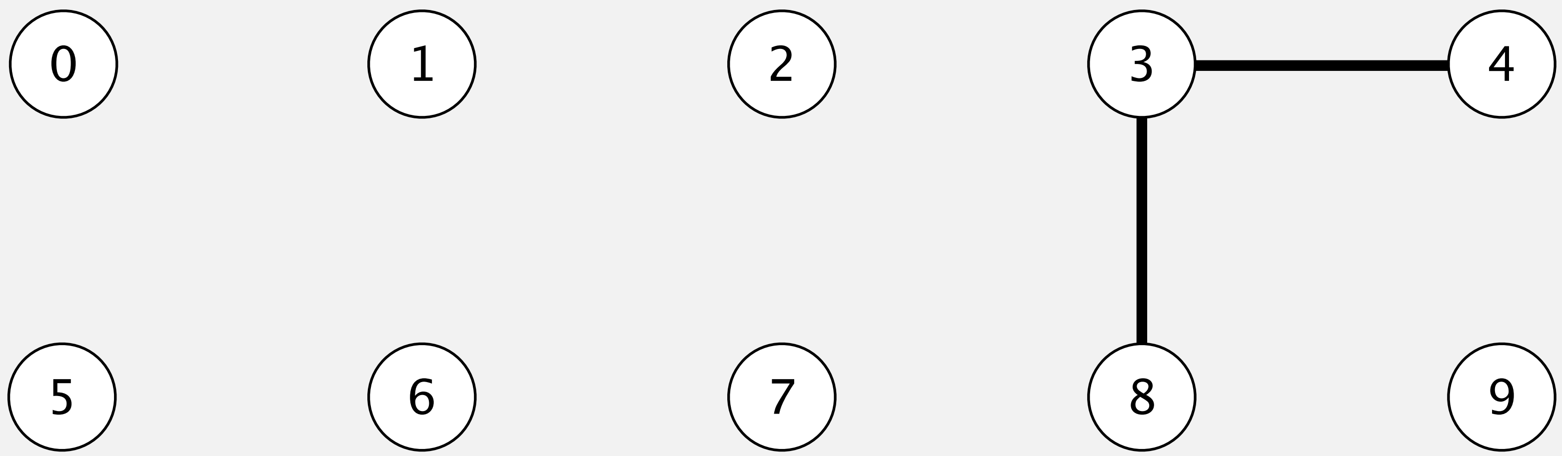
Quick-find demo



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

Quick-find demo

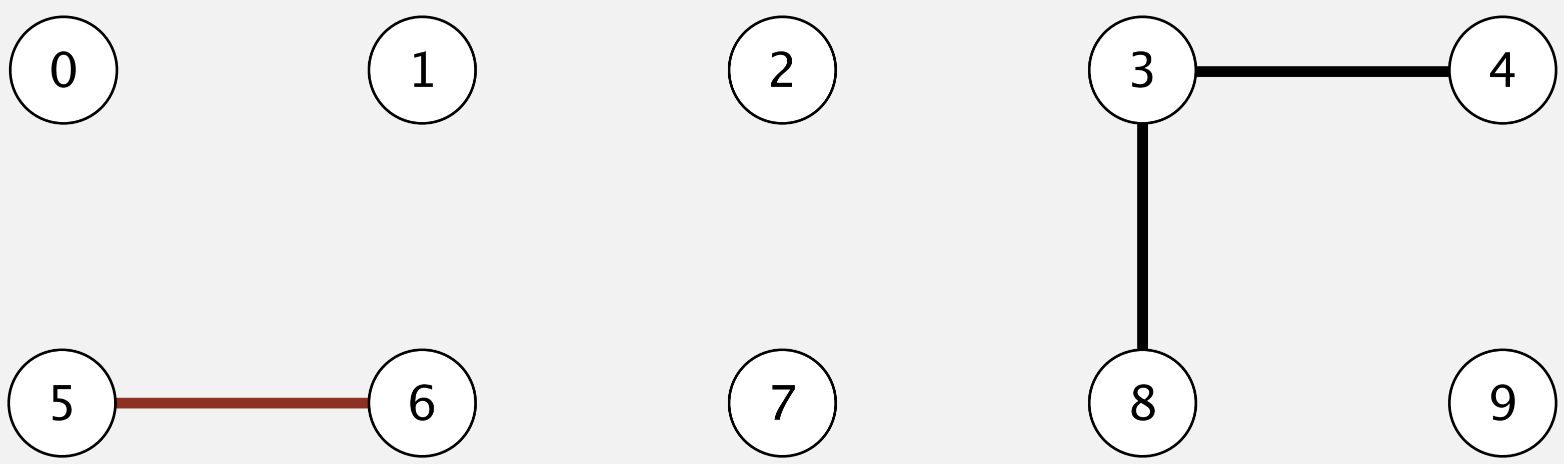
union(6, 5)



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

Quick-find demo

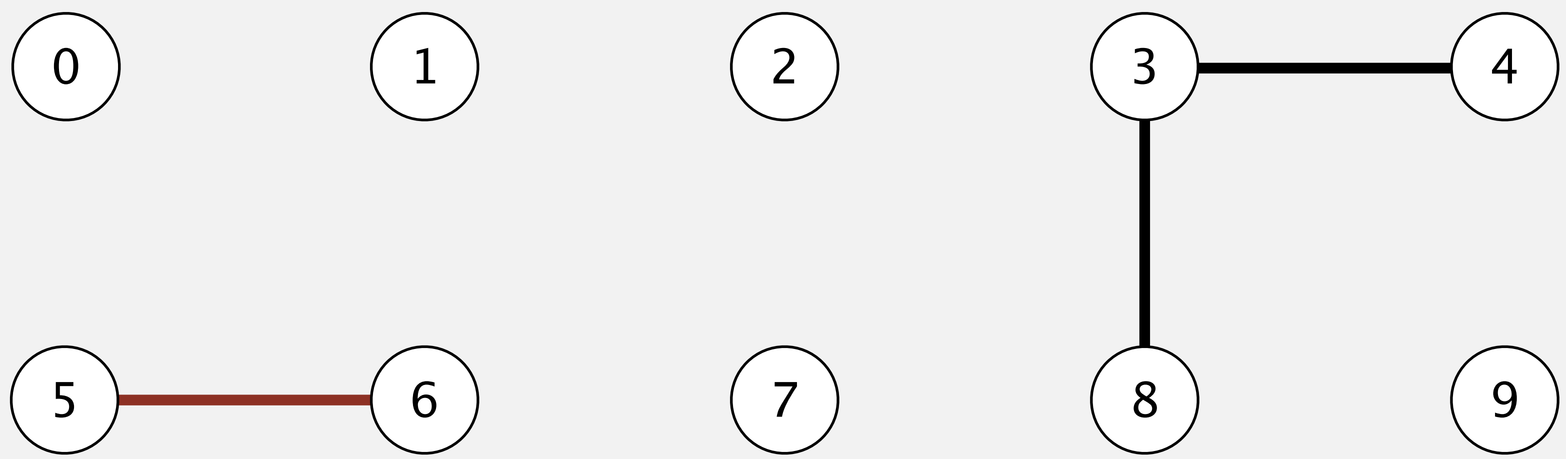
union(6, 5)



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

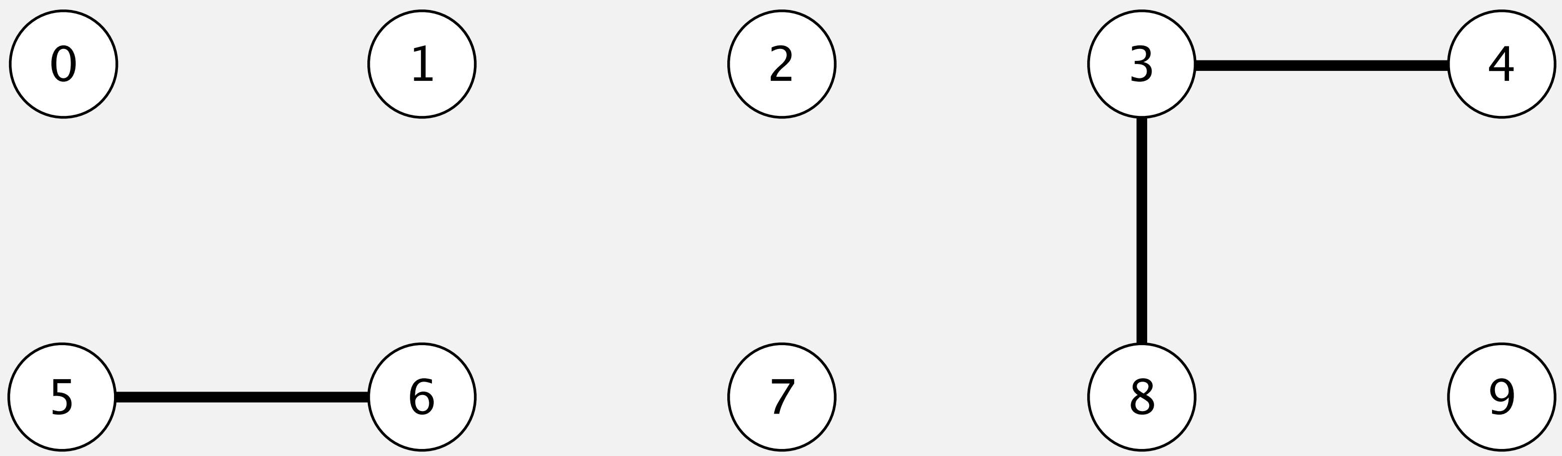
Quick-find demo

union(6, 5)



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

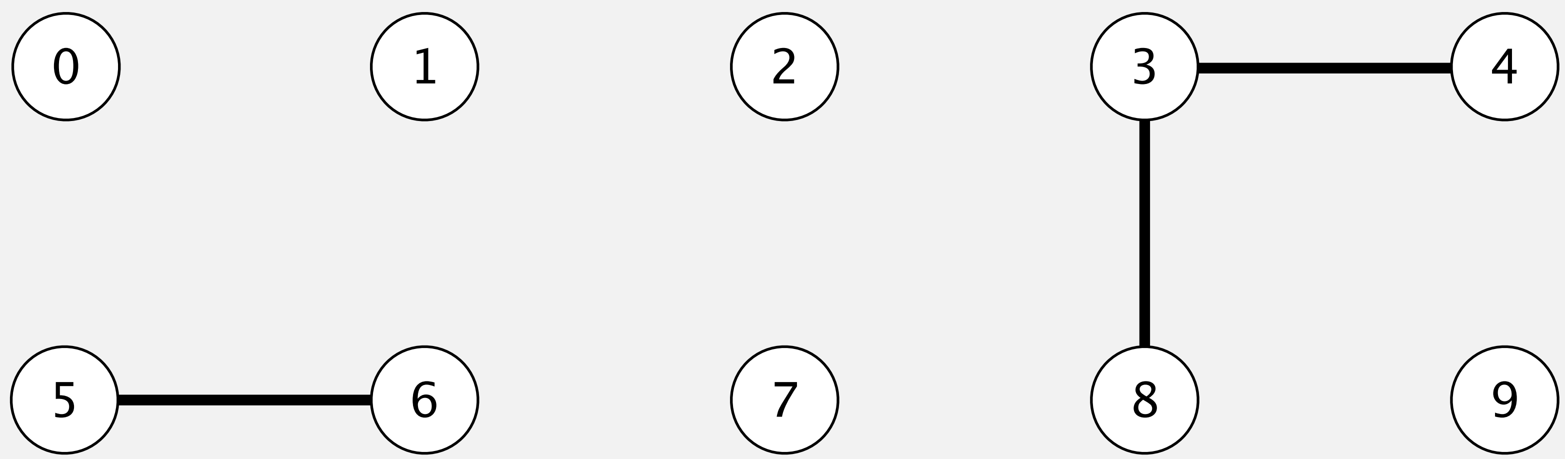
Quick-find demo



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

Quick-find demo

union(9, 4)

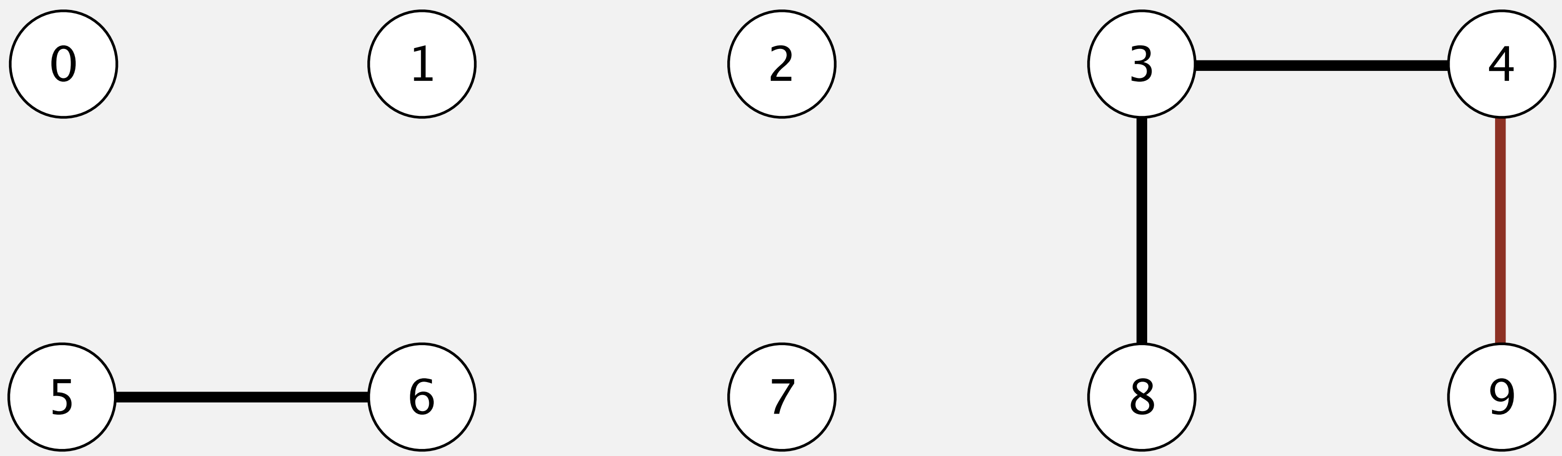


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	8	8	5	5	7	8	9

↑ (4) ↑ (9)

Quick-find demo

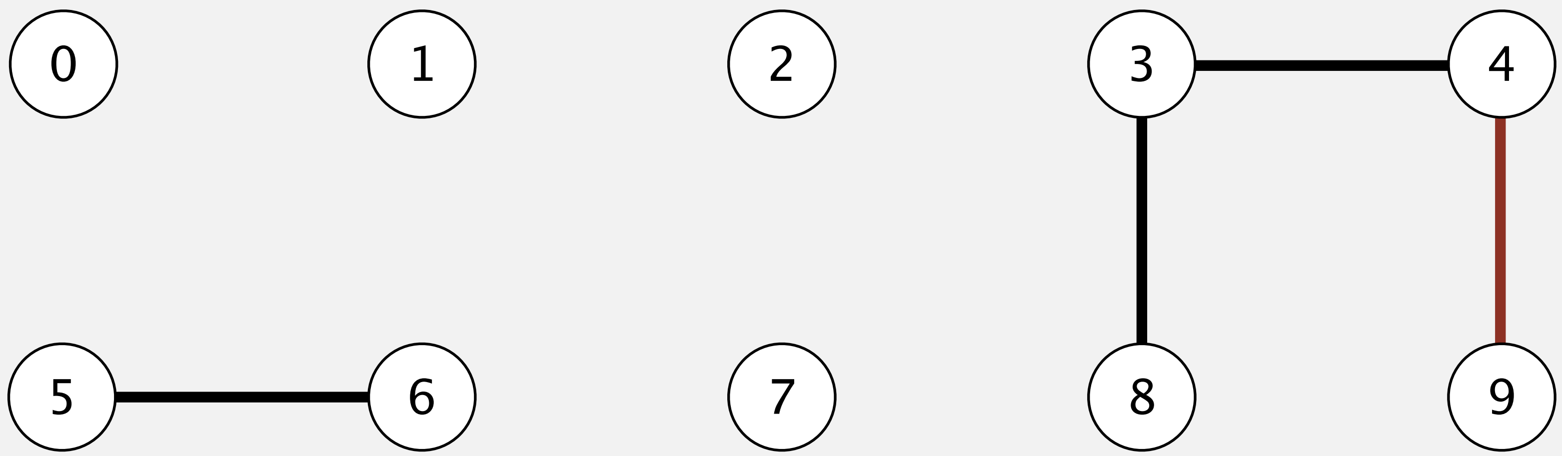
union(9, 4)



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

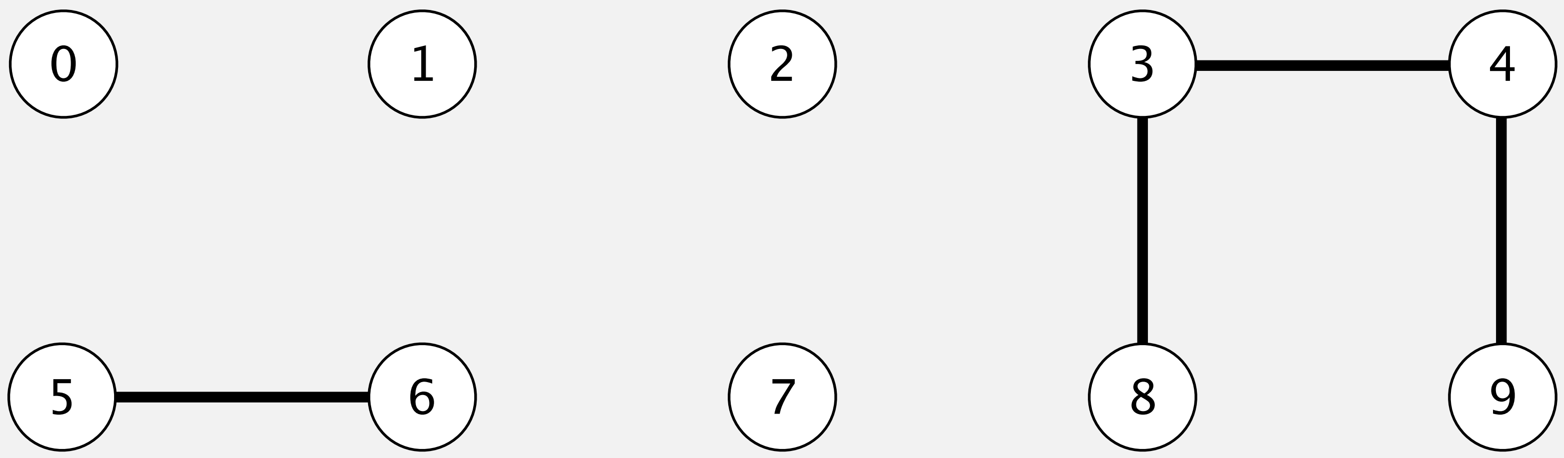
Quick-find demo

union(9, 4)



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

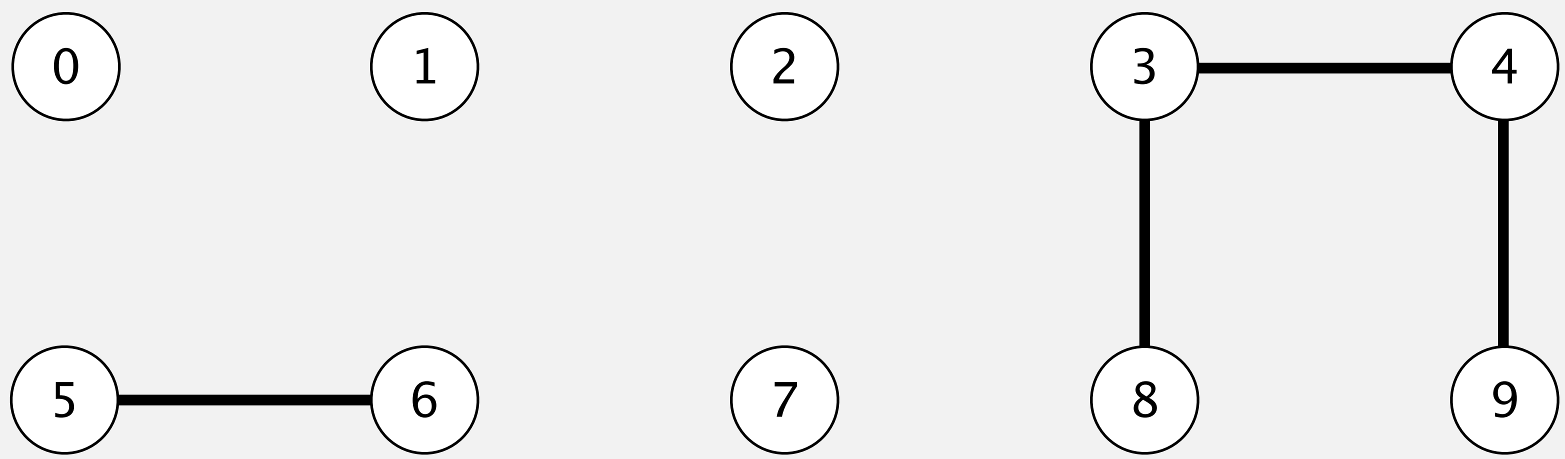
Quick-find demo



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

Quick-find demo

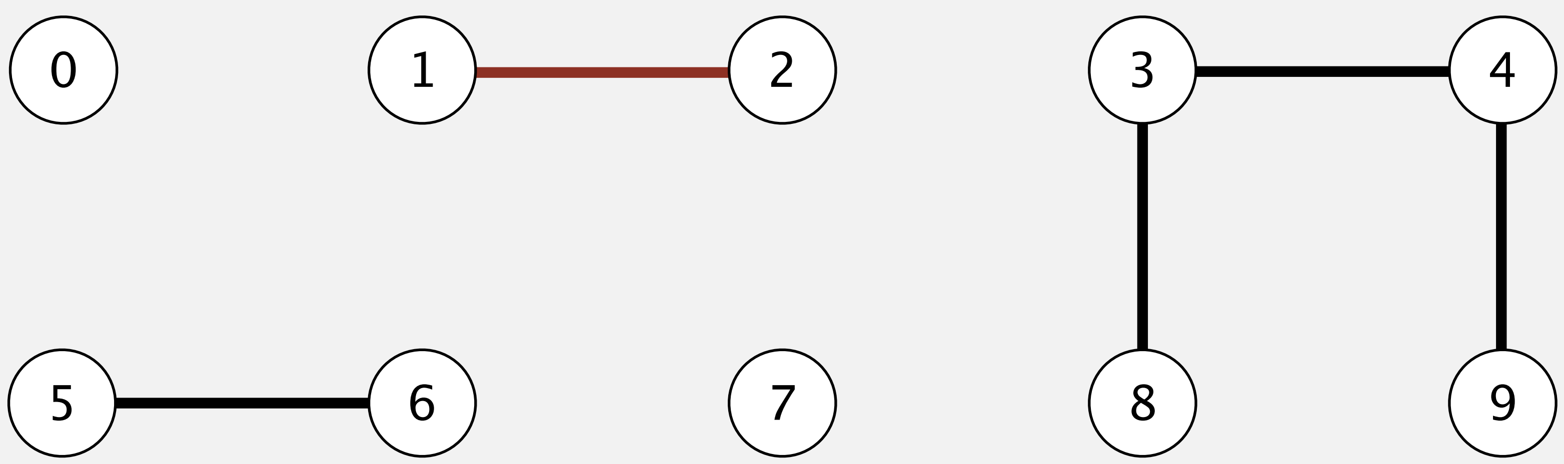
union(2, 1)



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

Quick-find demo

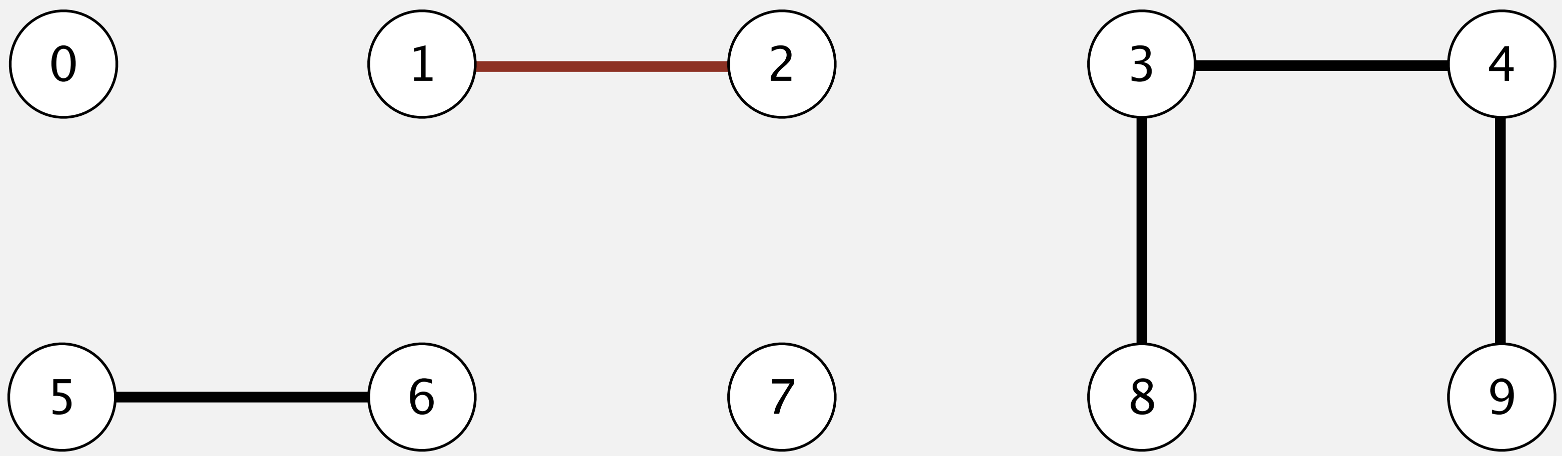
union(2, 1)



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

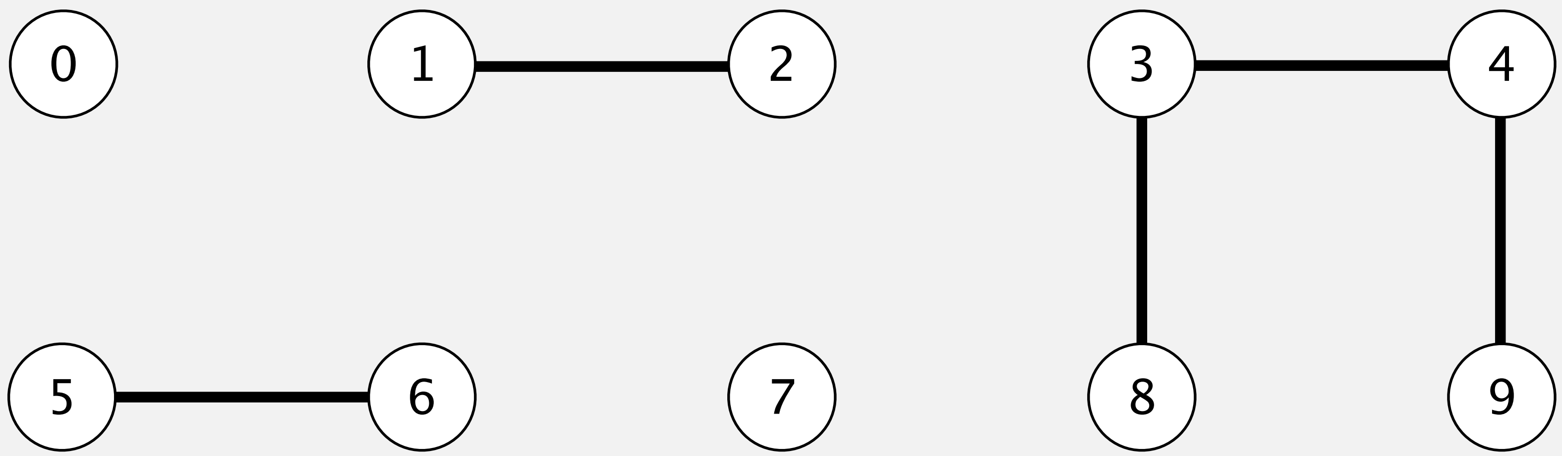
Quick-find demo

union(2, 1)



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

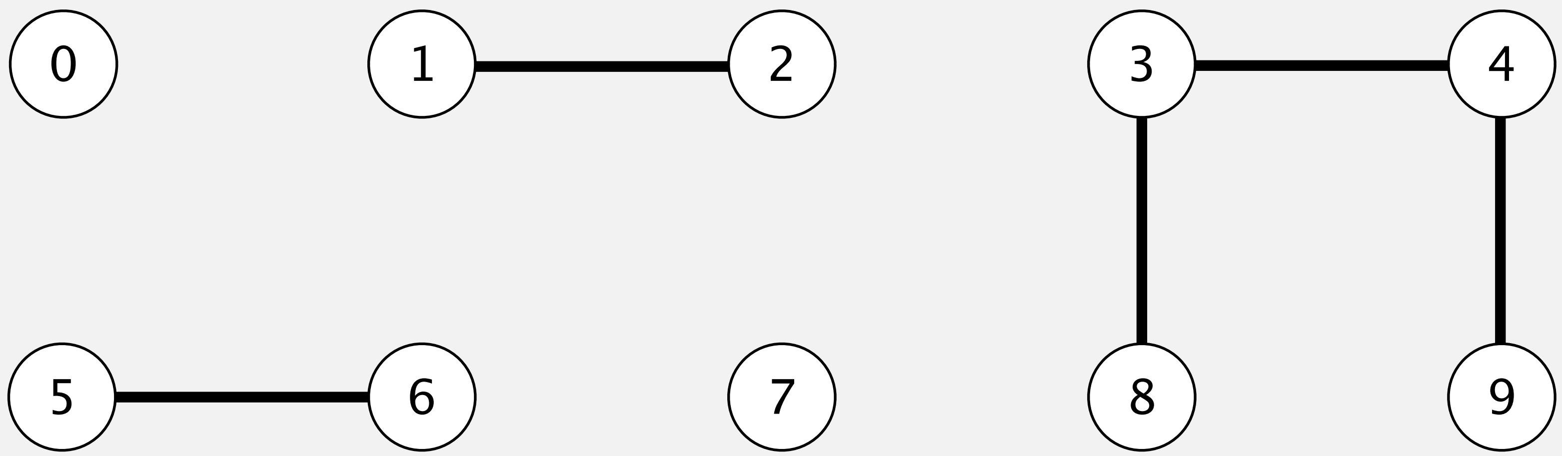
Quick-find demo



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

Quick-find demo

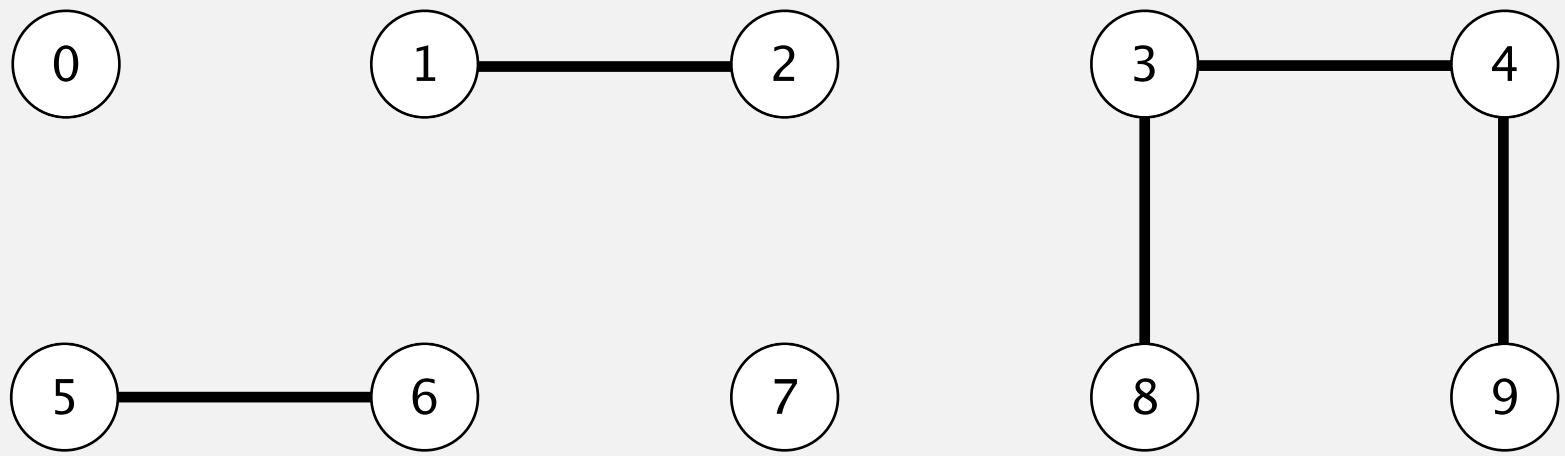
connected(8, 9)



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

Quick-find demo

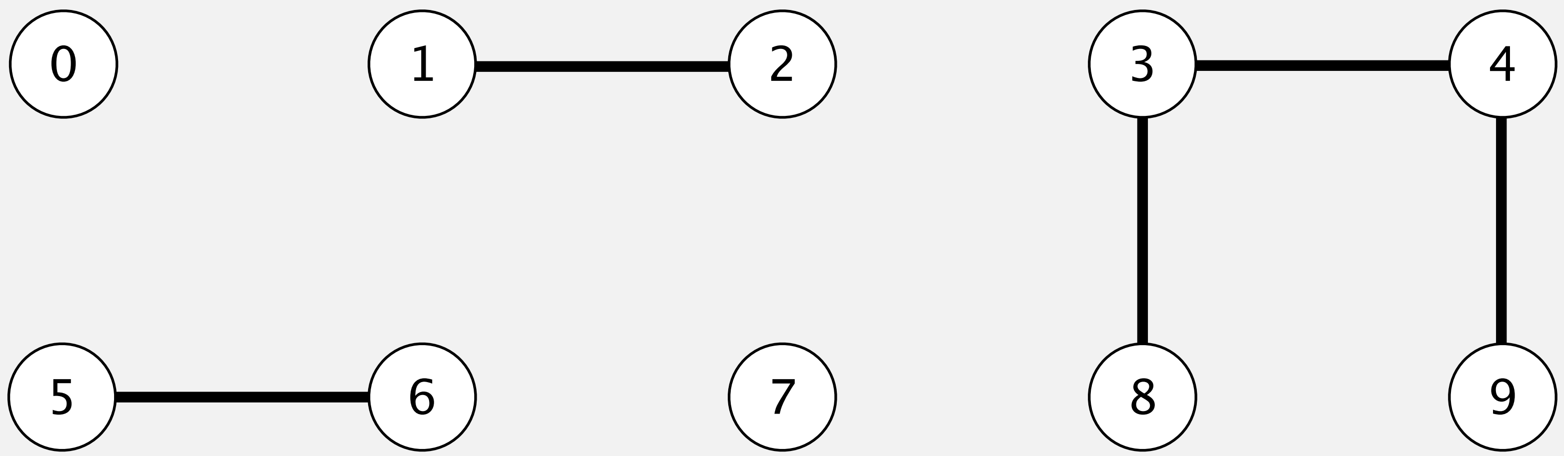
connected(8, 9)



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

↑ ↑ **already connected**

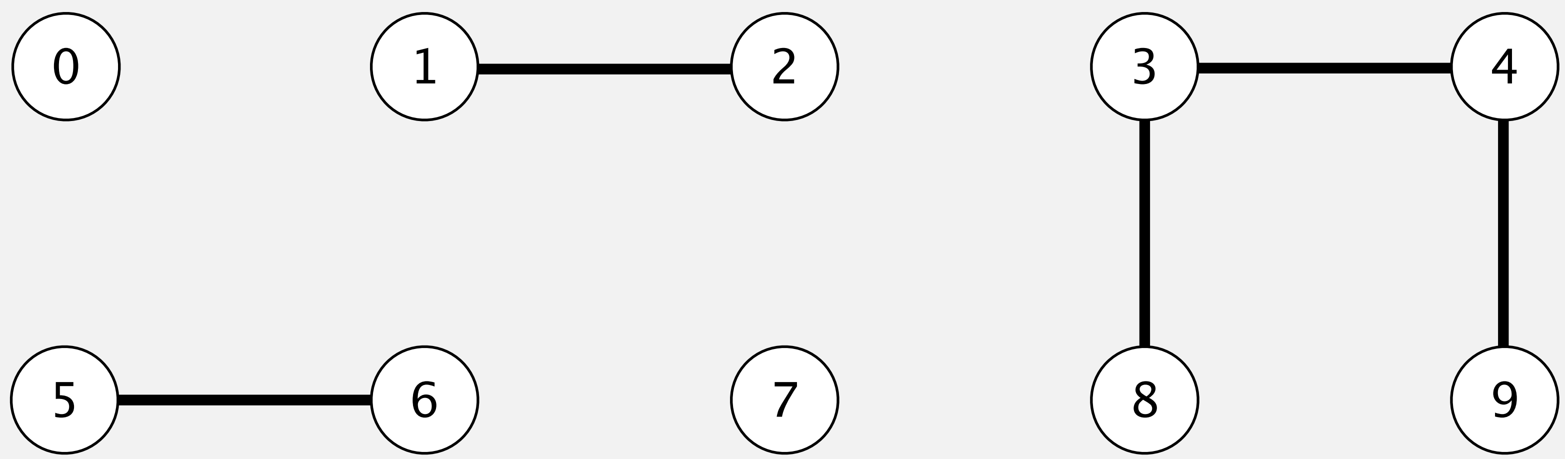
Quick-find demo



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

Quick-find demo

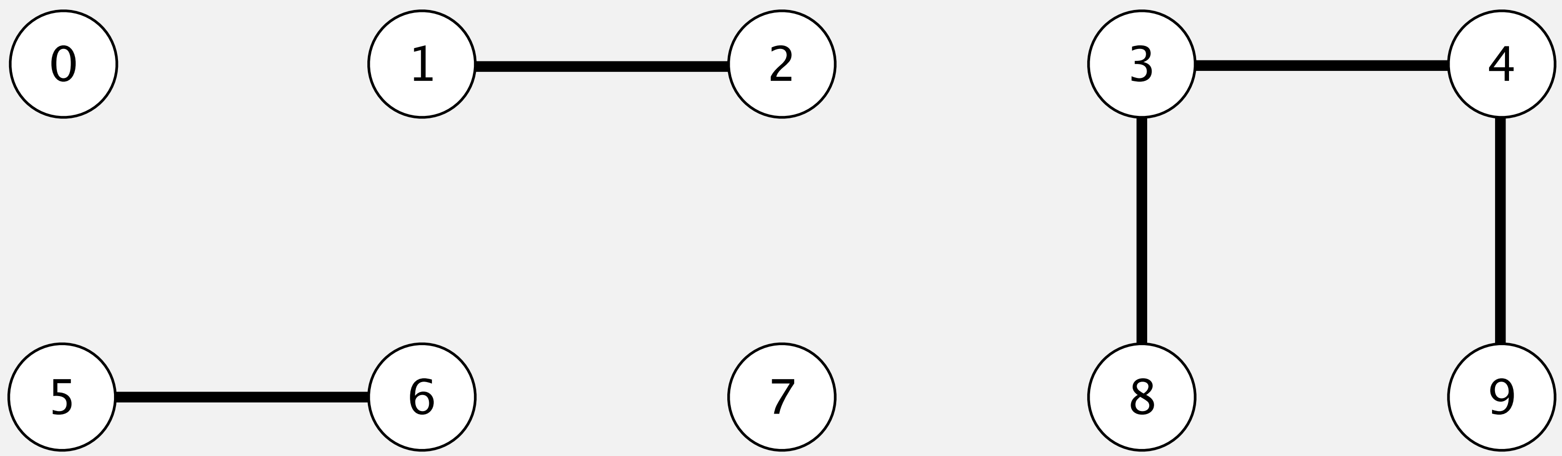
connected(5, 0)



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

Quick-find demo

connected(5, 0)

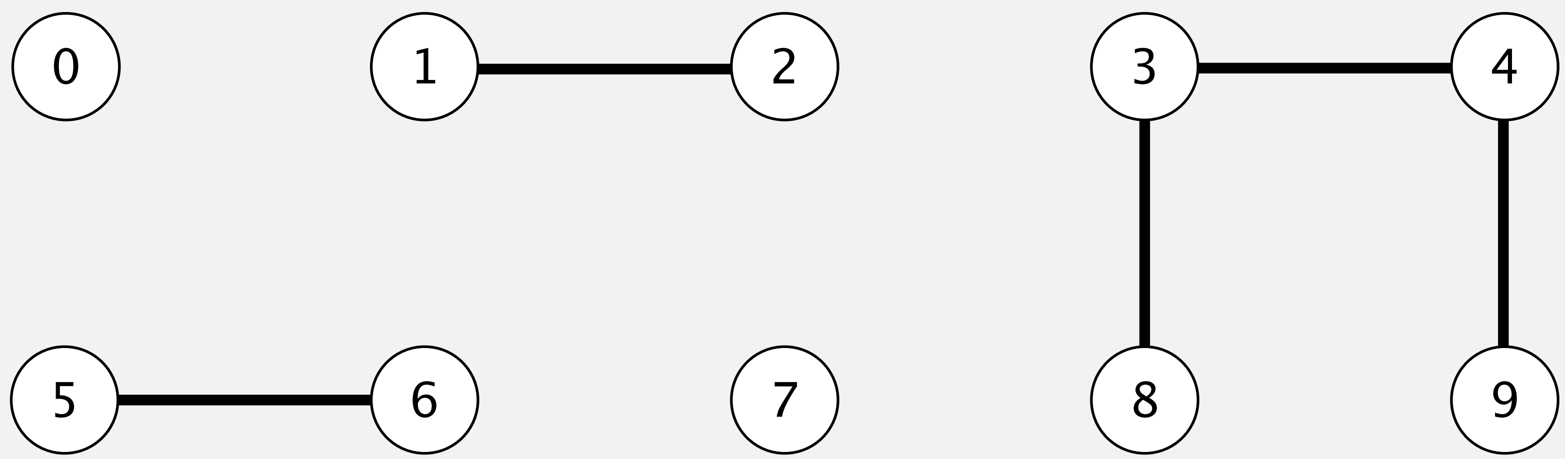


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	5	5	7	8	8

↑ (0) ↑ (5)

not connected

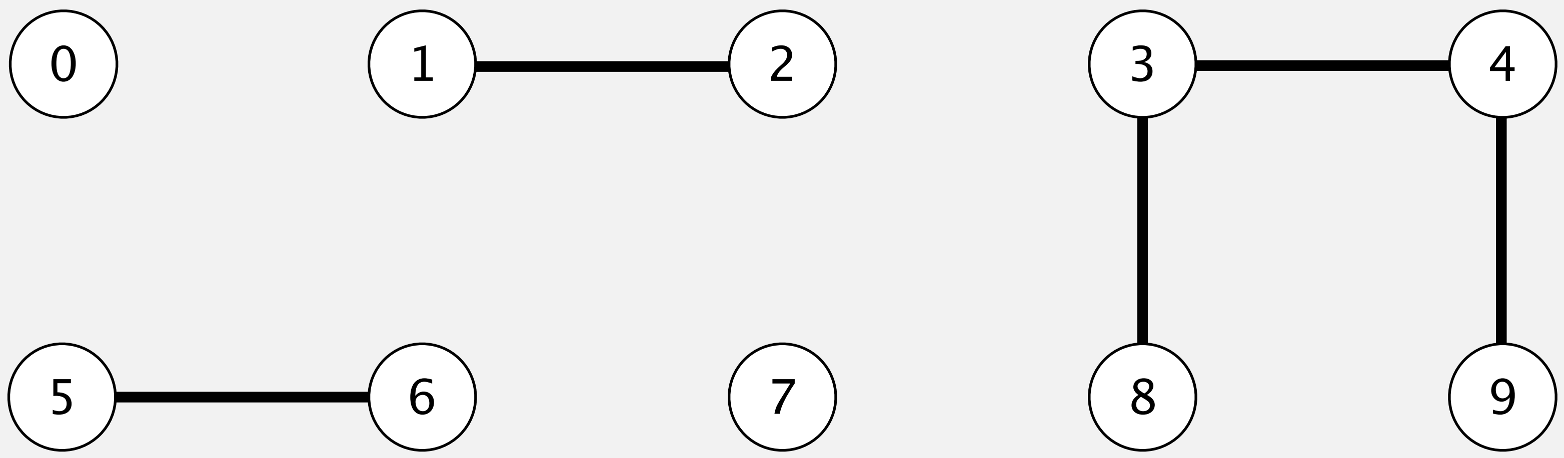
Quick-find demo



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

Quick-find demo

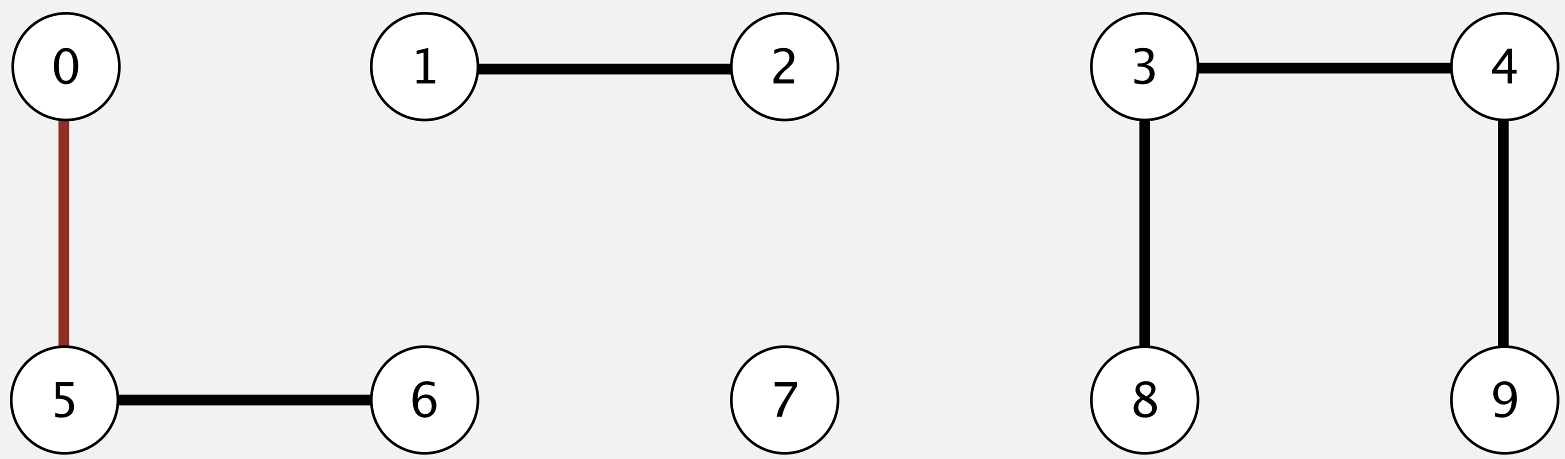
union(5, 0)



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

Quick-find demo

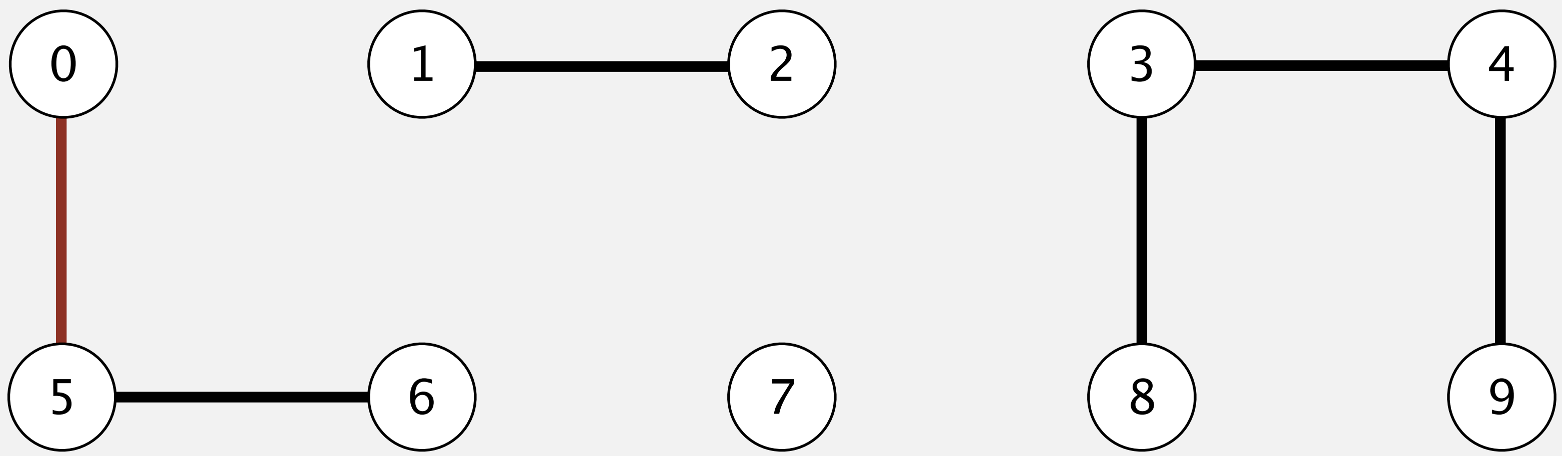
union(5, 0)



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

Quick-find demo

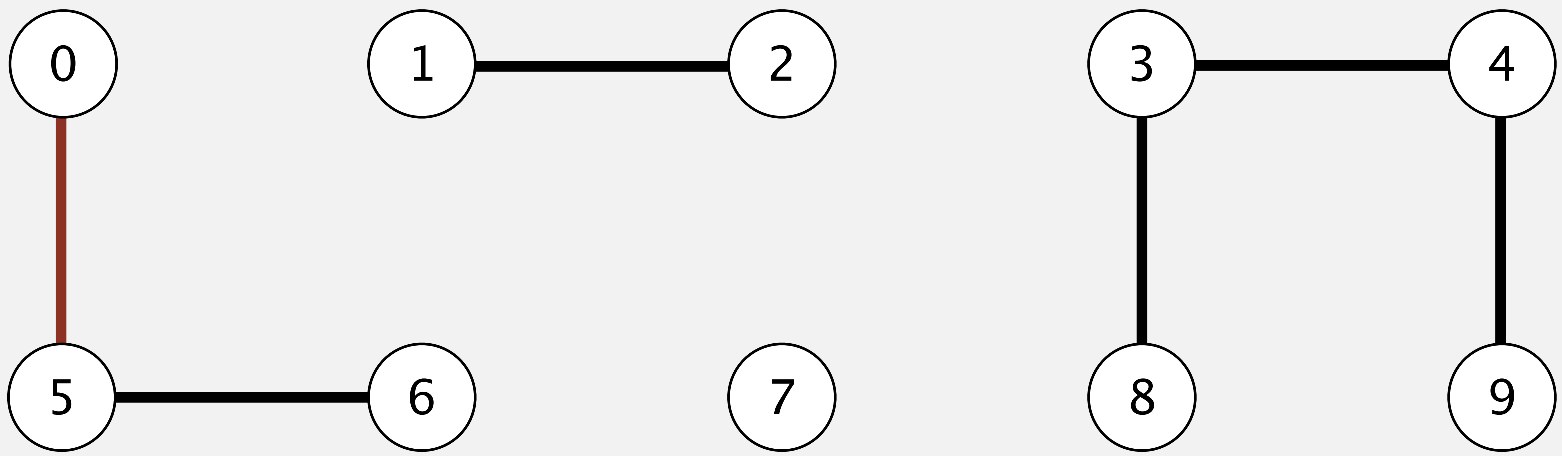
union(5, 0)



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

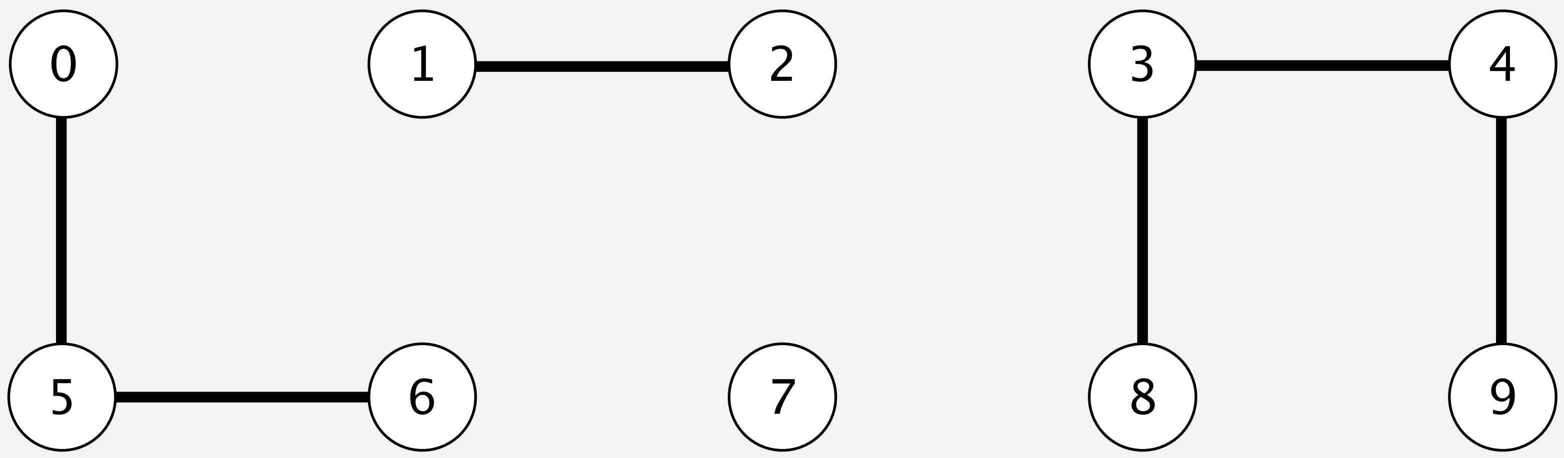
Quick-find demo

union(5, 0)



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

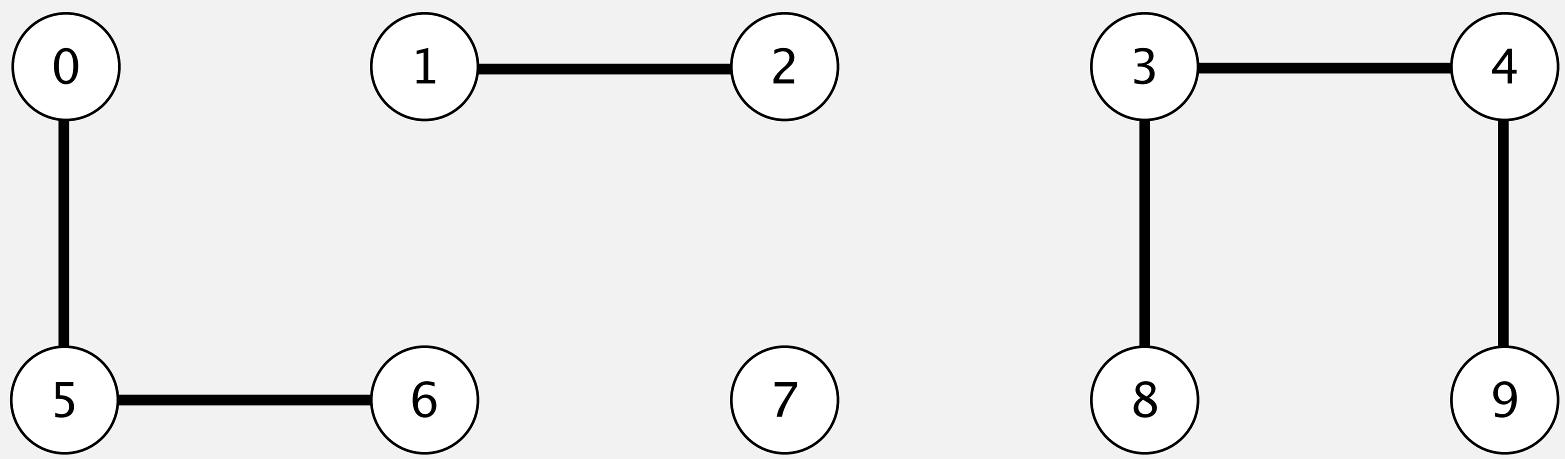
Quick-find demo



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

Quick-find demo

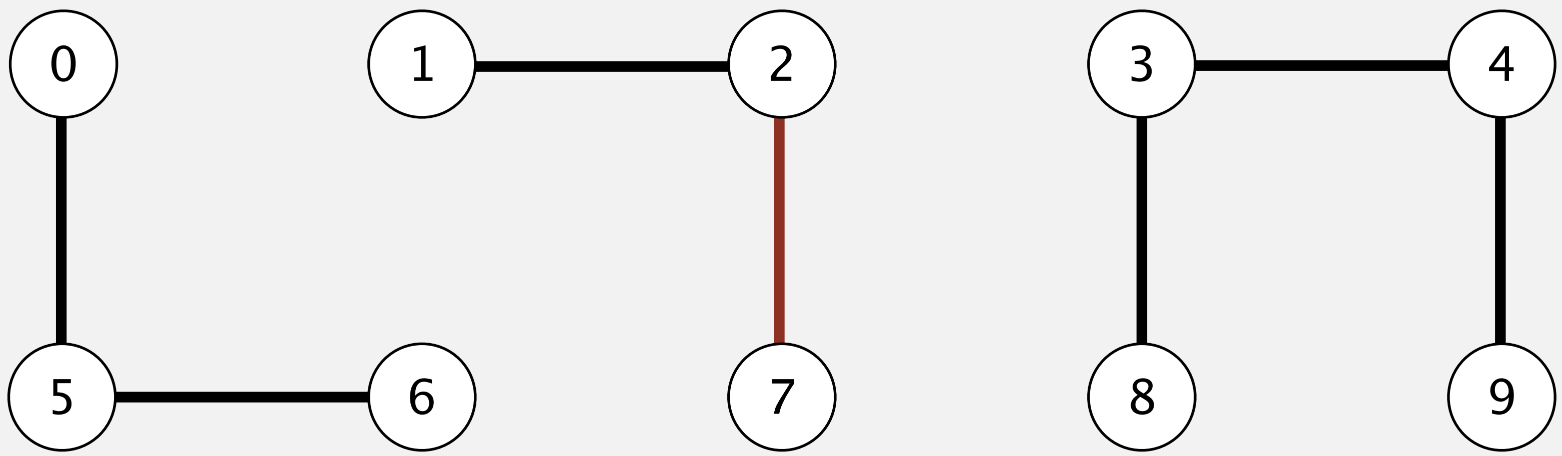
union(7, 2)



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

Quick-find demo

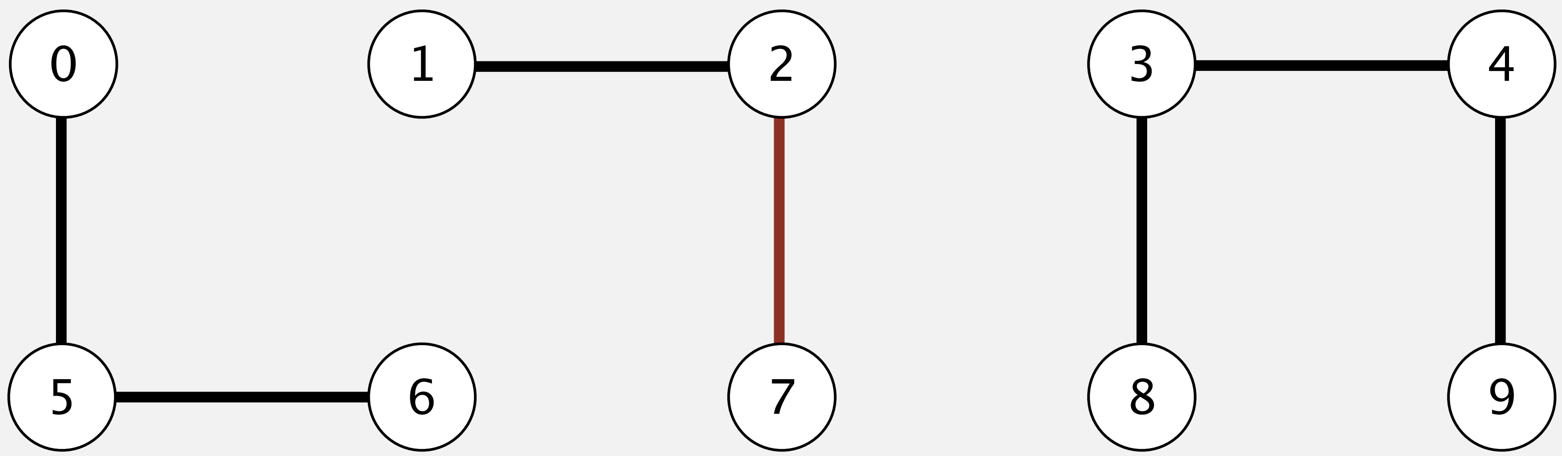
union(7, 2)



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

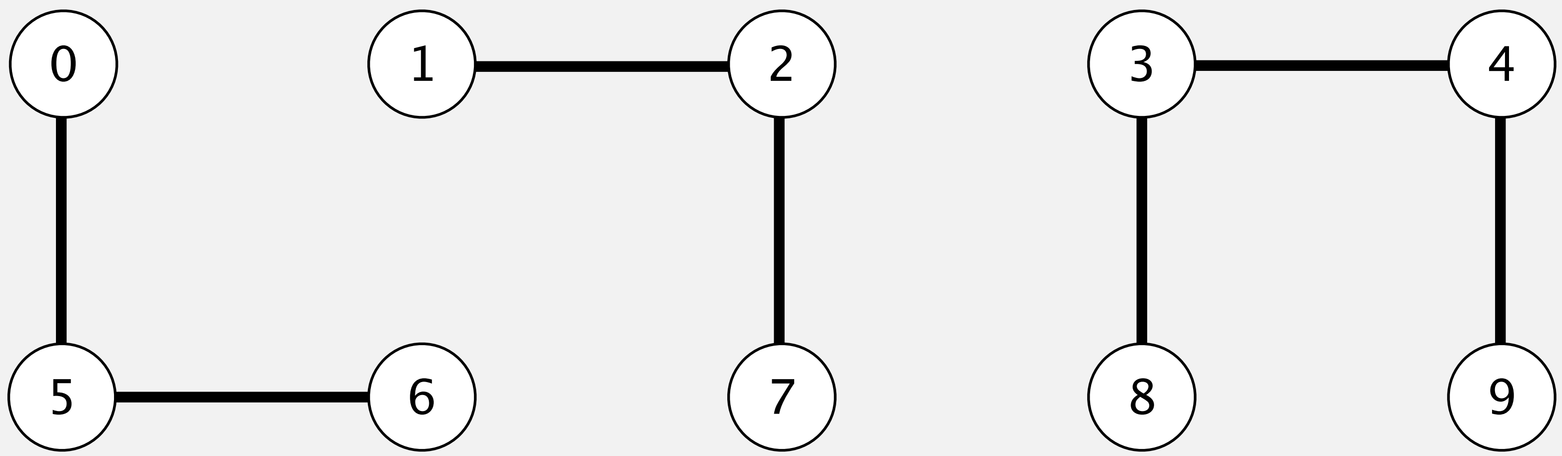
Quick-find demo

union(7, 2)



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

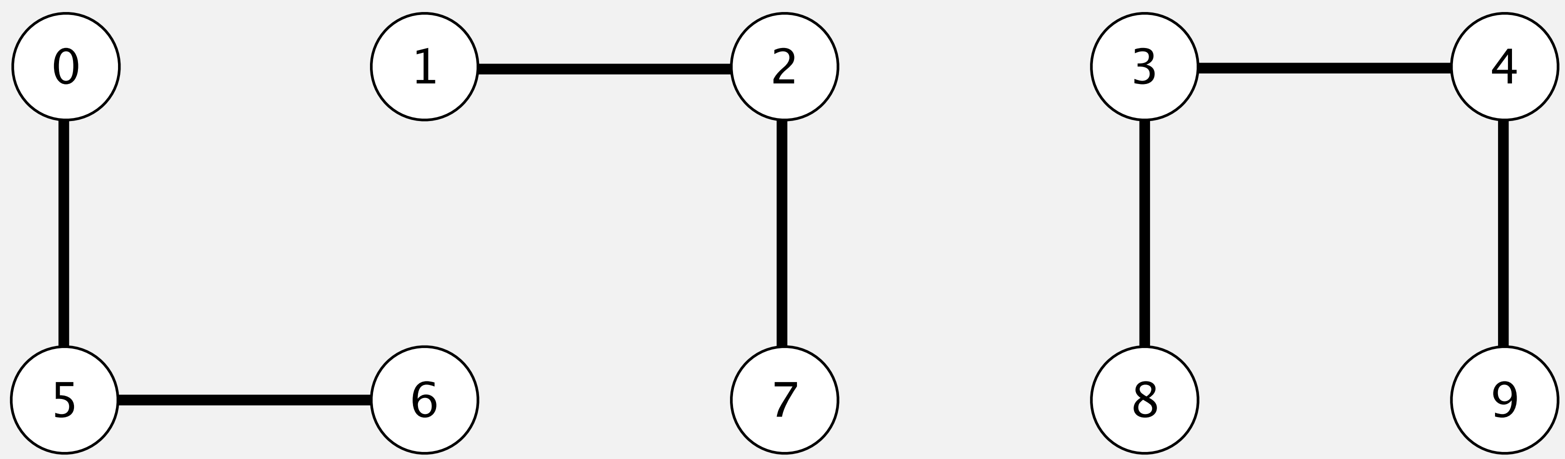
Quick-find demo



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

Quick-find demo

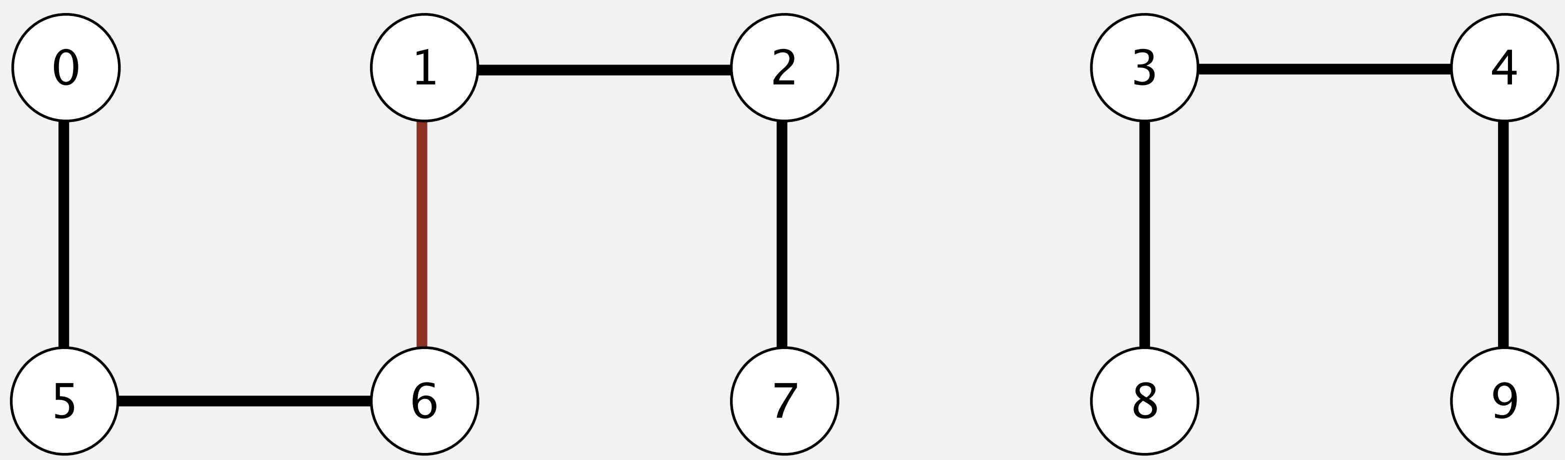
union(6, 1)



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

Quick-find demo

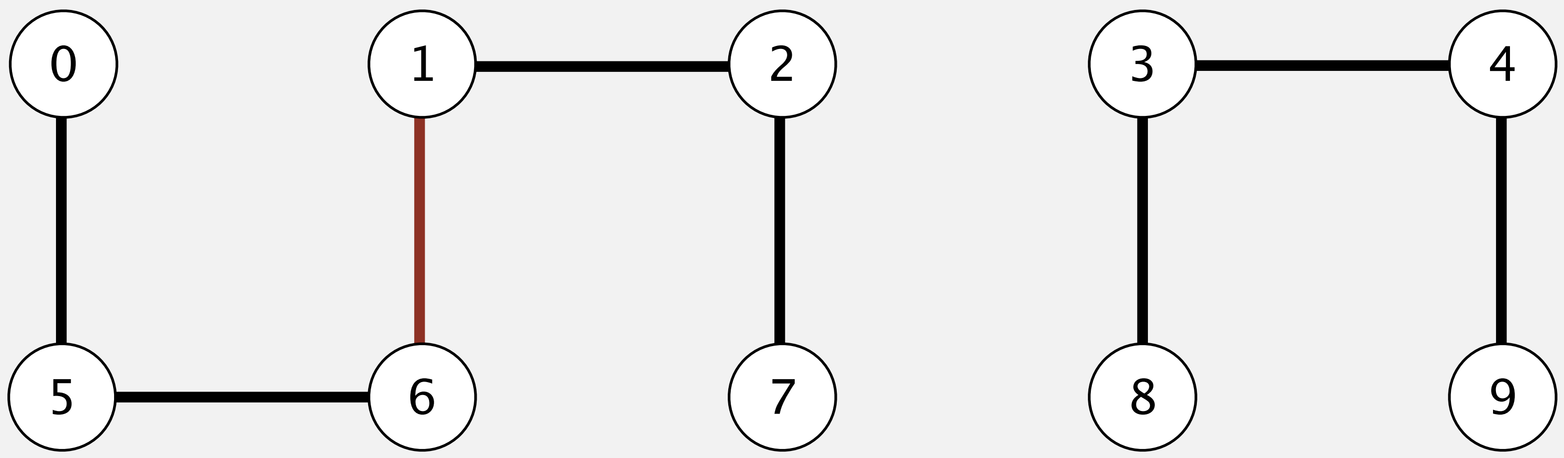
union(6, 1)



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

Quick-find demo

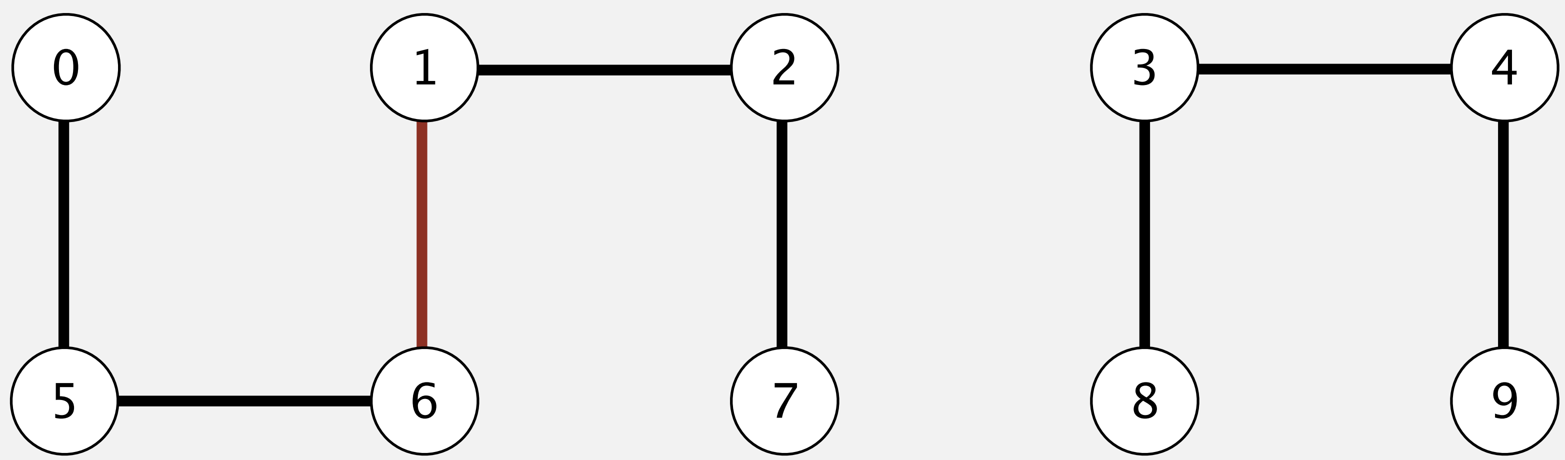
union(6, 1)



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

Quick-find demo

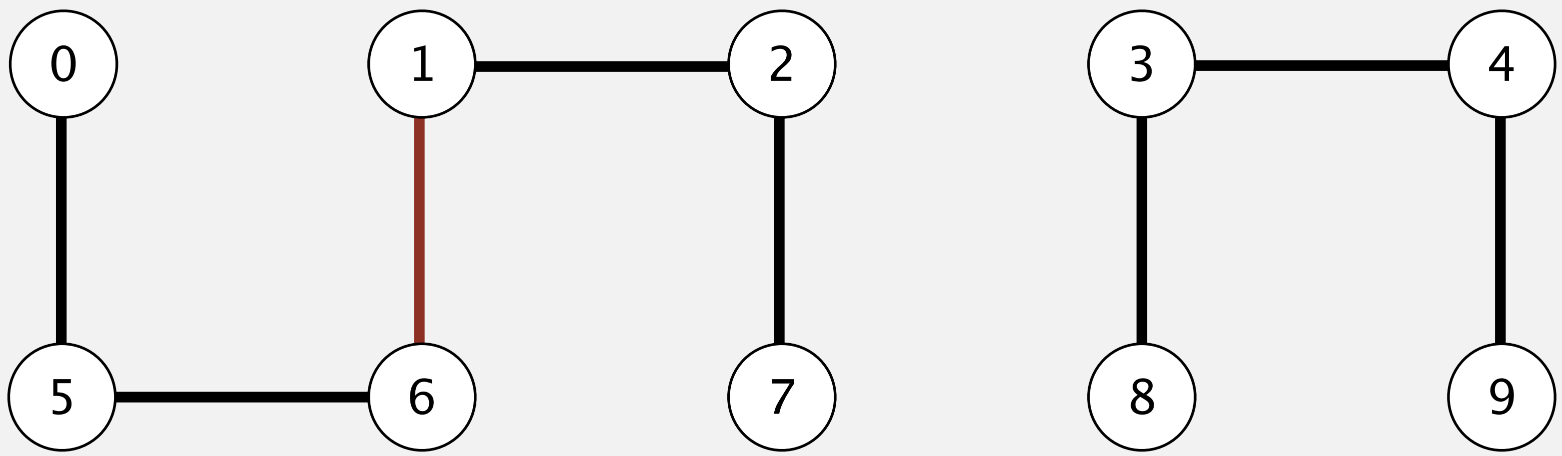
union(6, 1)



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

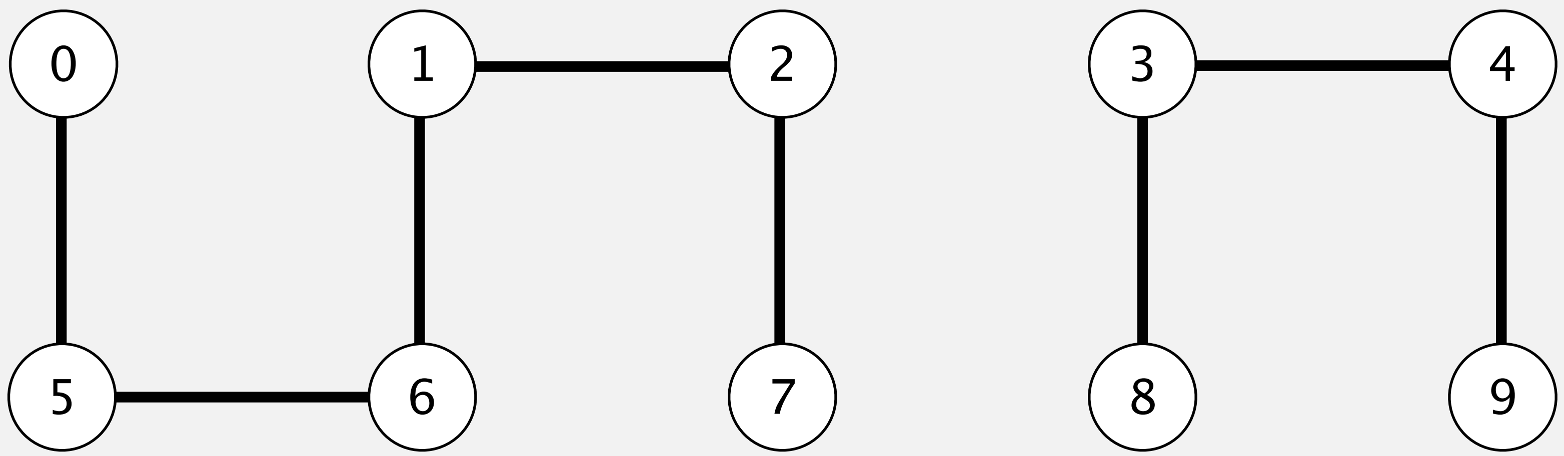
Quick-find demo

union(6, 1)



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

Quick-find demo

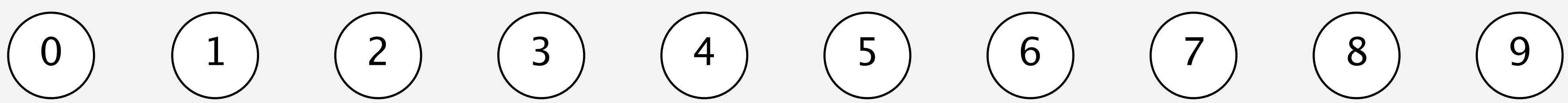


	0	1	2	3	4	5	6	7	8	9
id[]	1	1	1	8	8	1	1	1	8	8

http://algs4.cs.princeton.edu

1.5 QUICK-UNION DEMO

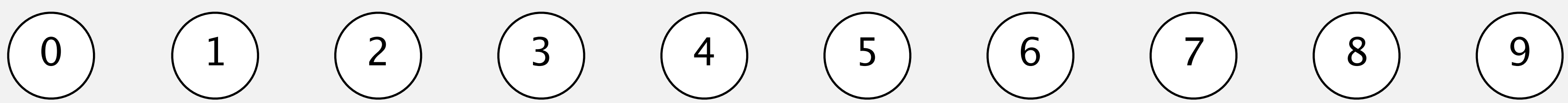
Quick-union demo



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

Quick-union demo

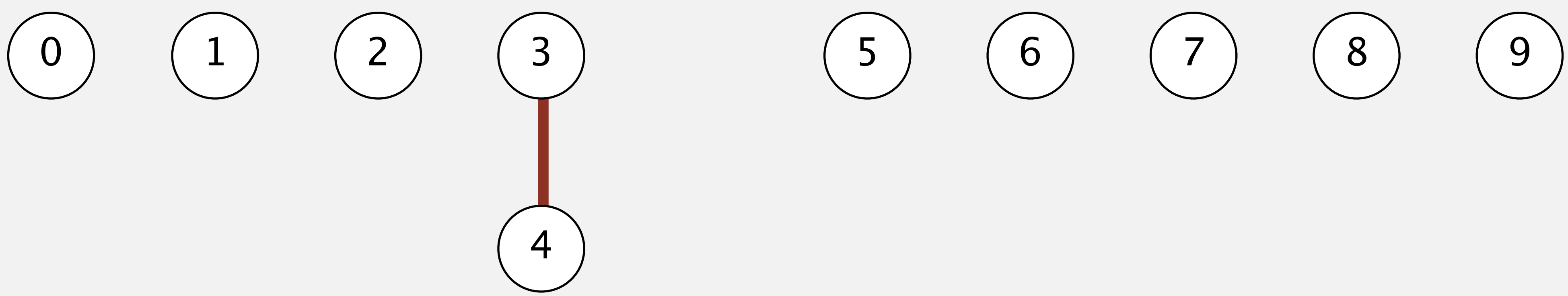
union(4, 3)



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

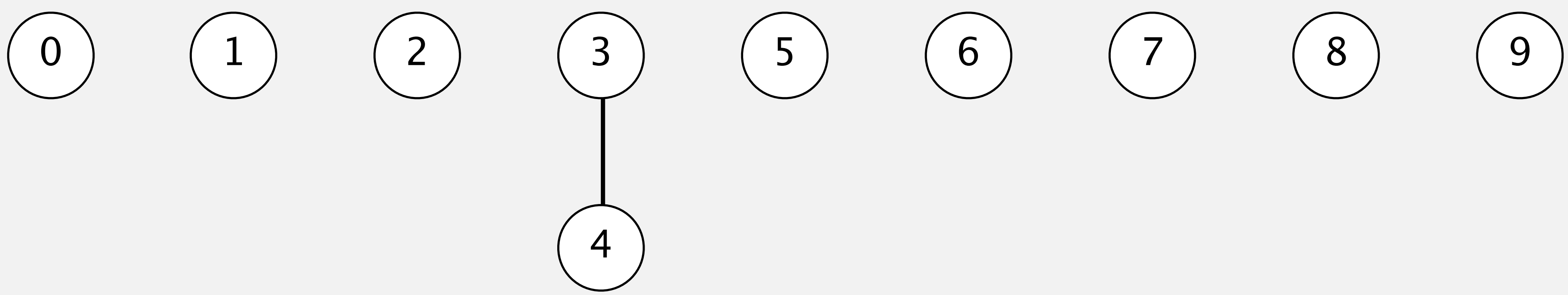
Quick-union demo

union(4, 3)



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

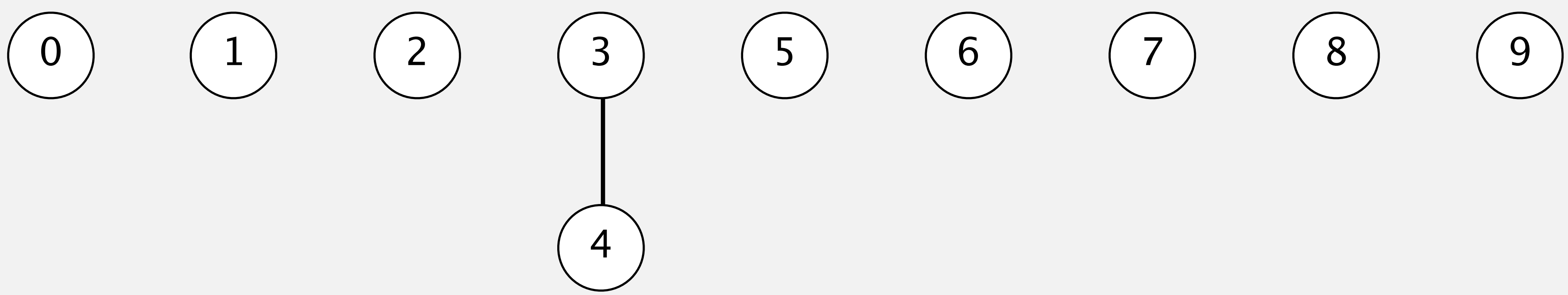
Quick-union demo



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

Quick-union demo

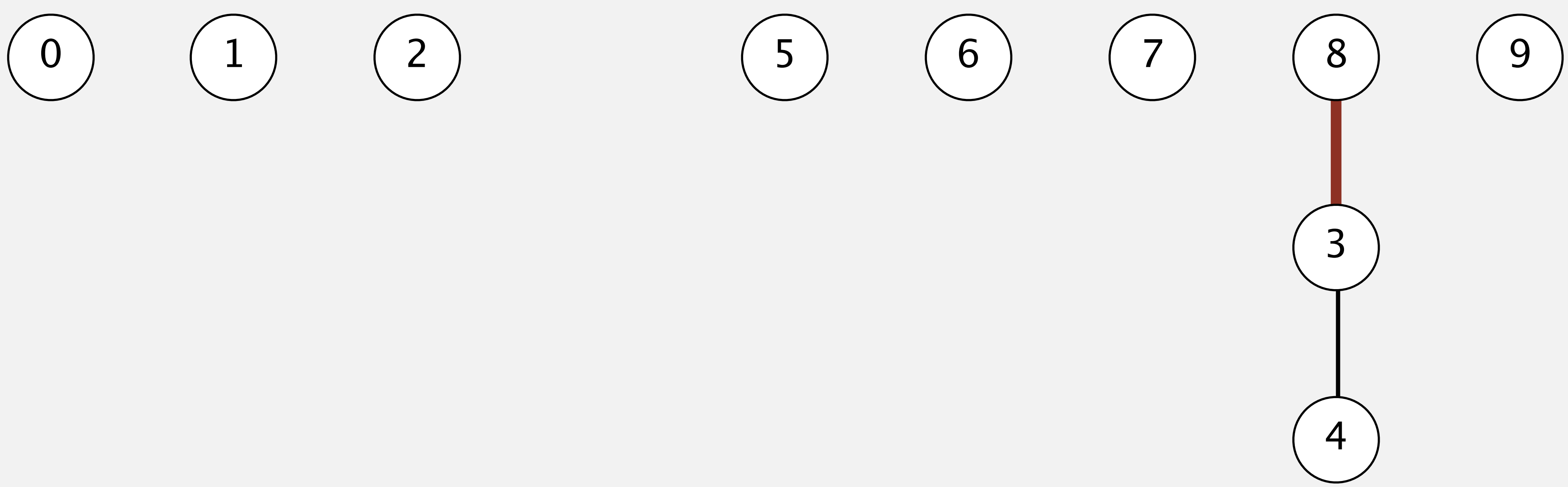
union(3, 8)



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

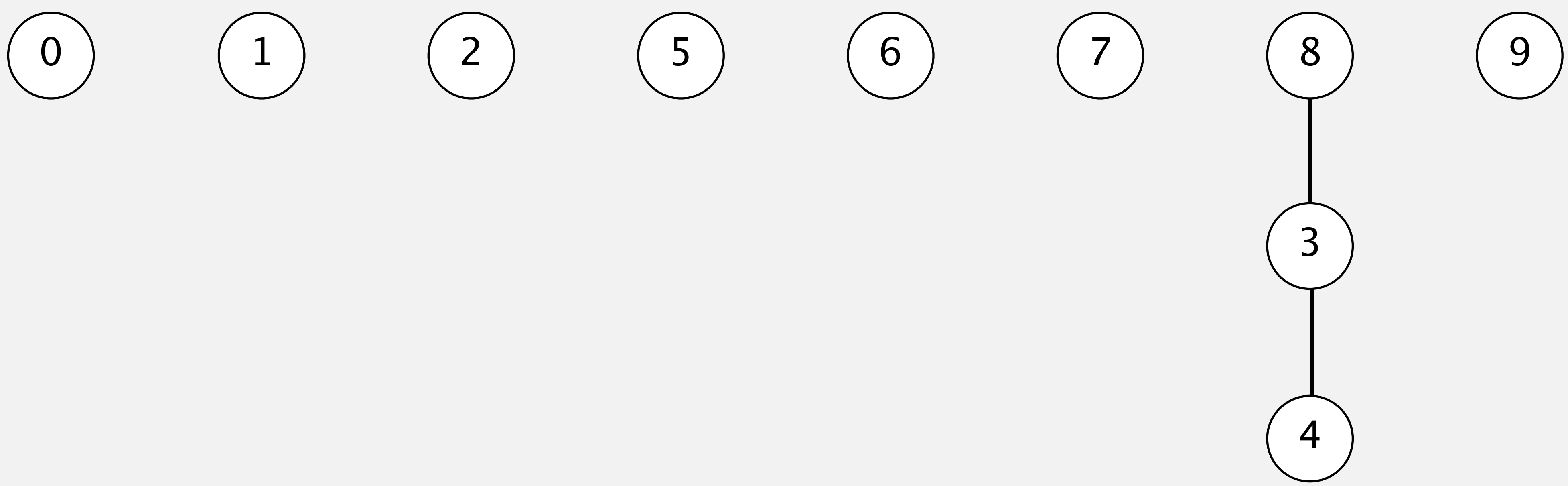
Quick-union demo

union(3, 8)



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

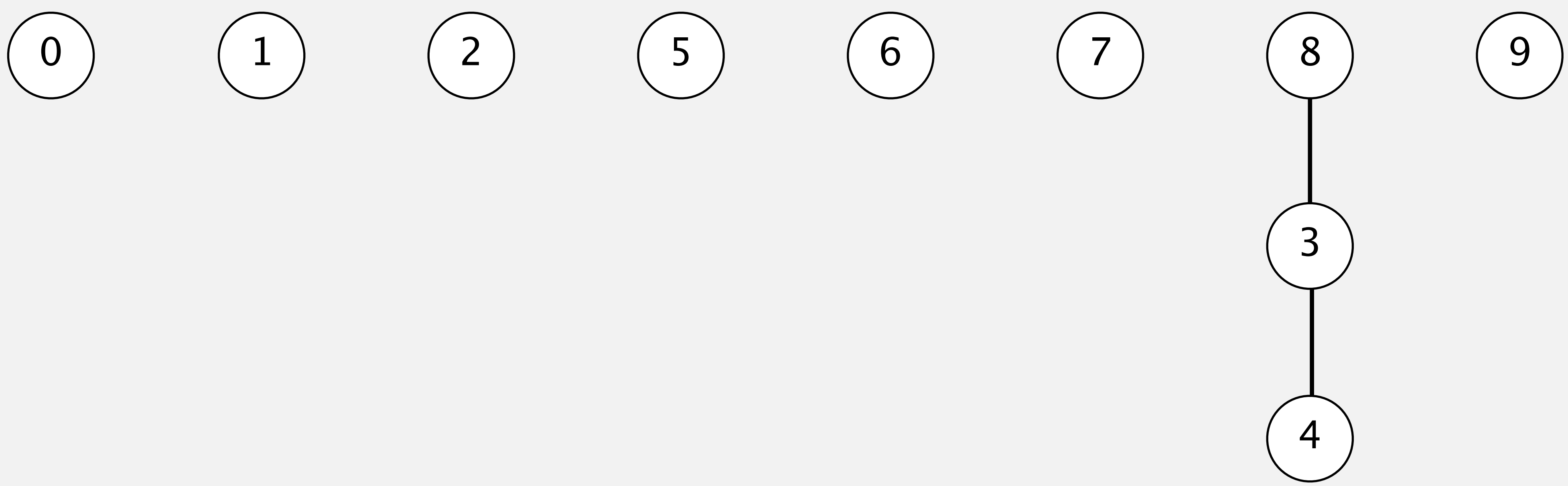
Quick-union demo



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

Quick-union demo

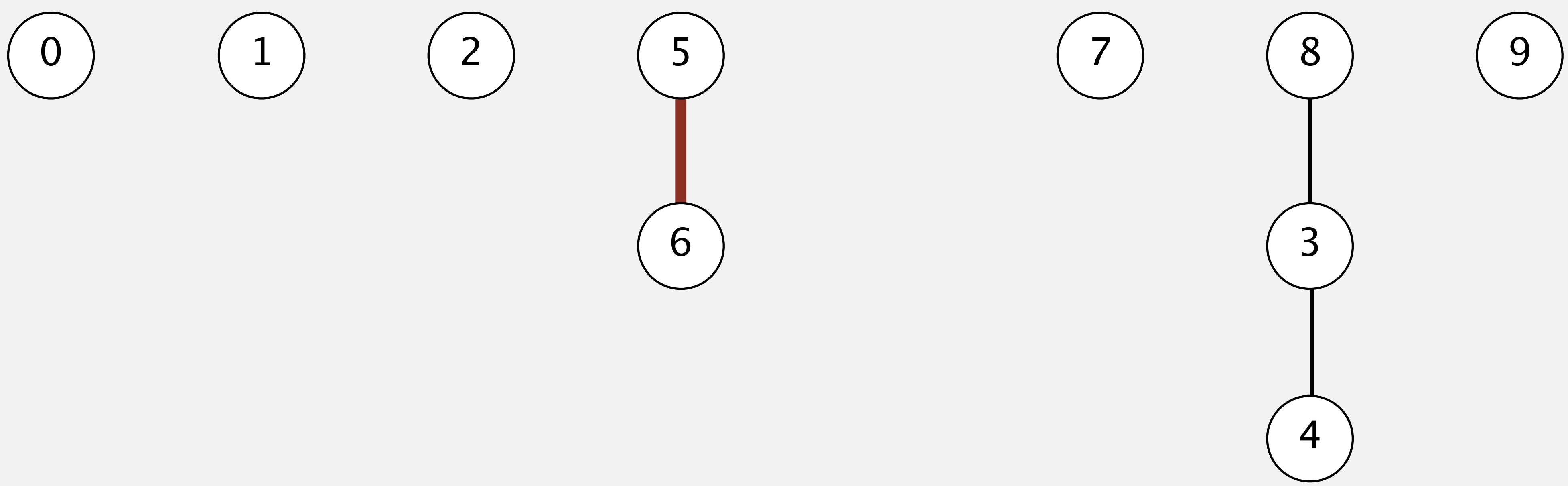
union(6, 5)



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

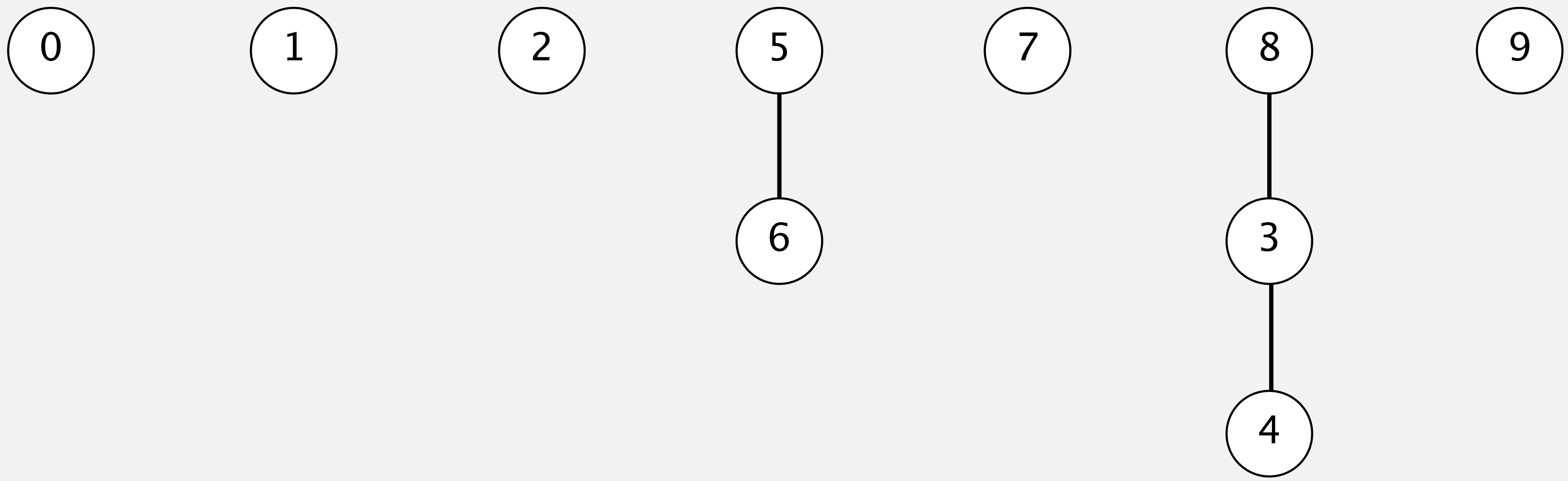
Quick-union demo

union(6, 5)



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

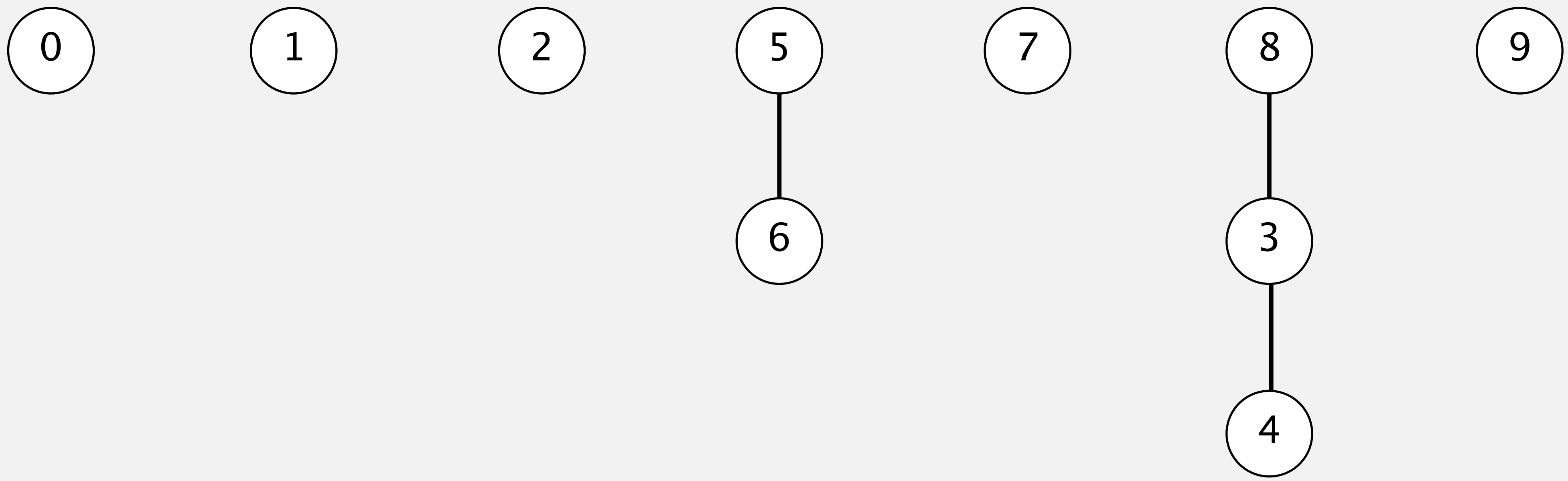
Quick-union demo



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

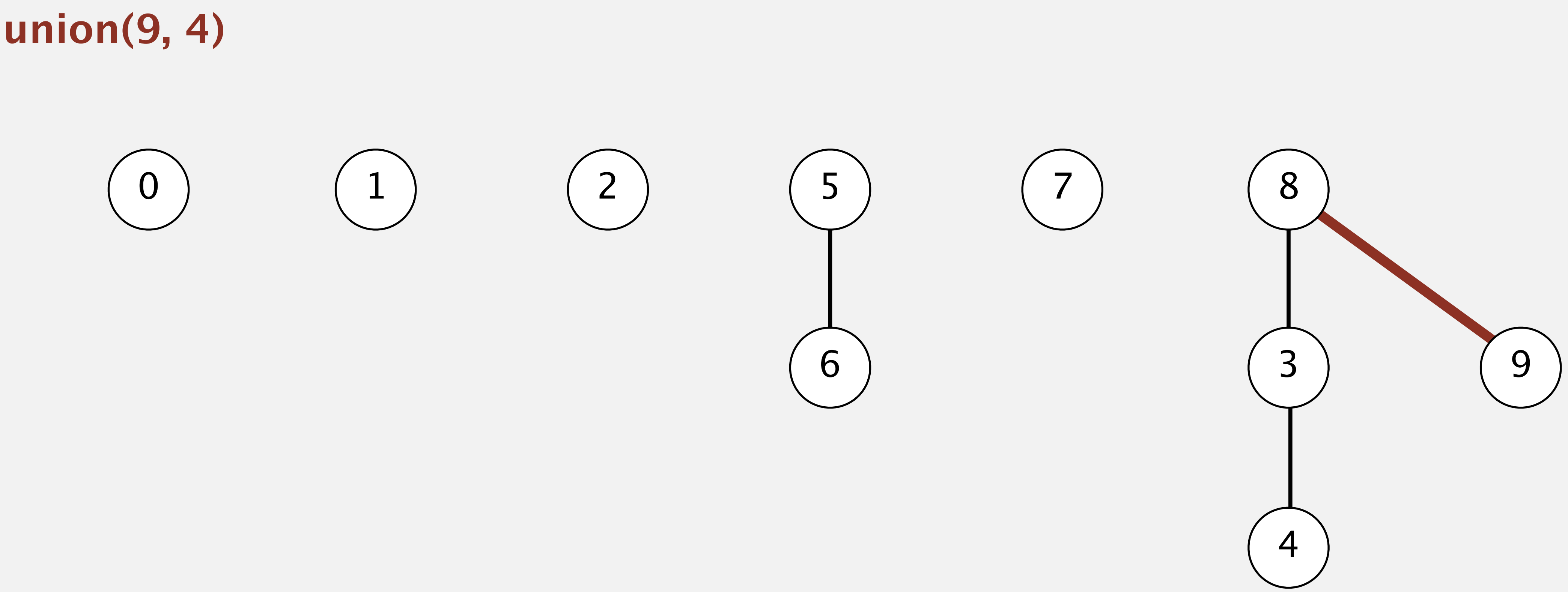
Quick-union demo

union(9, 4)



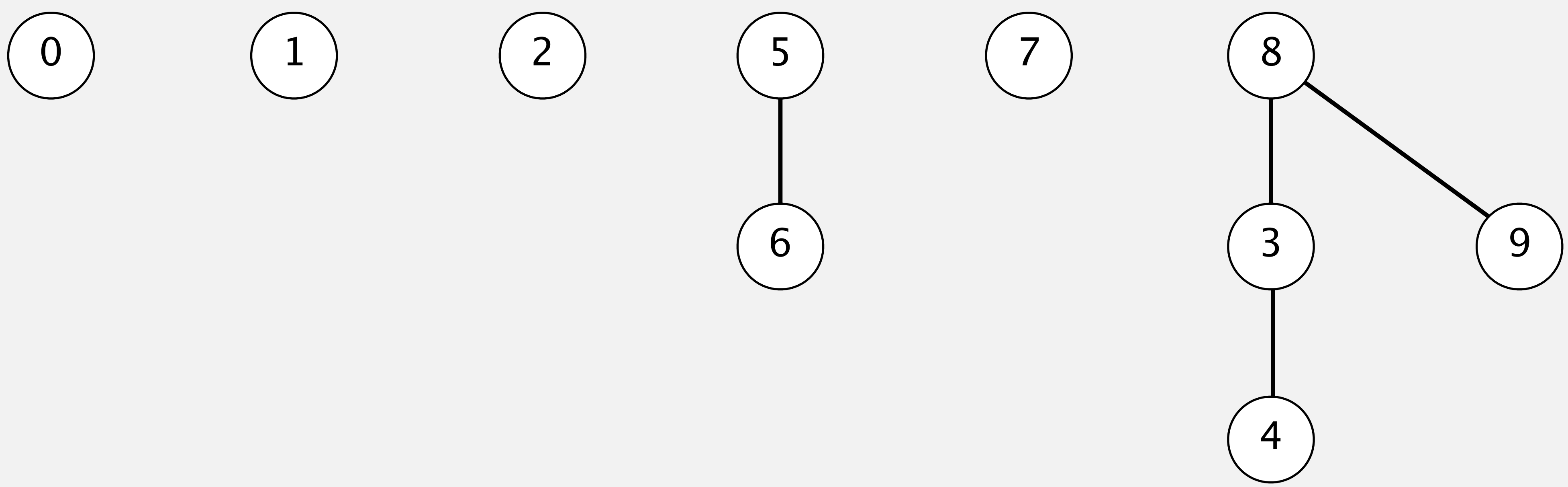
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	8	3	5	5	7	8	9

Quick-union demo



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

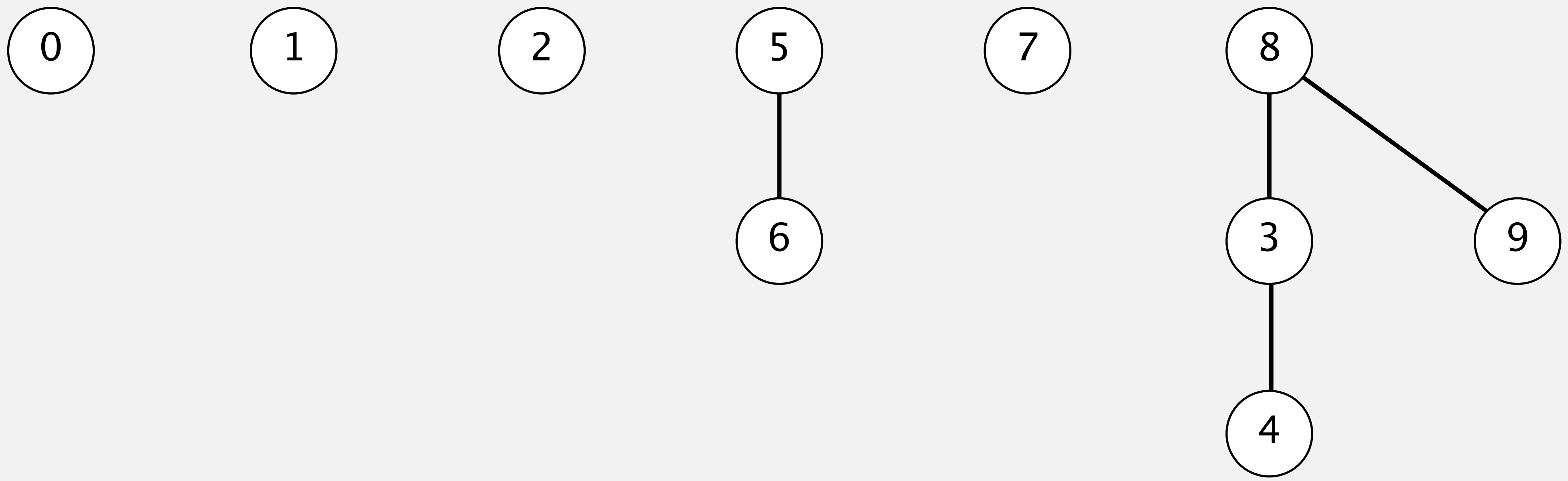
Quick-union demo



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

Quick-union demo

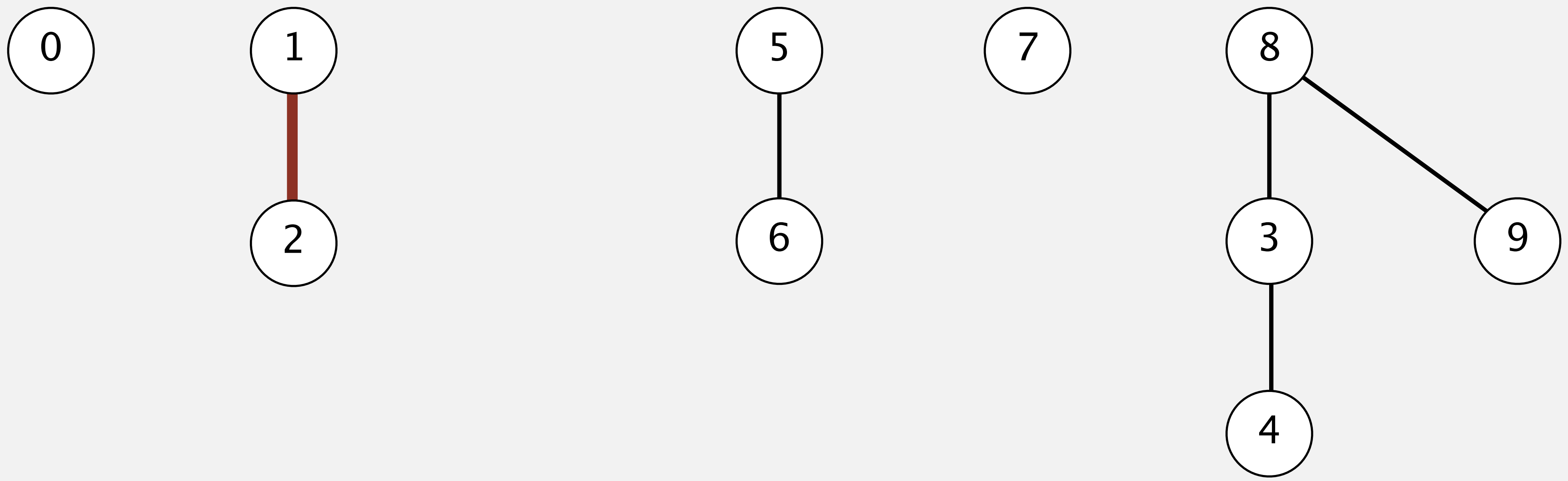
union(2, 1)



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

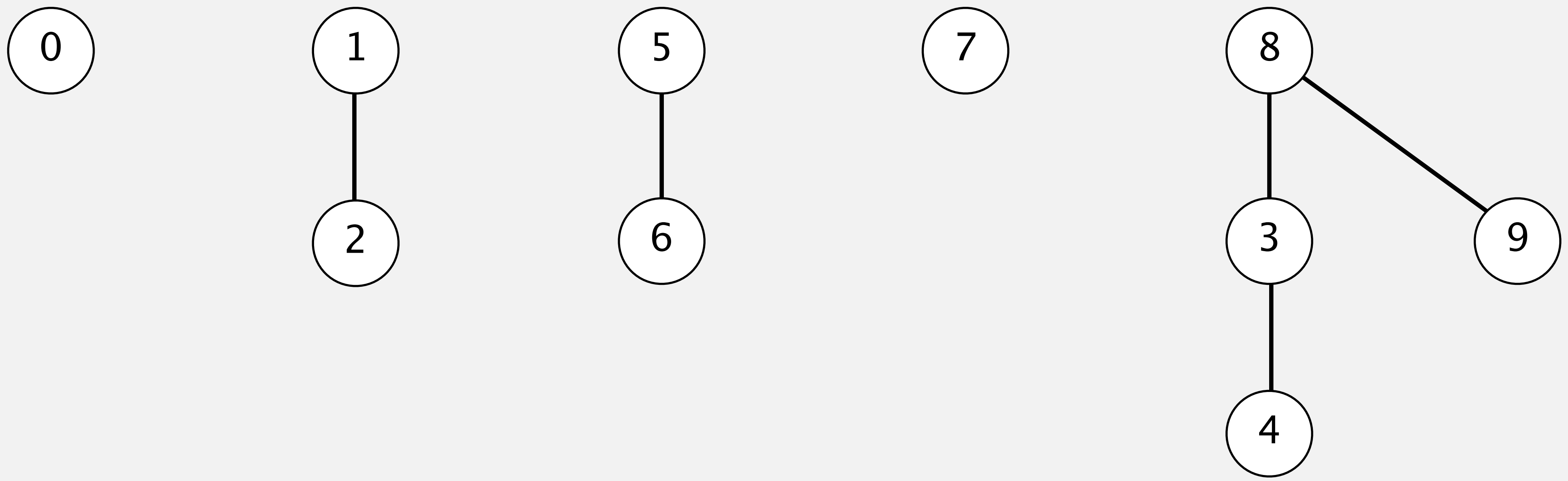
Quick-union demo

union(2, 1)



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

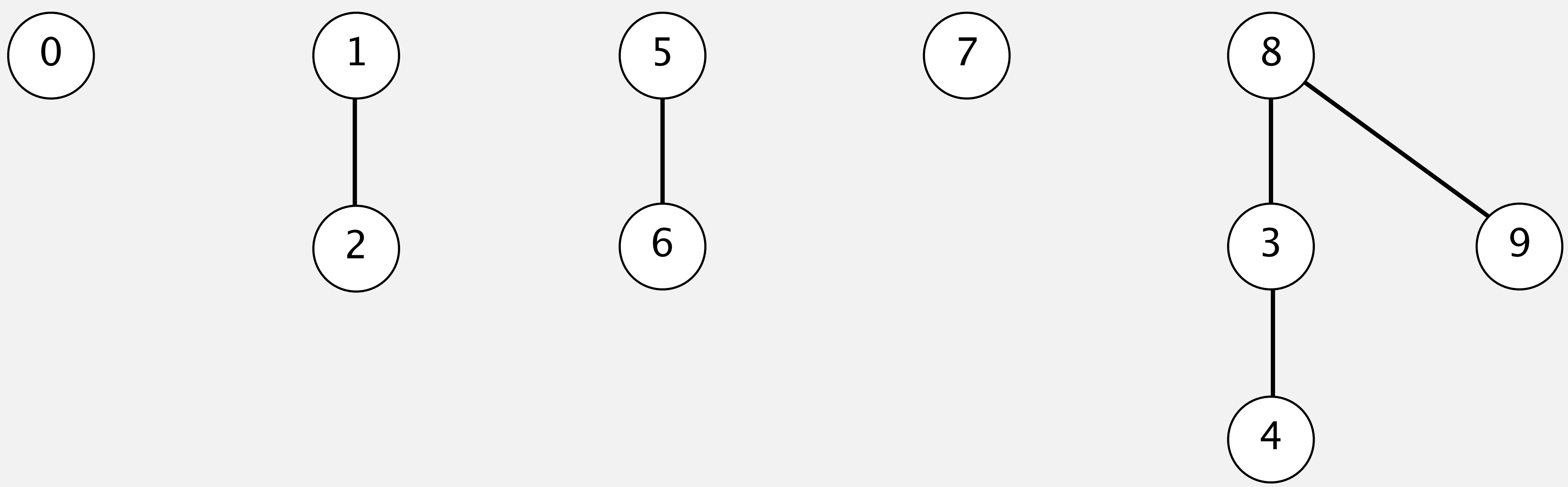
Quick-union demo



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

Quick-union demo

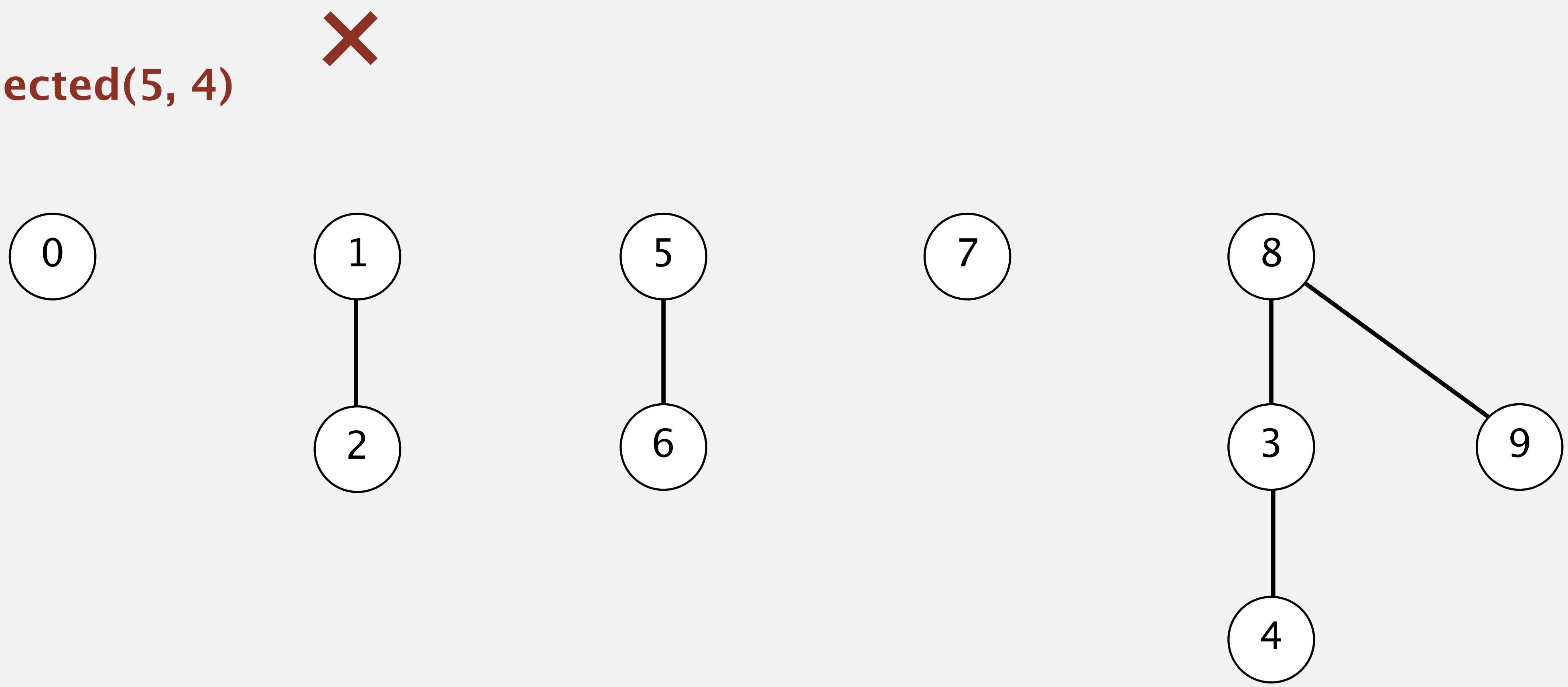
connected(8, 9) ✔



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

Quick-union demo

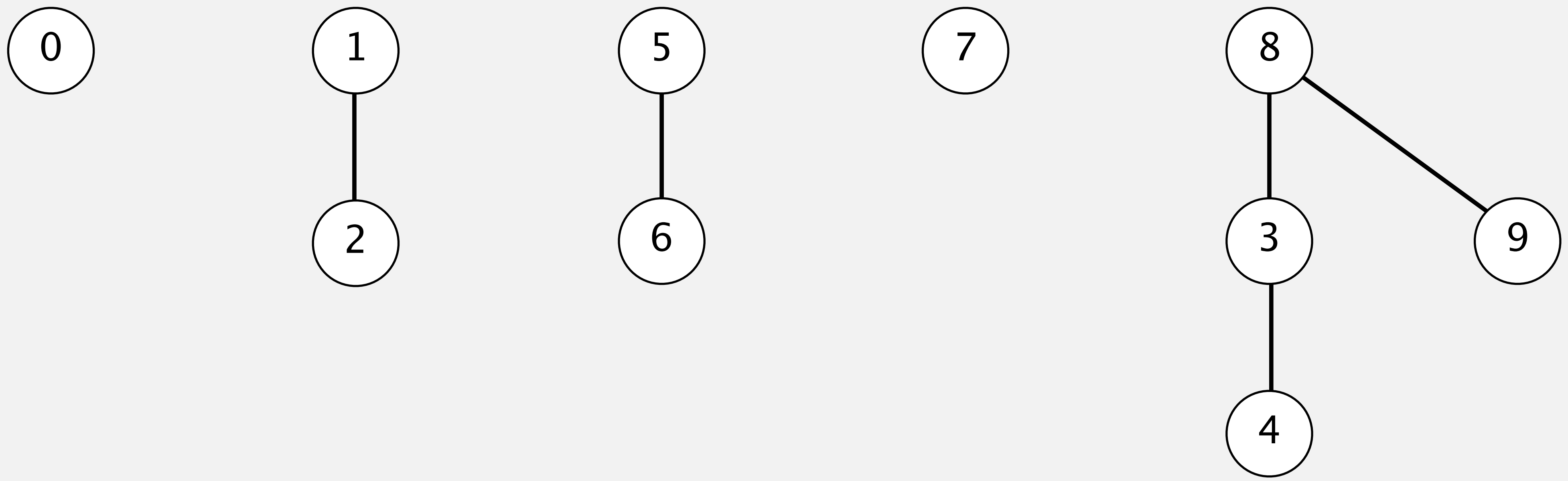
connected(5, 4)



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

Quick-union demo

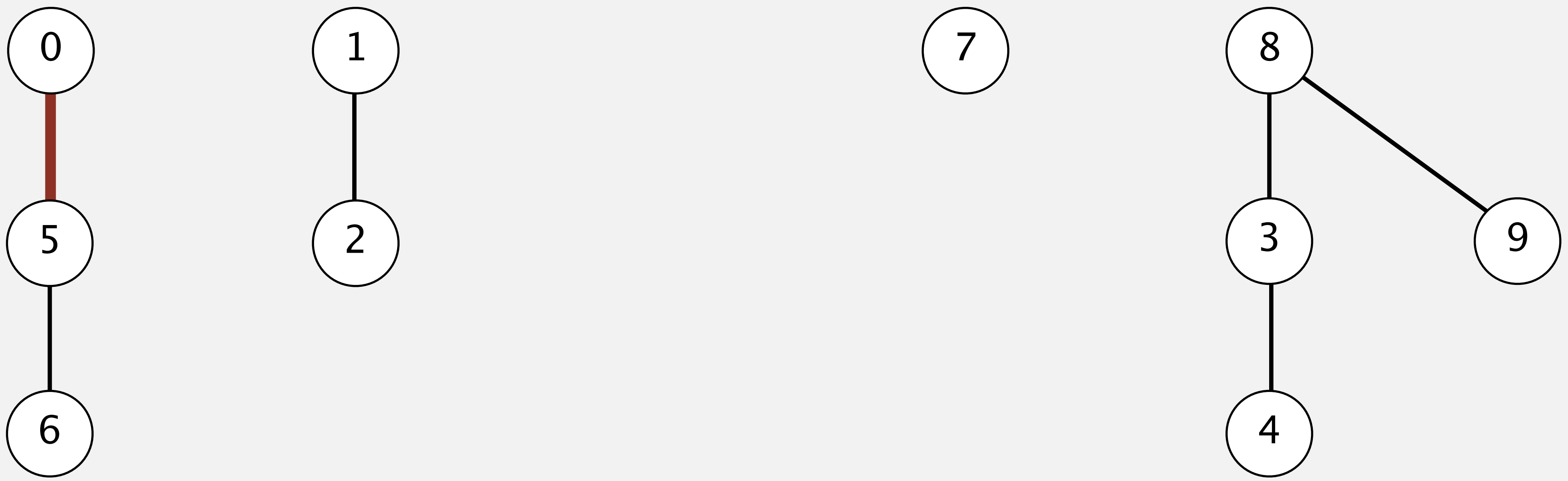
union(5, 0)



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

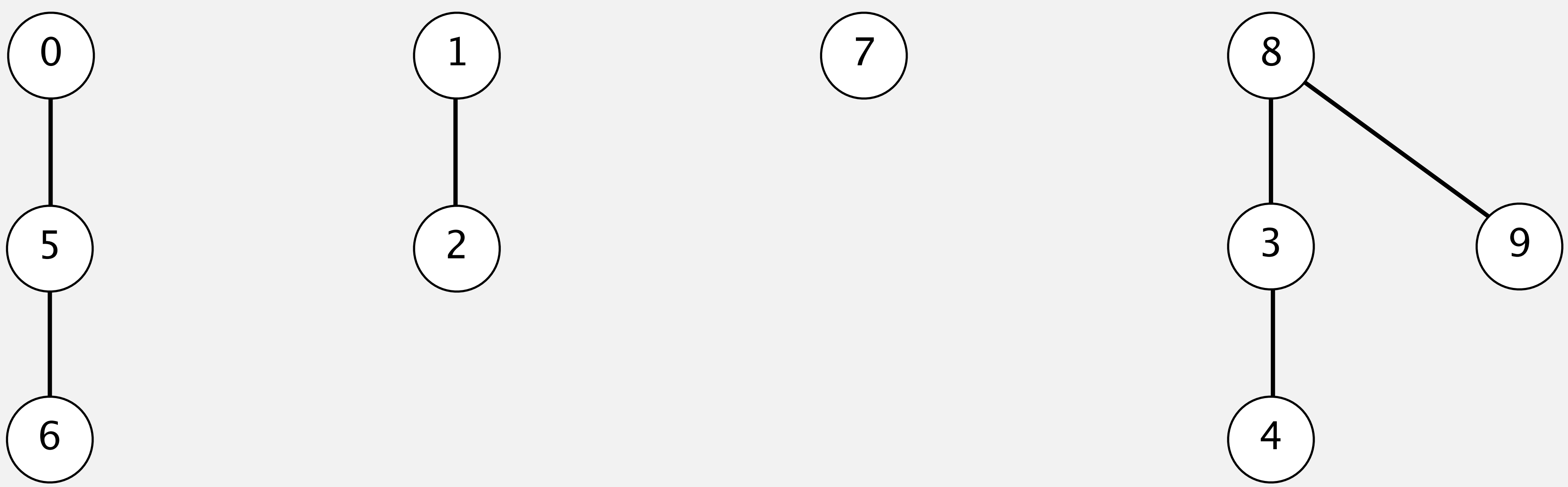
Quick-union demo

union(5, 0)



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

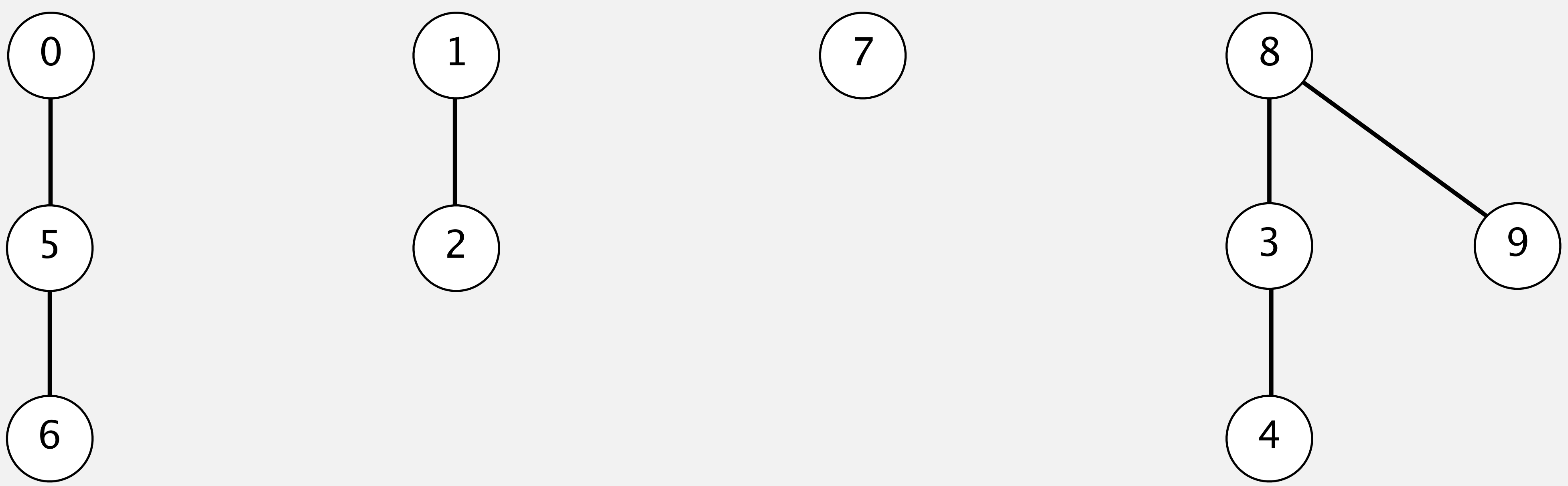
Quick-union demo



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

Quick-union demo

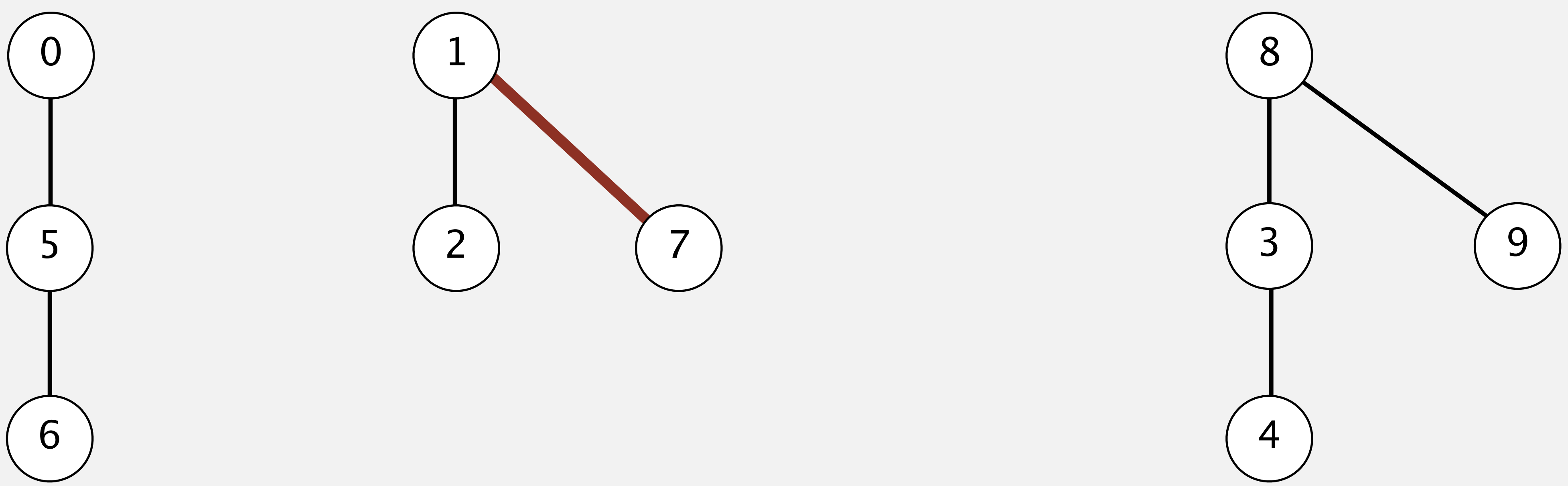
union(7, 2)



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

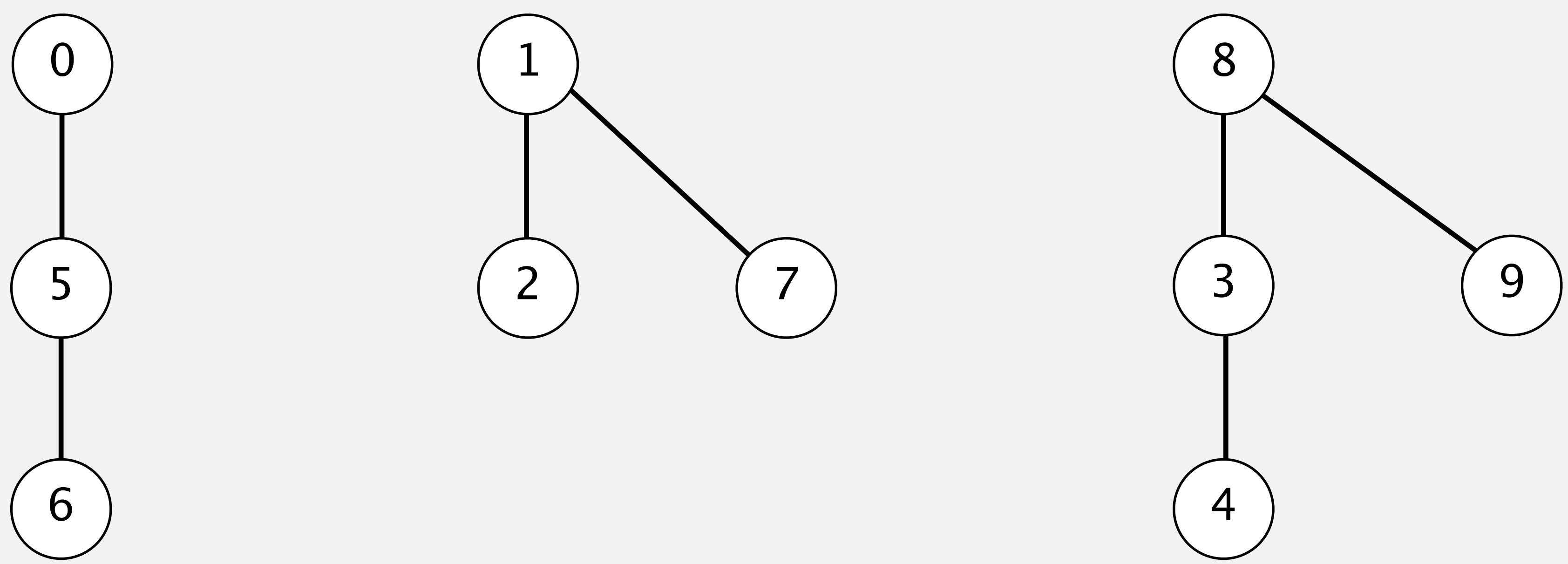
Quick-union demo

union(7, 2)



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

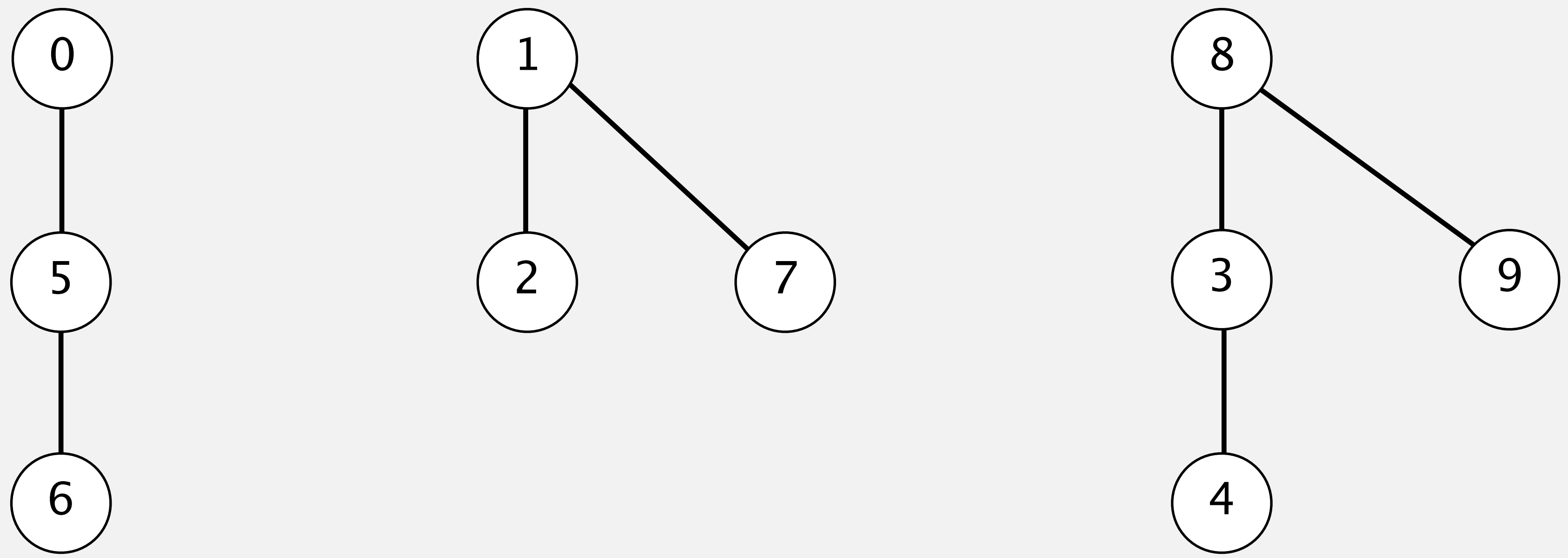
Quick-union demo



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

Quick-union demo

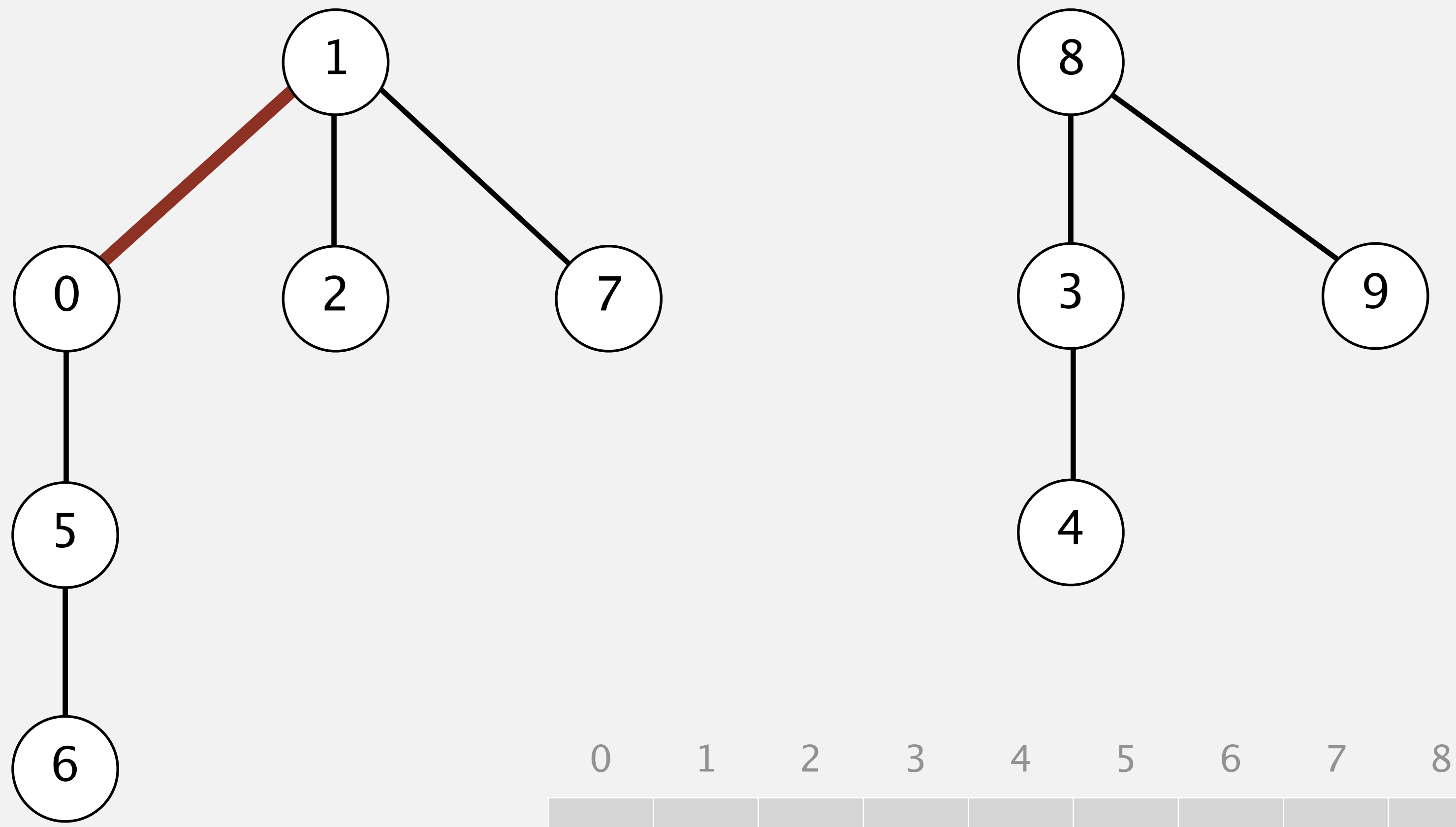
union(6, 1)



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

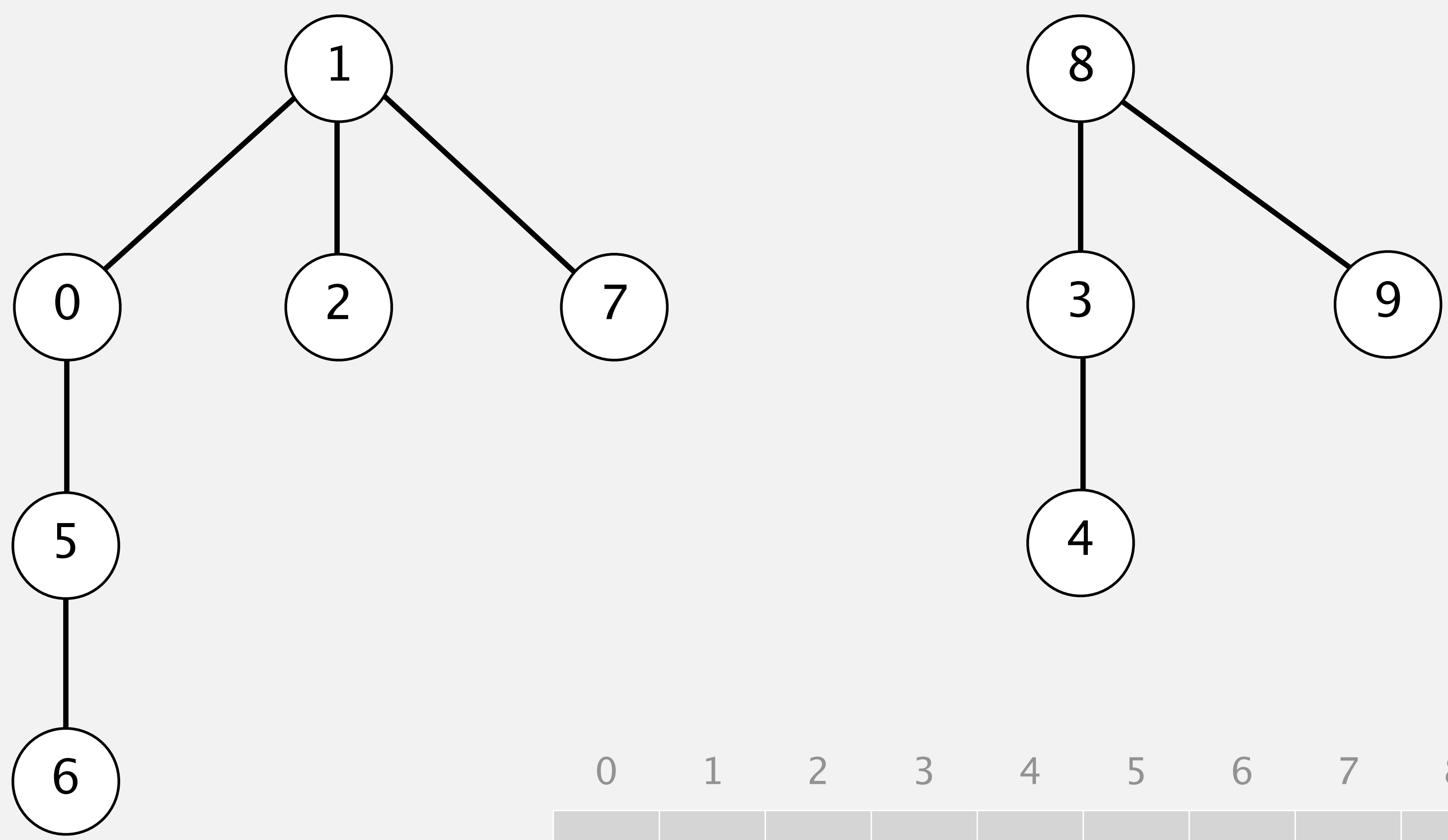
Quick-union demo

union(6, 1)



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

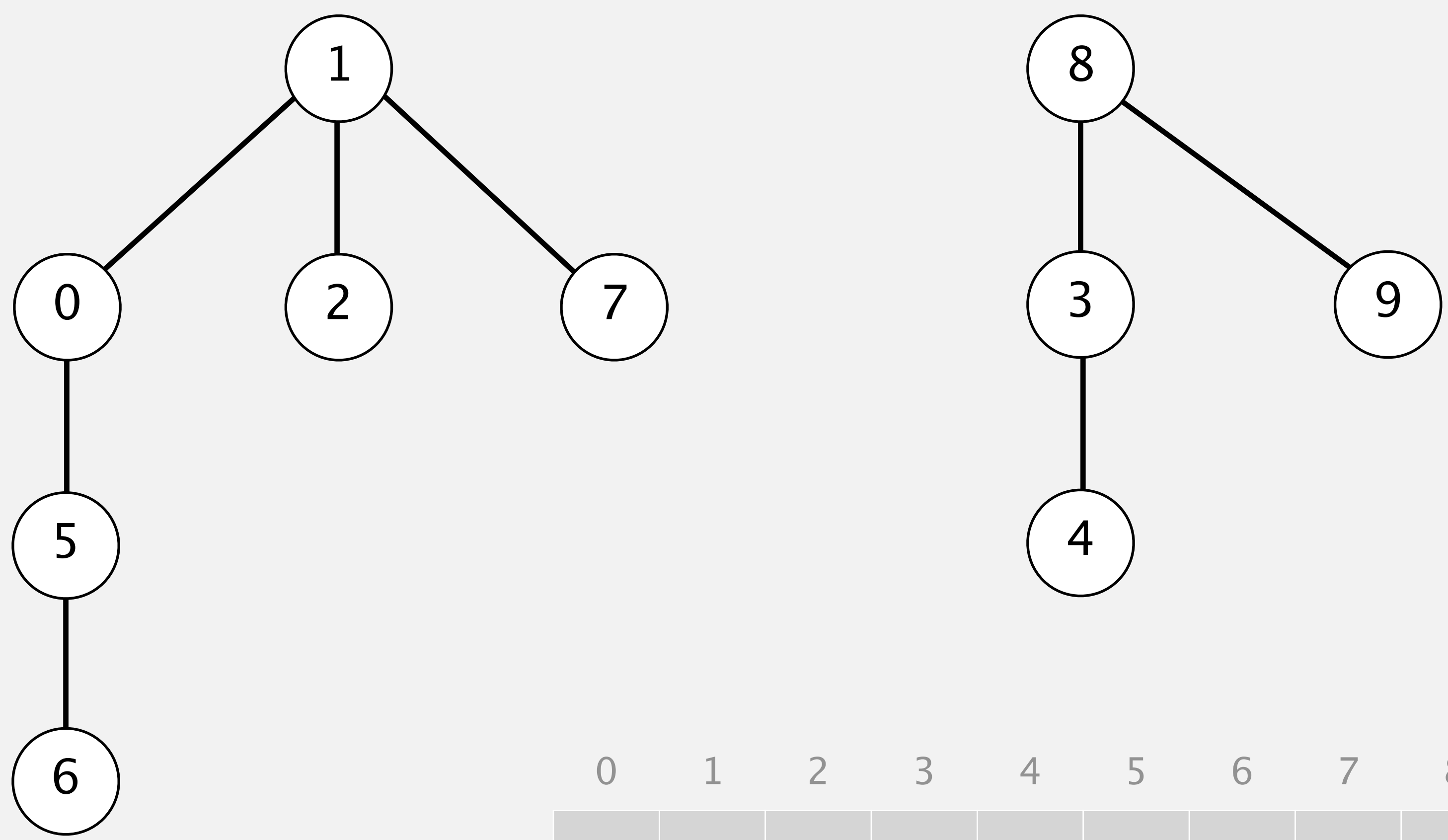
Quick-union demo



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

Quick-union demo

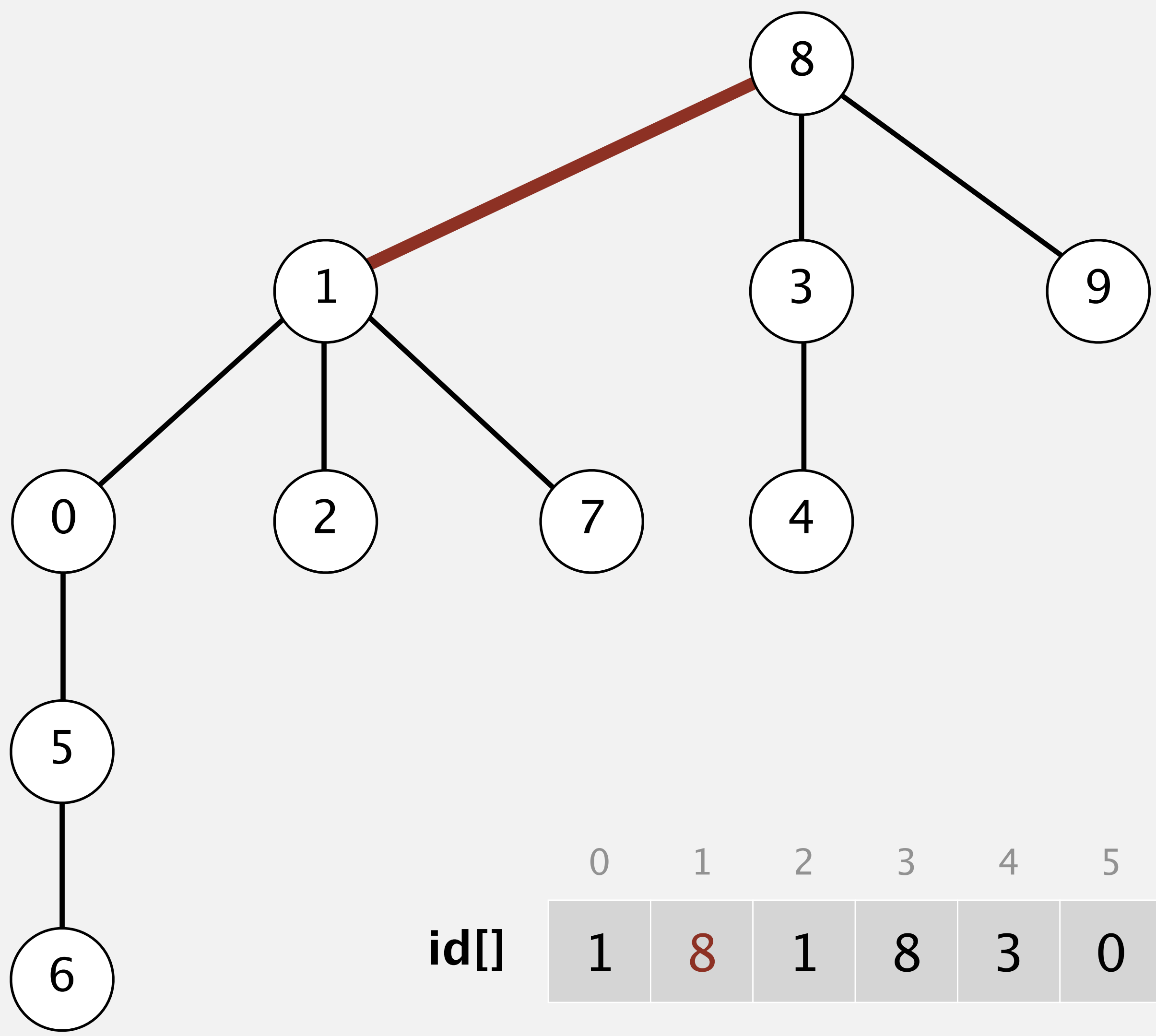
union(7, 3)



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

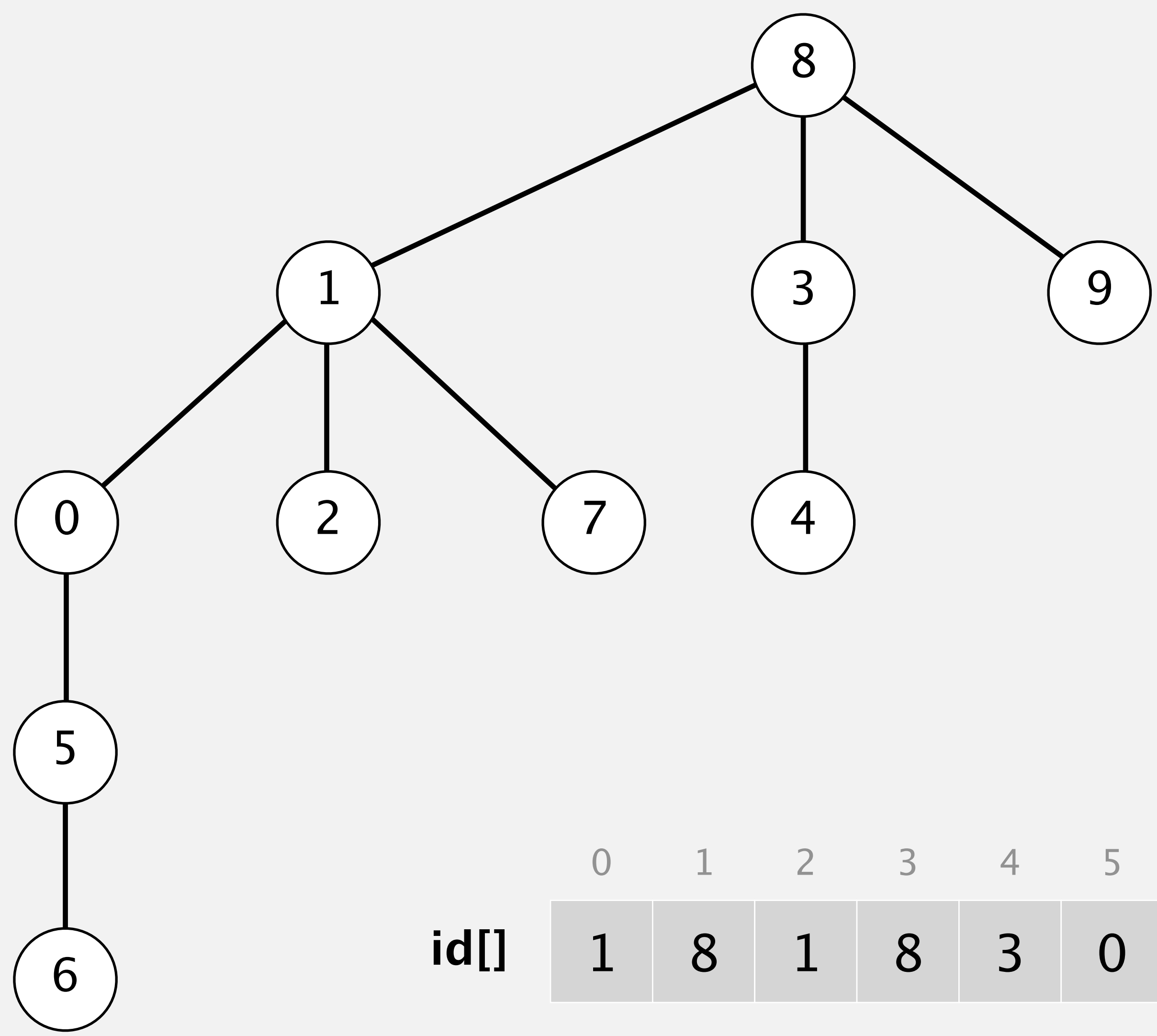
Quick-union demo

union(7, 3)



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

Quick-union demo

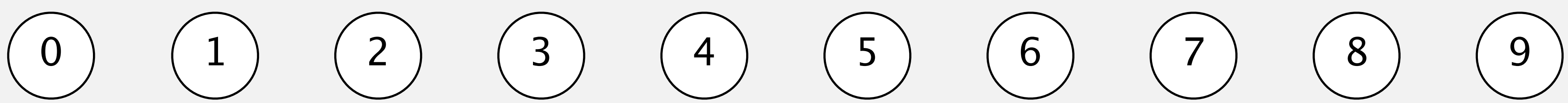


	0	1	2	3	4	5	6	7	8	9
id[]	1	8	1	8	3	0	5	1	8	8

http://algs4.cs.princeton.edu

1.5 Weighted Quick-Union Demo

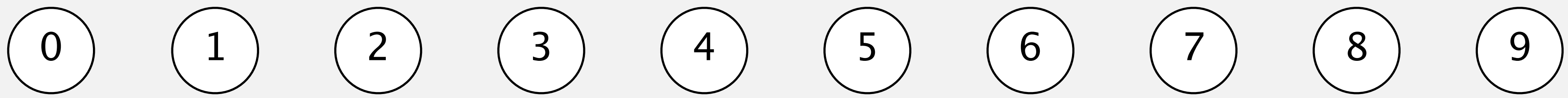
Weighted quick-union demo



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

Weighted quick-union demo

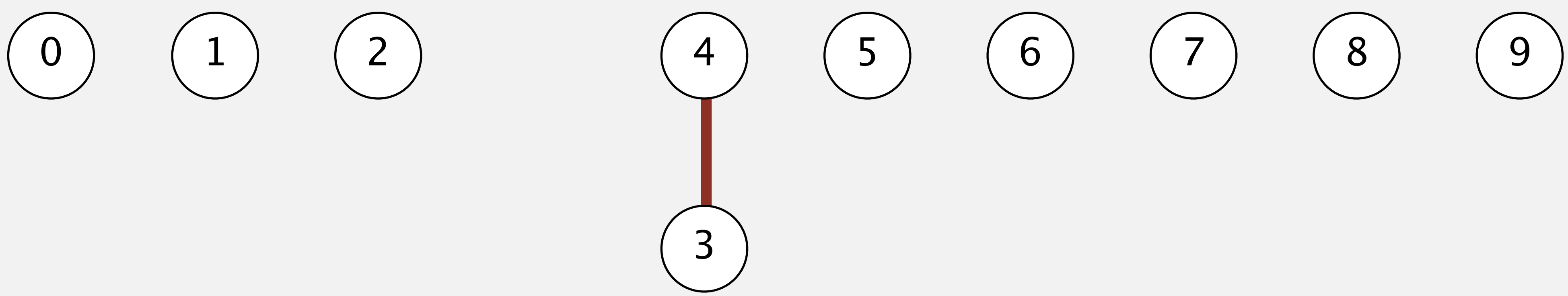
union(4, 3)



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

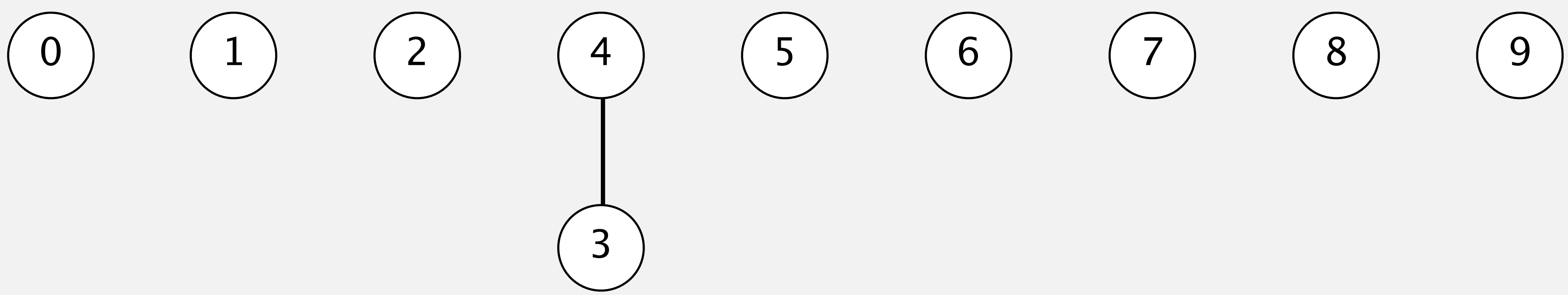
Weighted quick-union demo

union(4, 3)



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

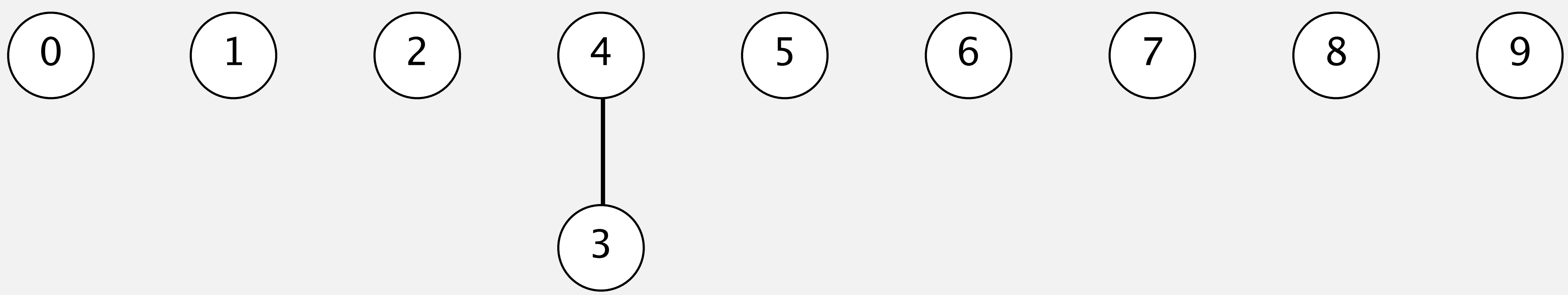
Weighted quick-union demo



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

Weighted quick-union demo

union(3, 8)

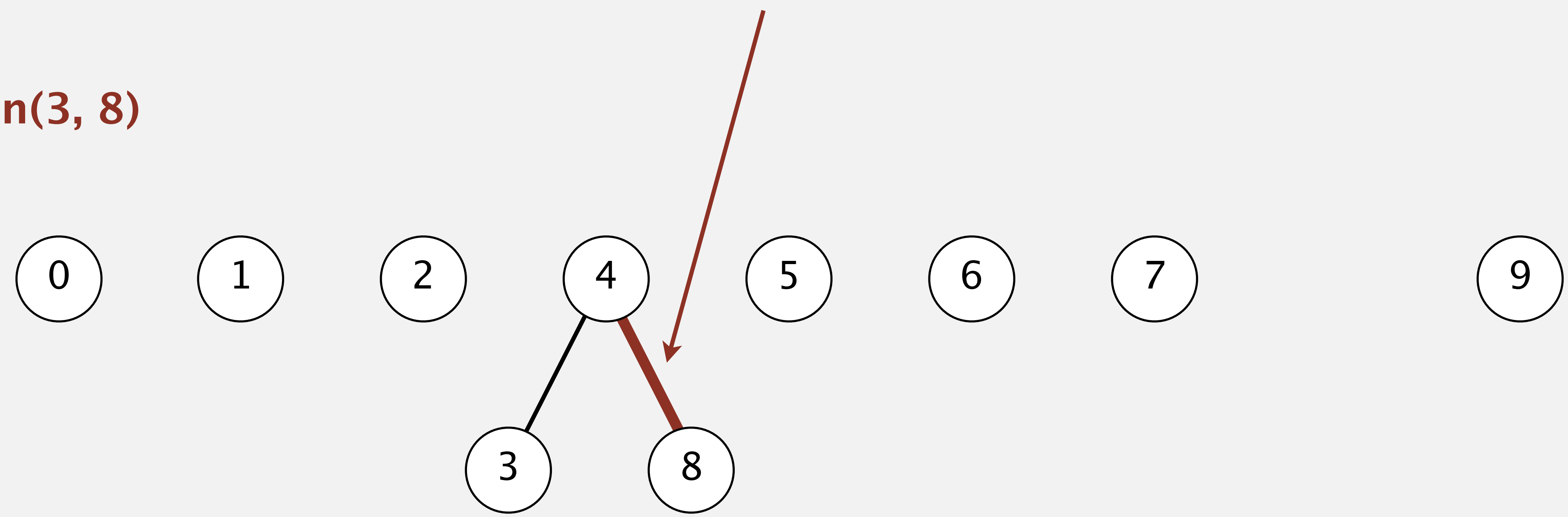


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	4	4	5	6	7	8	9

Weighted quick-union demo

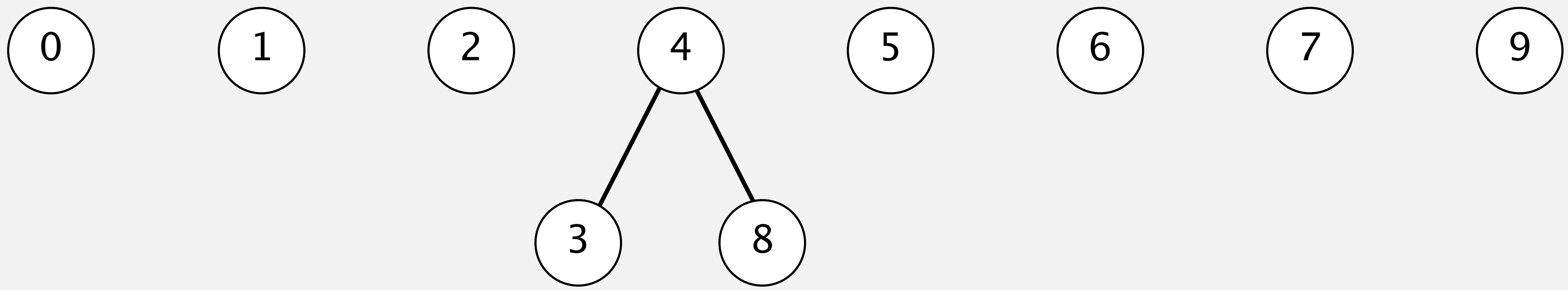
weighting: make 8 point to 4 (instead of 4 to 8)

union(3, 8)



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

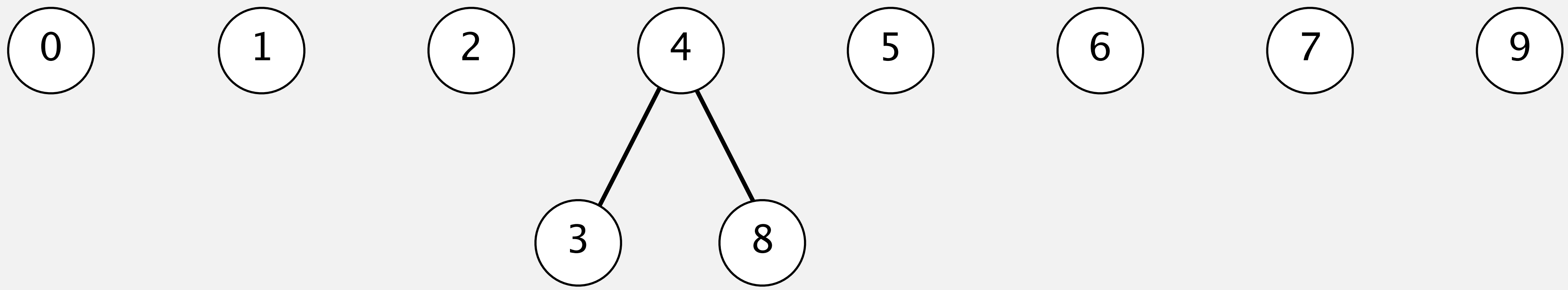
Weighted quick-union demo



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

Weighted quick-union demo

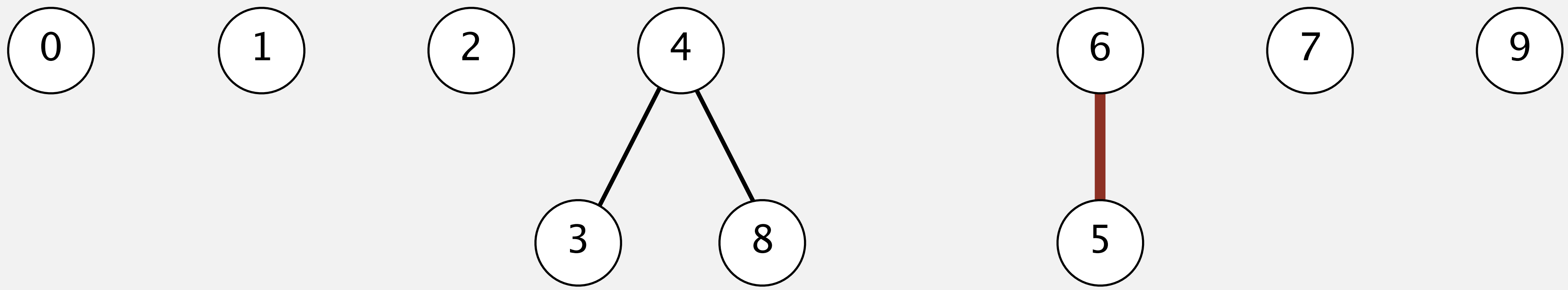
union(6, 5)



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

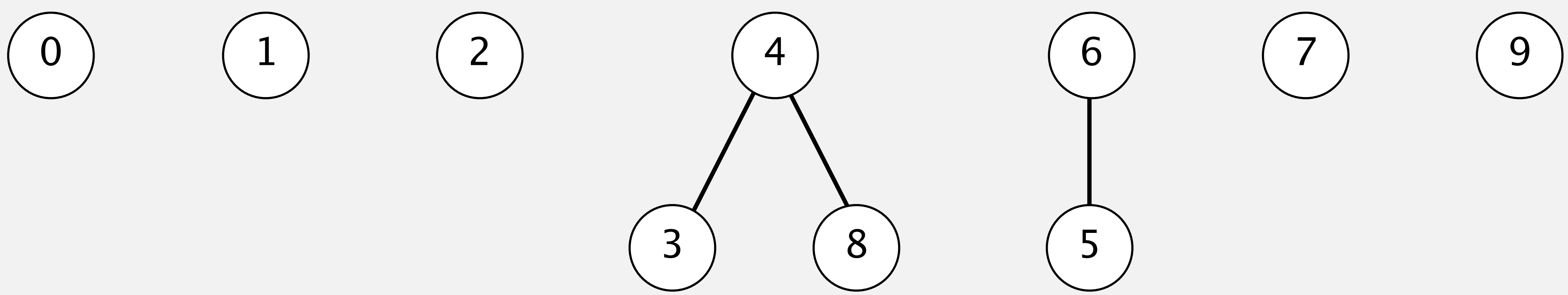
Weighted quick-union demo

union(6, 5)



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

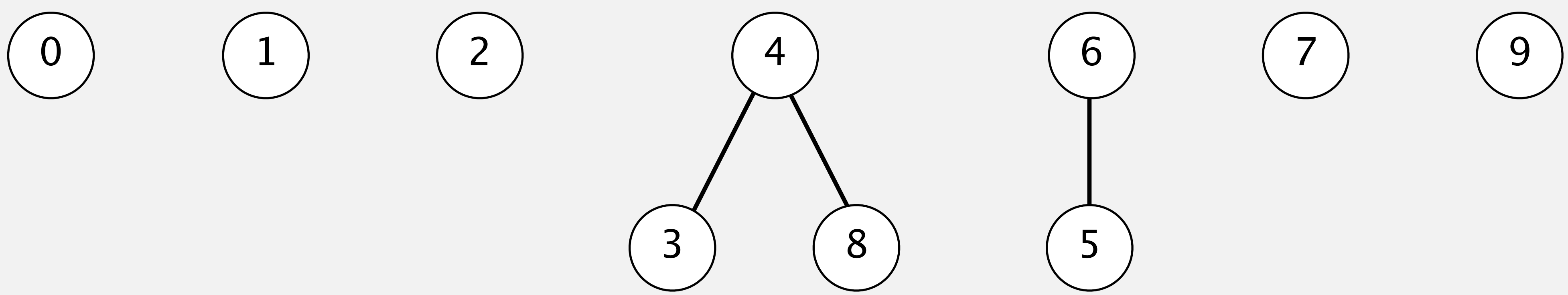
Weighted quick-union demo



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

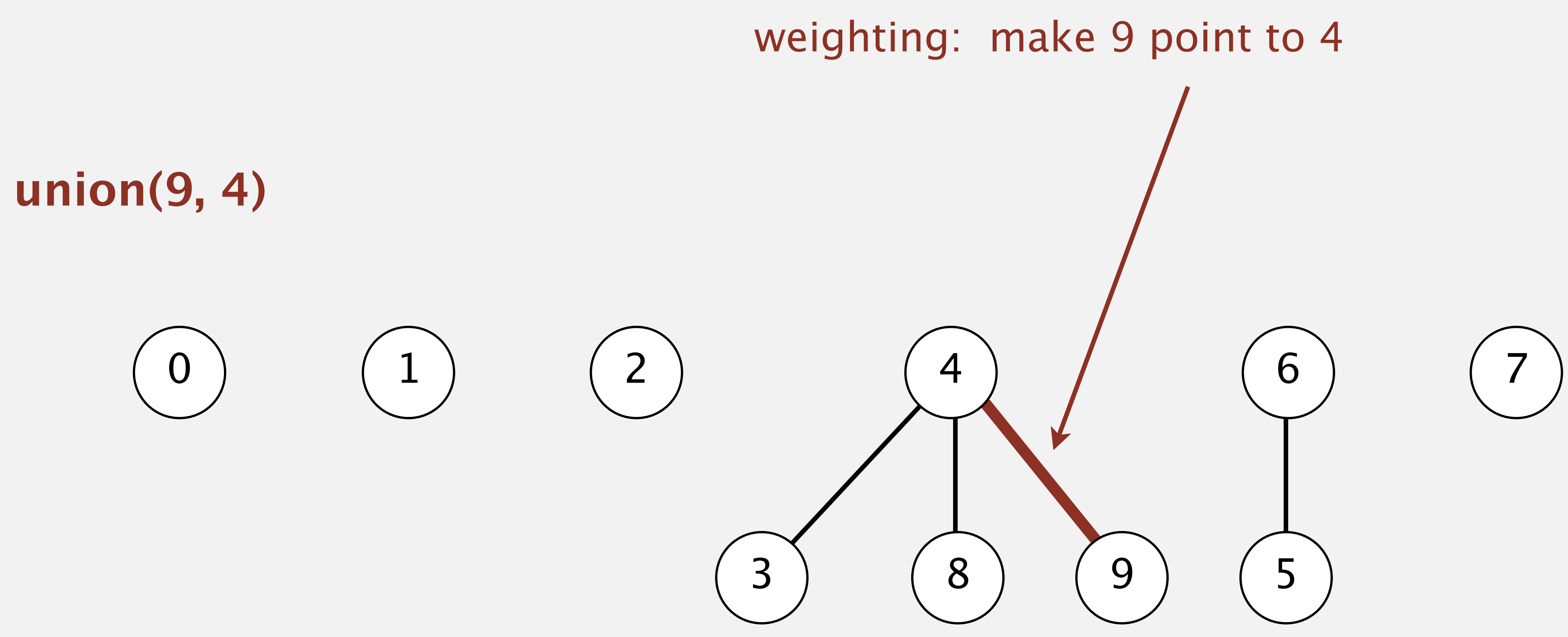
Weighted quick-union demo

union(9, 4)



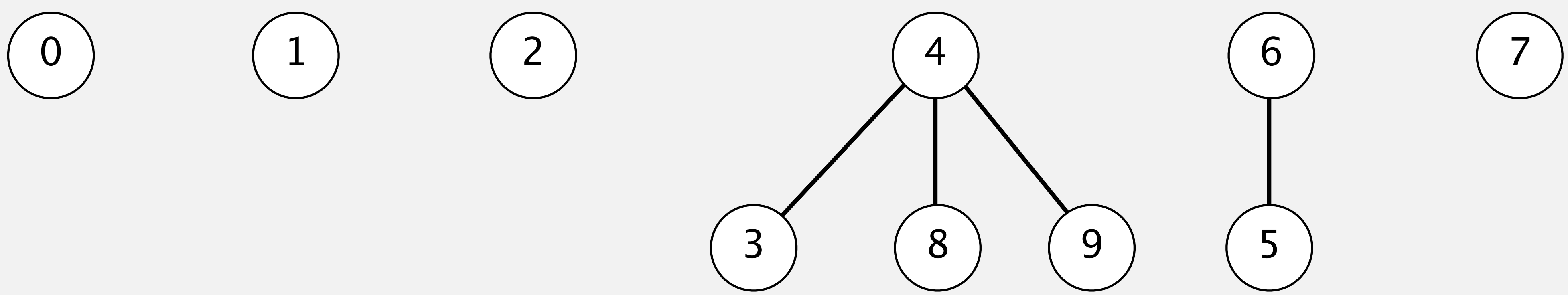
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	4	4	6	6	7	4	9

Weighted quick-union demo



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

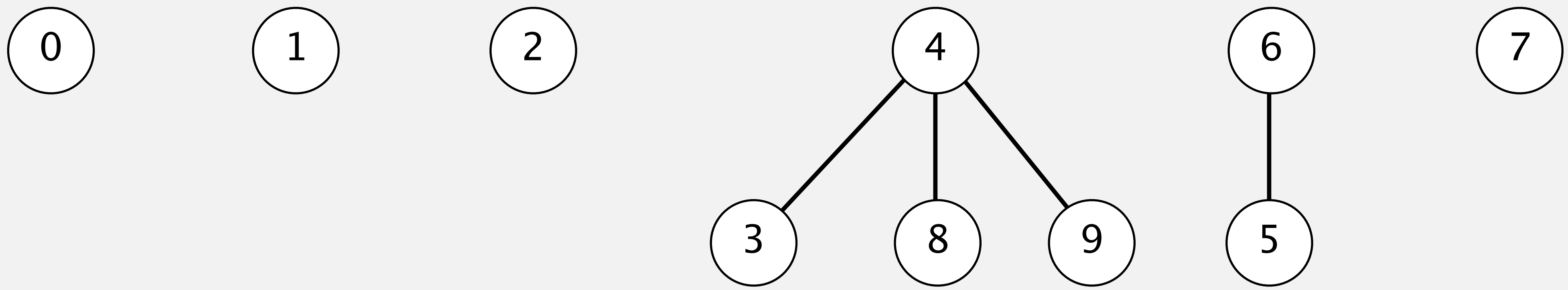
Weighted quick-union demo



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

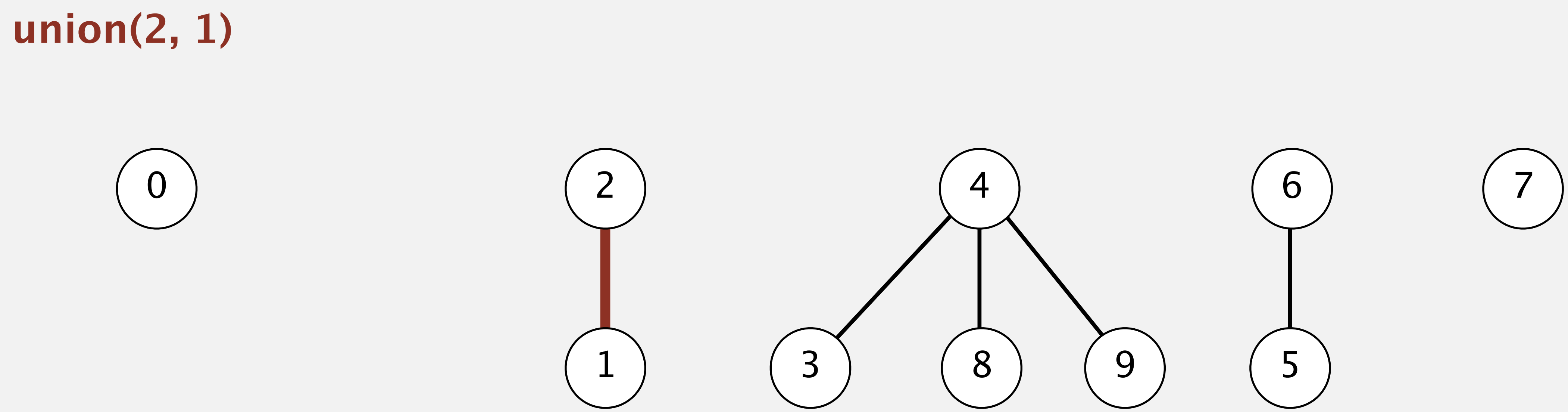
Weighted quick-union demo

union(2, 1)



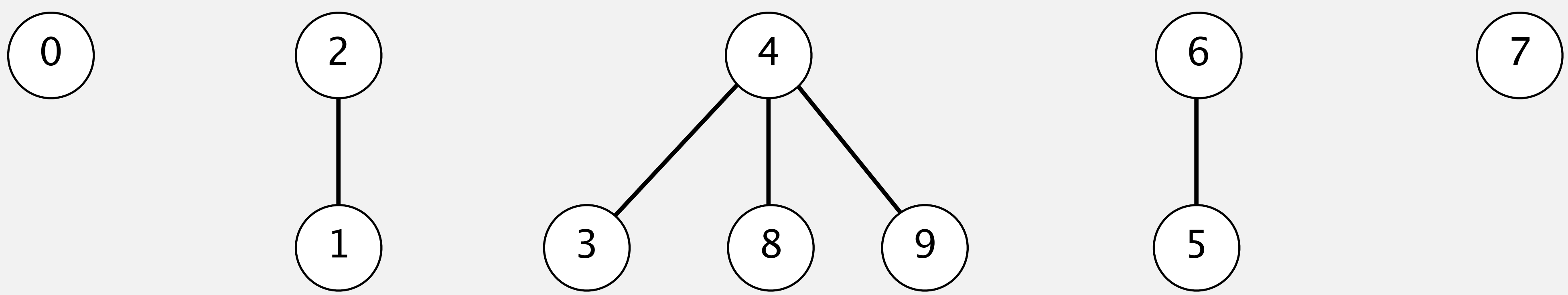
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	4	4	6	6	7	4	4

Weighted quick-union demo



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

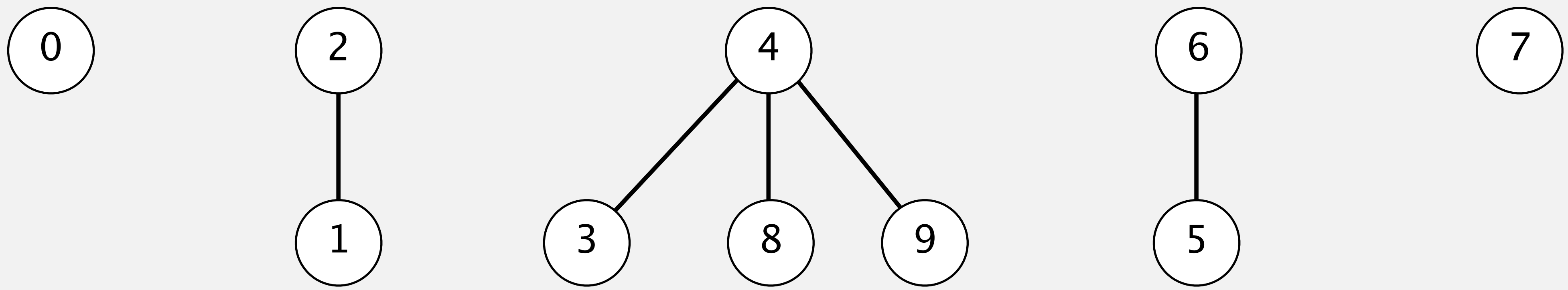
Weighted quick-union demo



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

Weighted quick-union demo

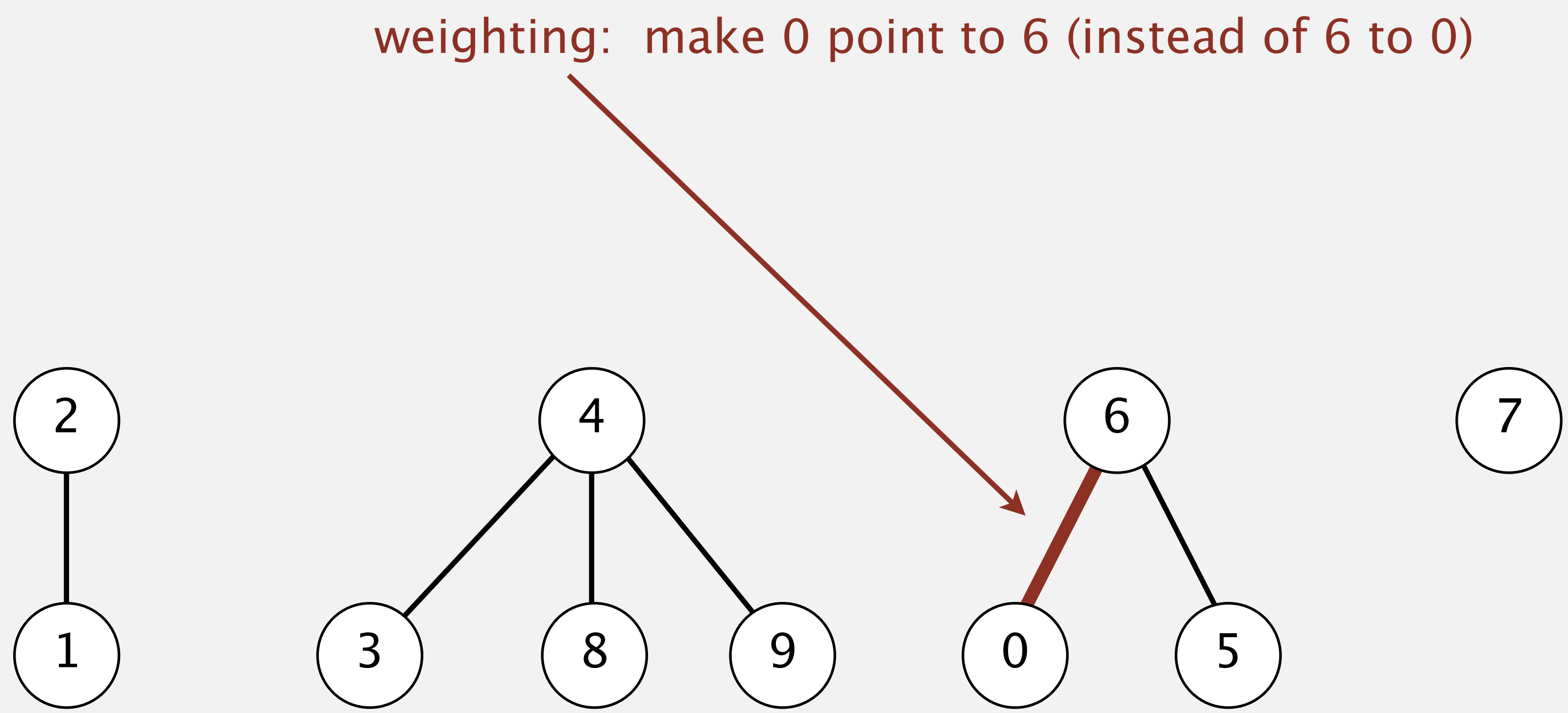
union(5, 0)



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

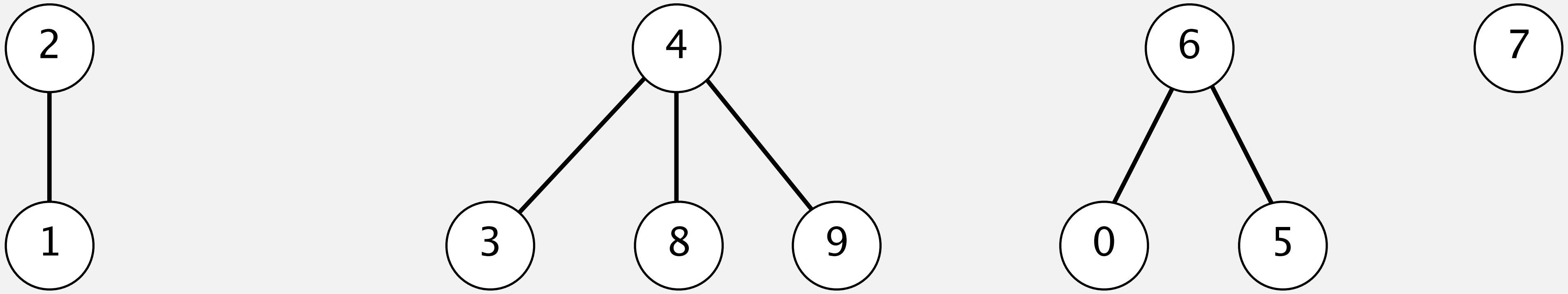
Weighted quick-union demo

union(5, 0)



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

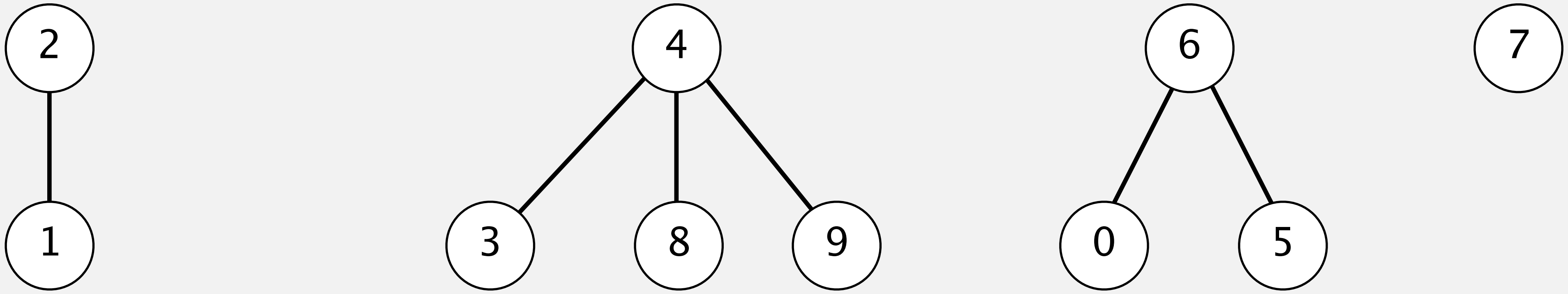
Weighted quick-union demo



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

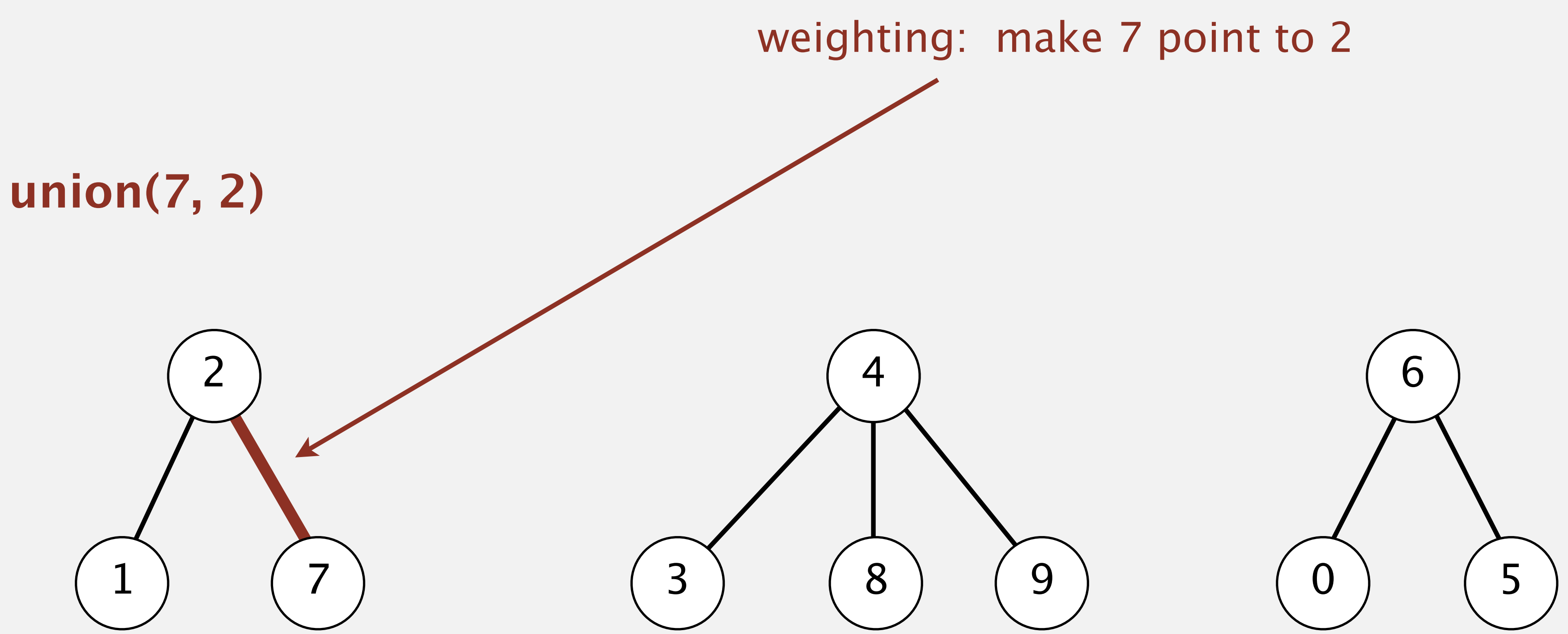
Weighted quick-union demo

union(7, 2)



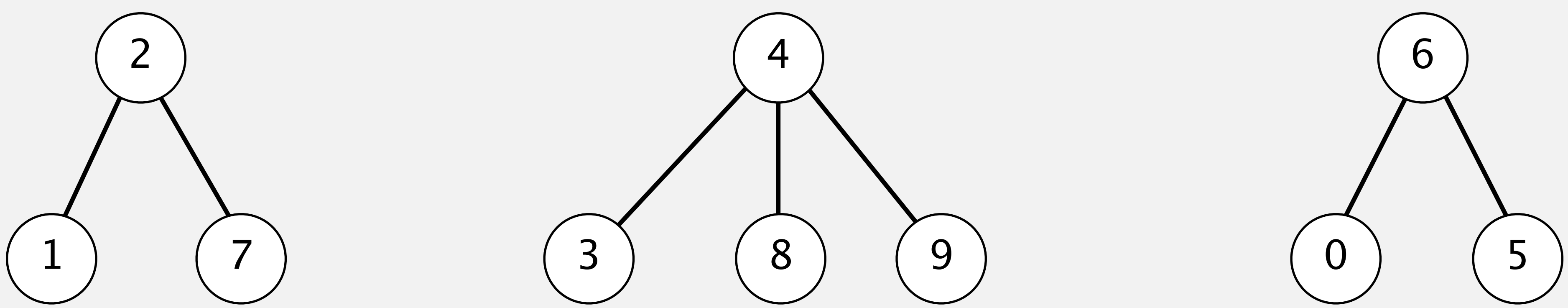
	0	1	2	3	4	5	6	7	8	9
id[]	6	2	2	4	4	6	6	7	4	4

Weighted quick-union demo



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

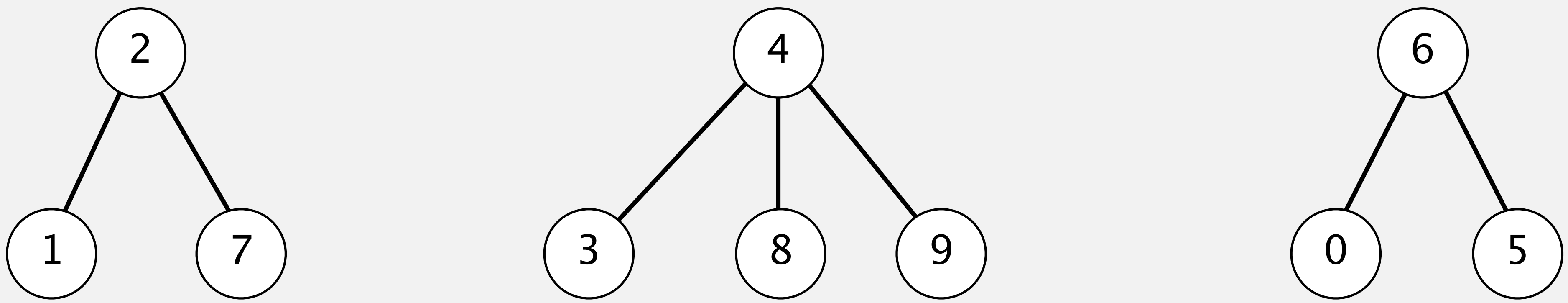
Weighted quick-union demo



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

Weighted quick-union demo

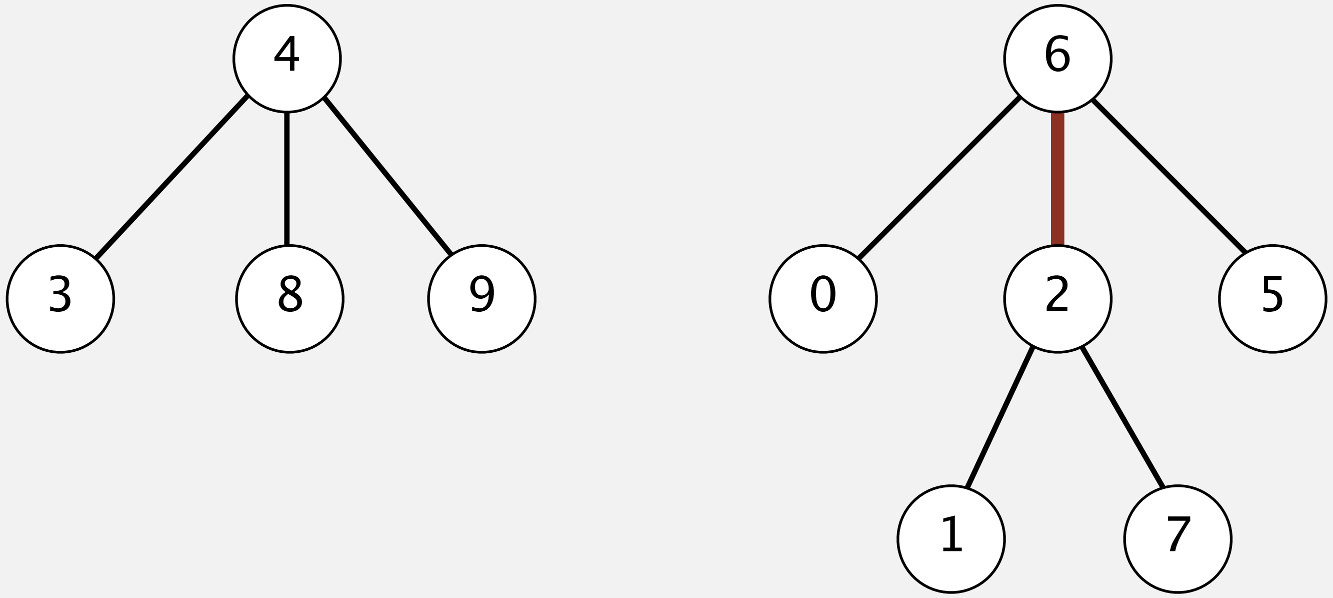
union(6, 1)



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

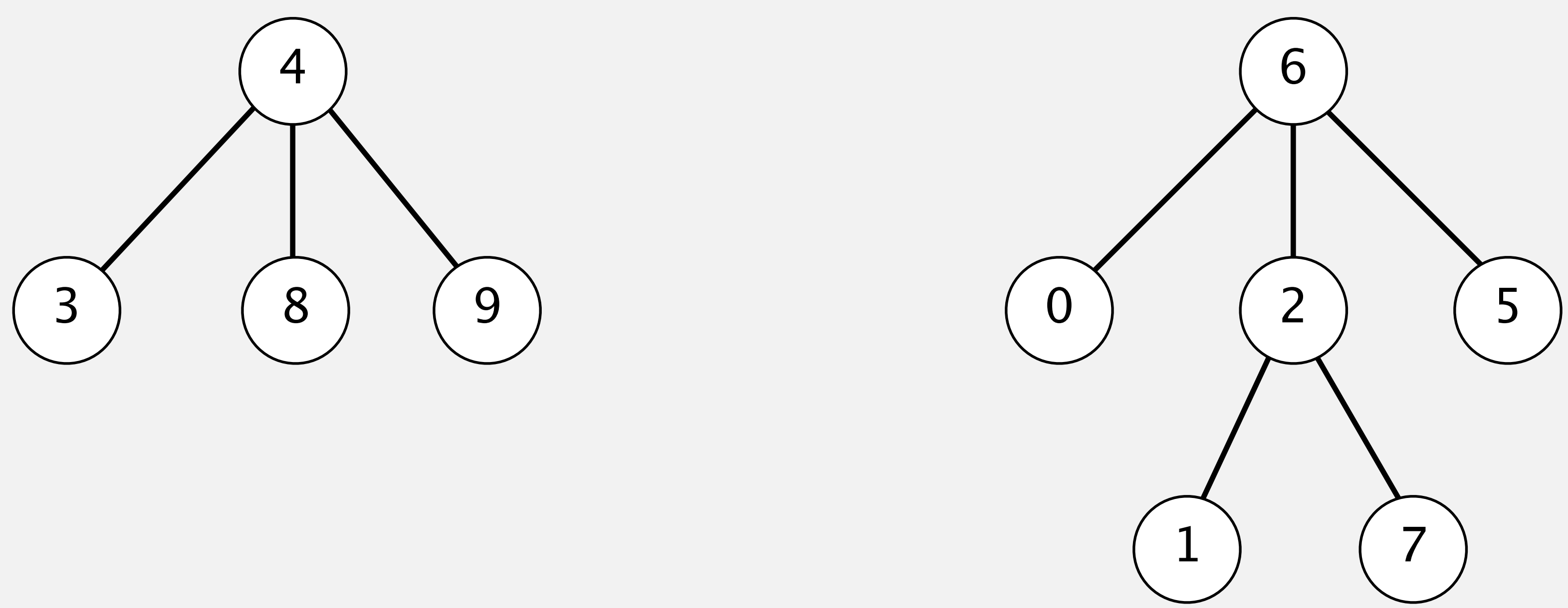
Weighted quick-union demo

union(6, 1)



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

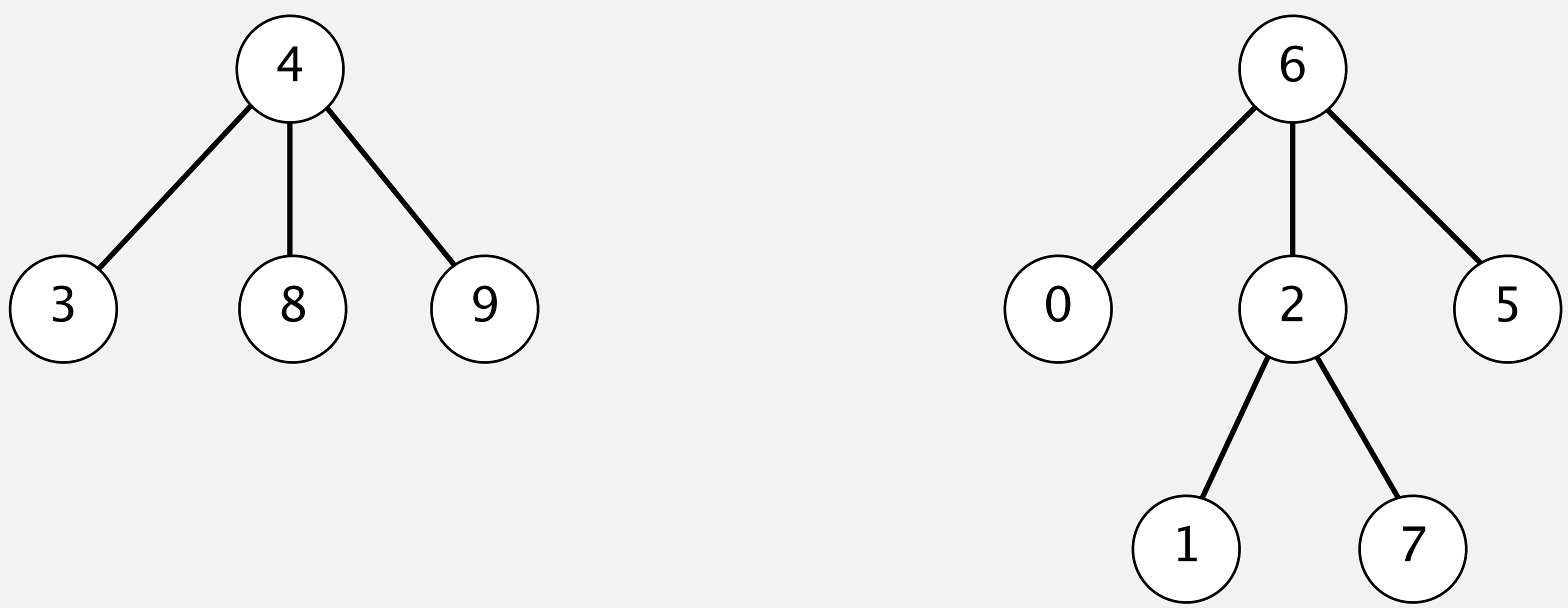
Weighted quick-union demo



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

Weighted quick-union demo

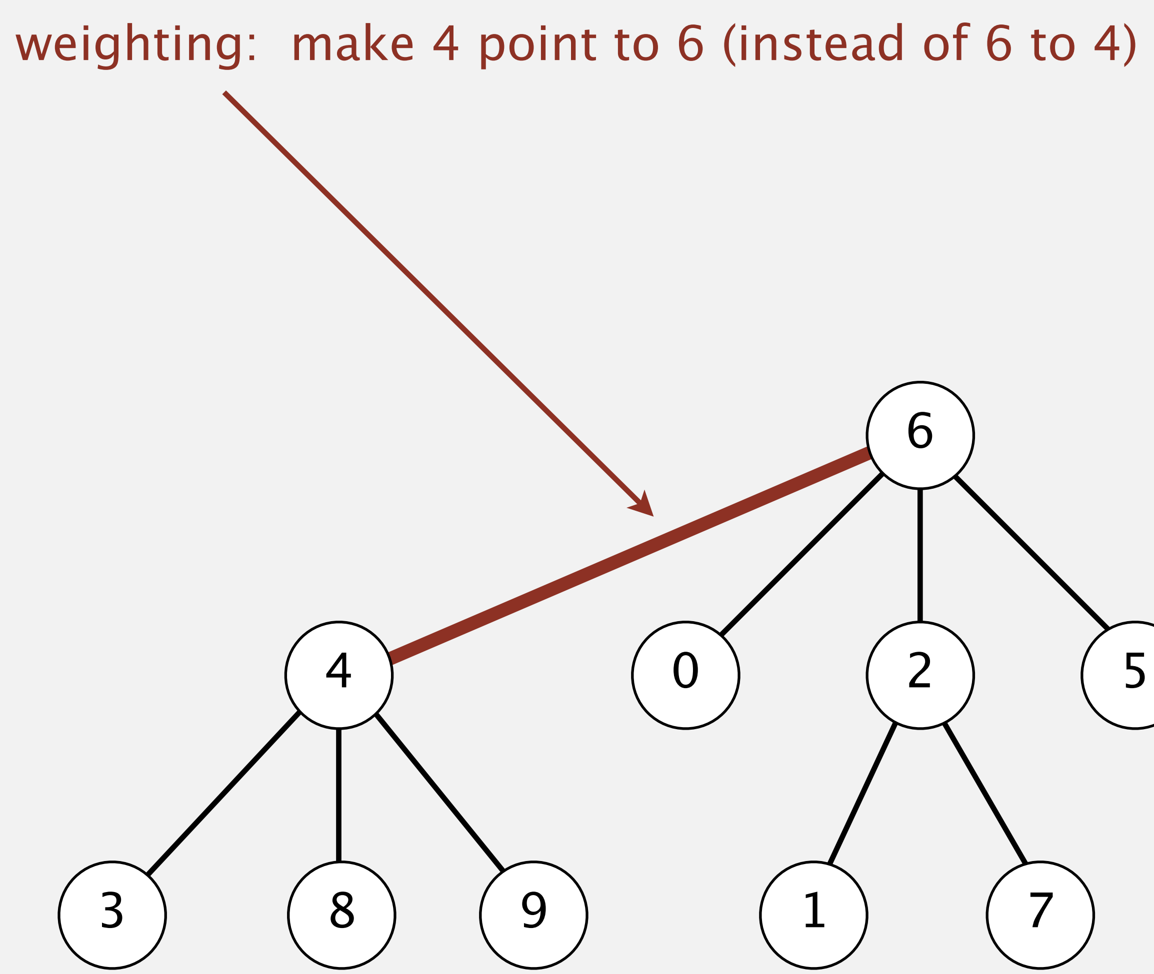
union(7, 3)



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

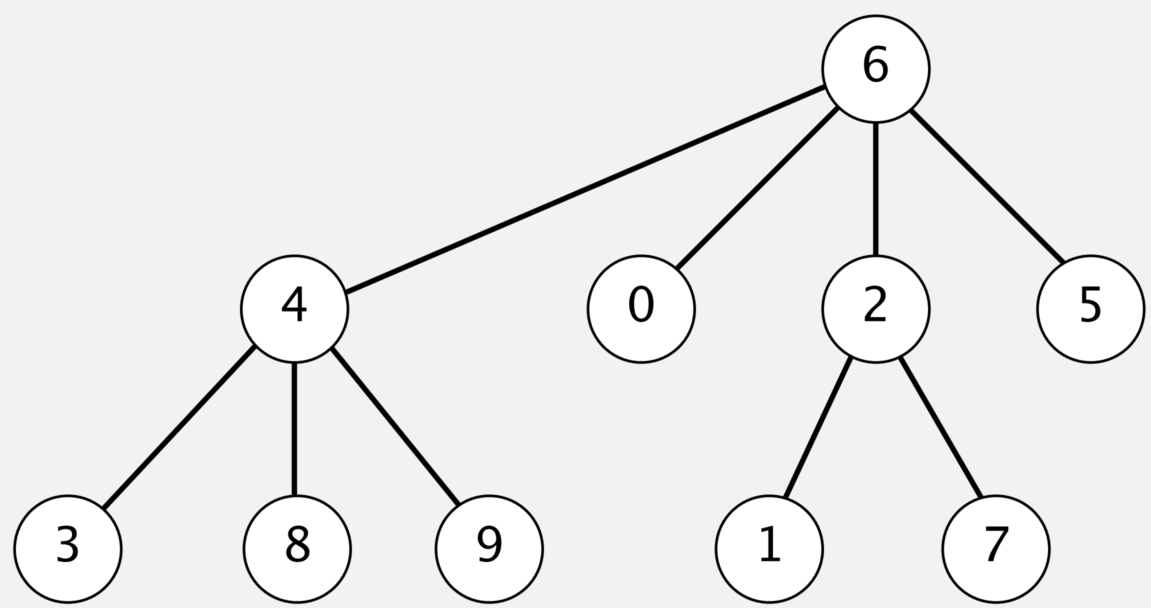
Weighted quick-union demo

union(7, 3)



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

Weighted quick-union demo



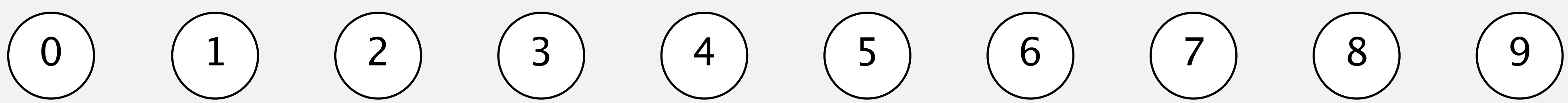
	0	1	2	3	4	5	6	7	8	9
id[]	6	2	6	4	6	6	6	2	4	4

http://algs4.cs.princeton.edu

1.5 PATH COMPRESSION DEMO

click to begin demo

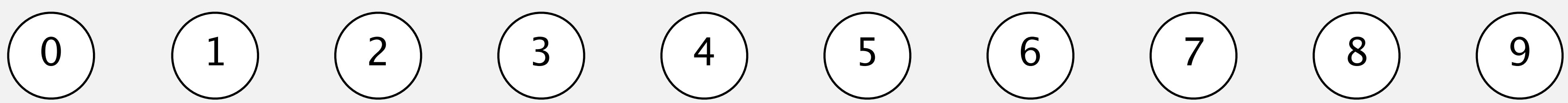
Weighted quick-union with path compression demo



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

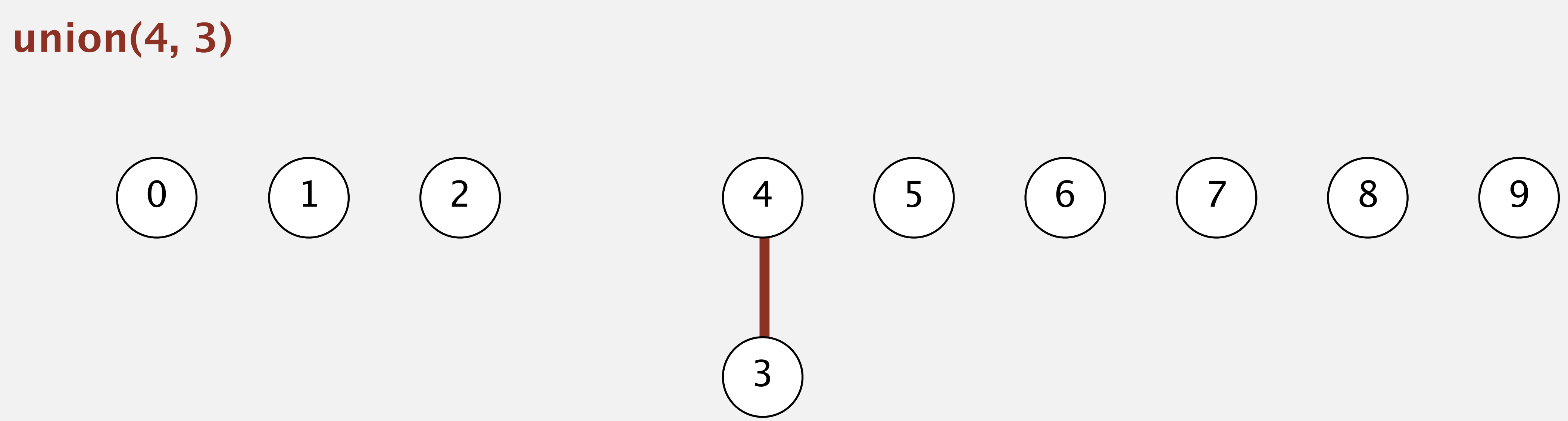
Weighted quick-union with path compression demo

union(4, 3)



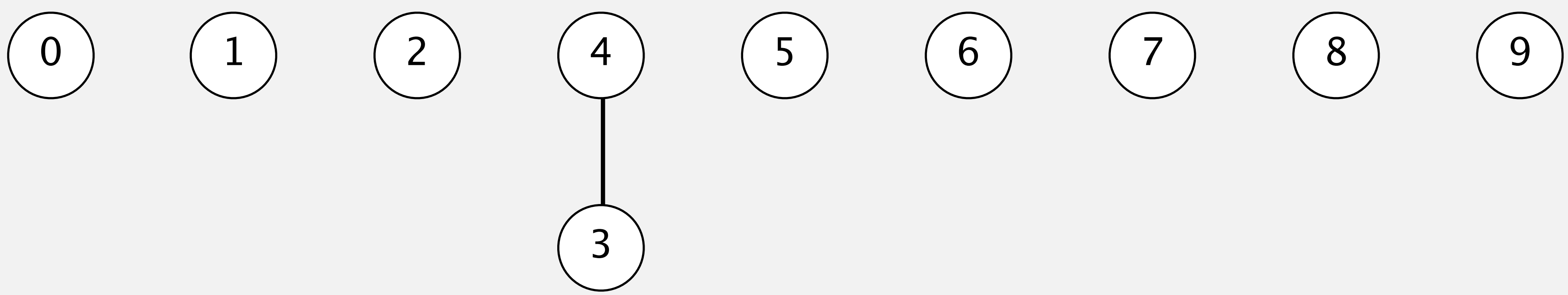
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Weighted quick-union with path compression demo



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

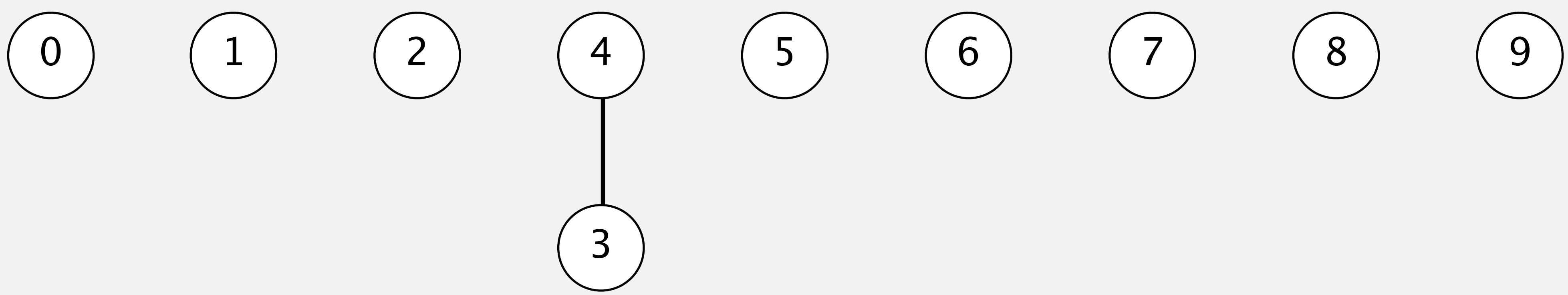
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

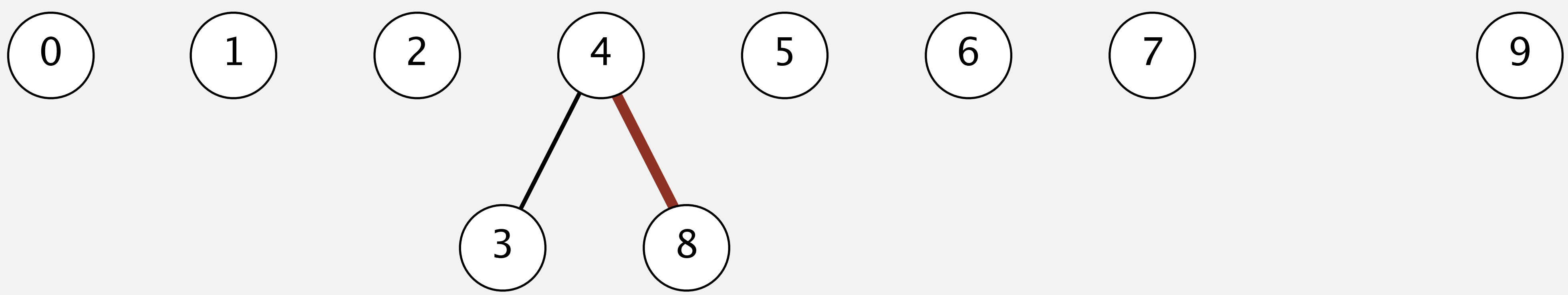
union(3, 8)



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

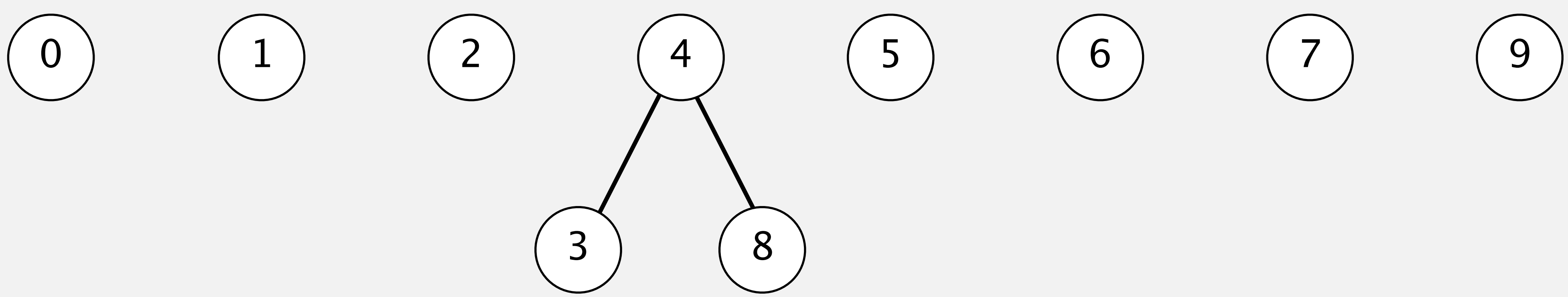
Weighted quick-union with path compression demo

union(3, 8)



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

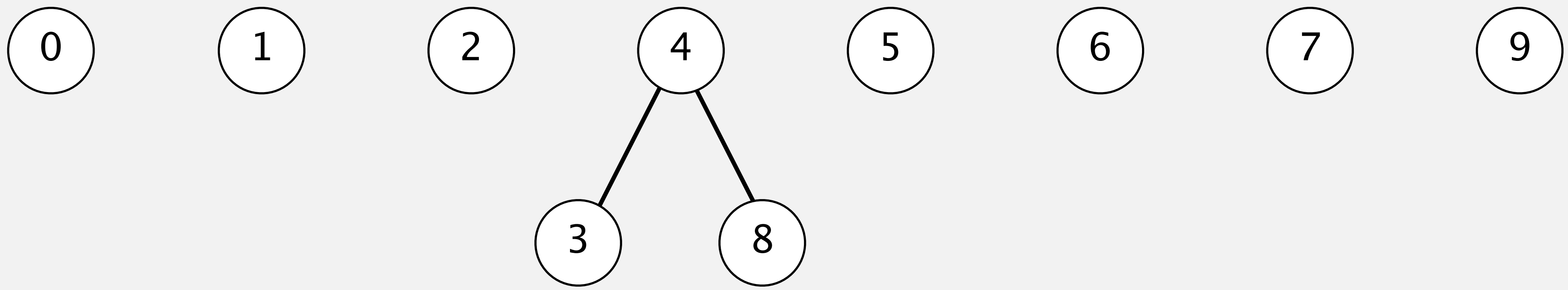
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

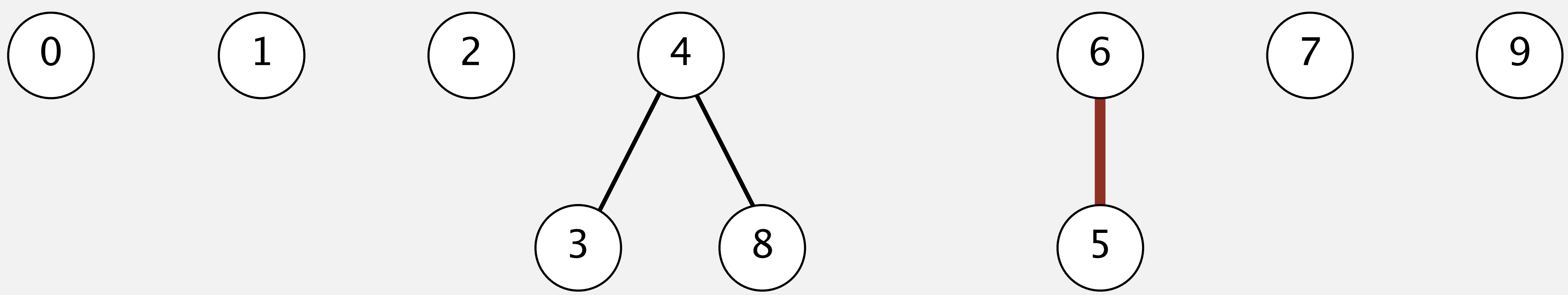
union(6, 5)



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

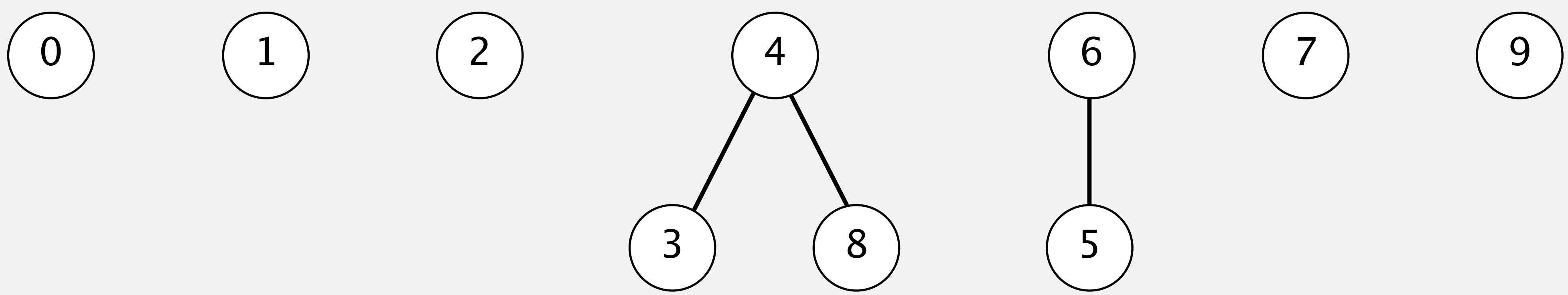
Weighted quick-union with path compression demo

union(6, 5)



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

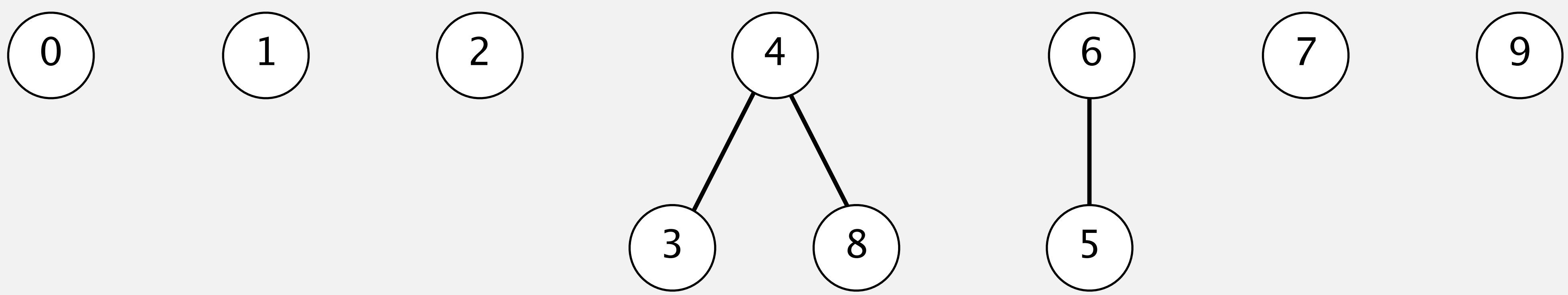
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

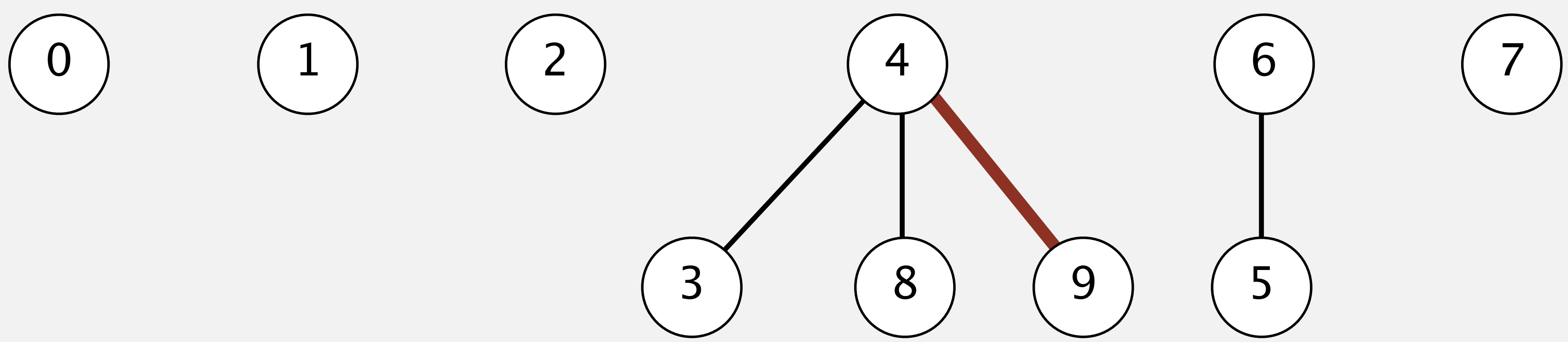
union(9, 4)



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

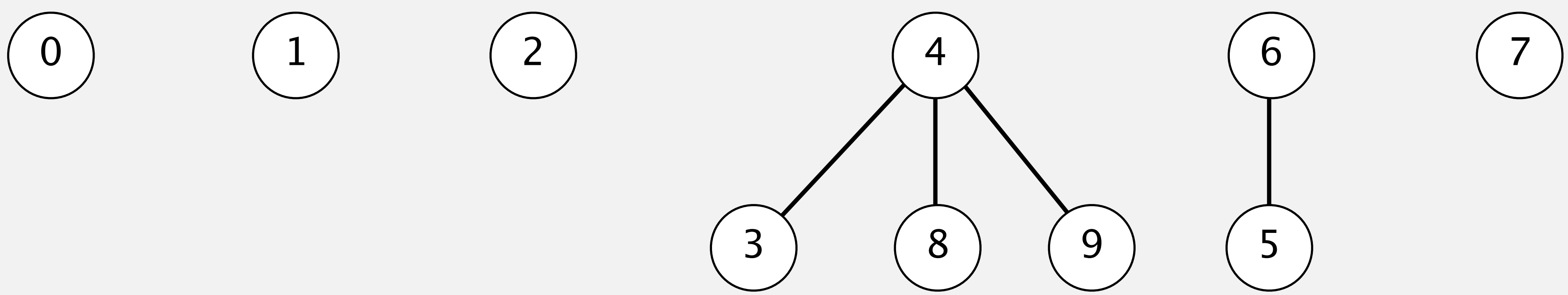
Weighted quick-union with path compression demo

union(9, 4)



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

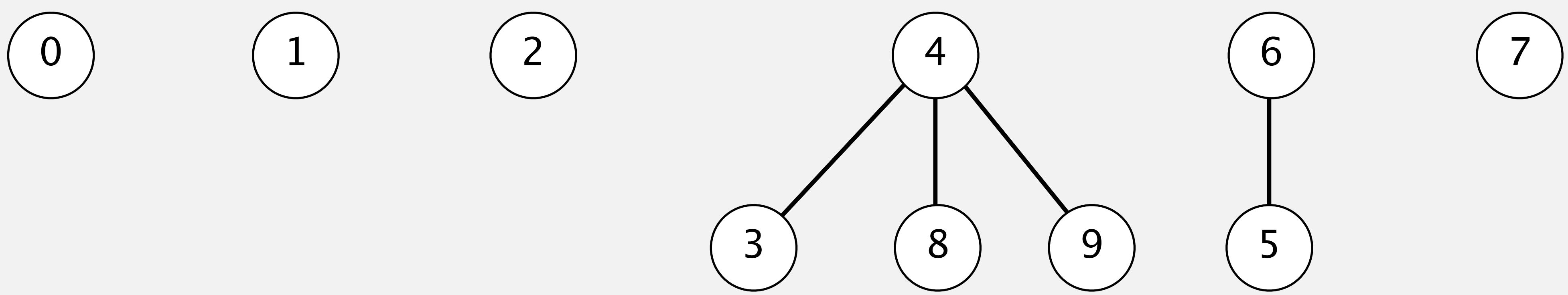
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

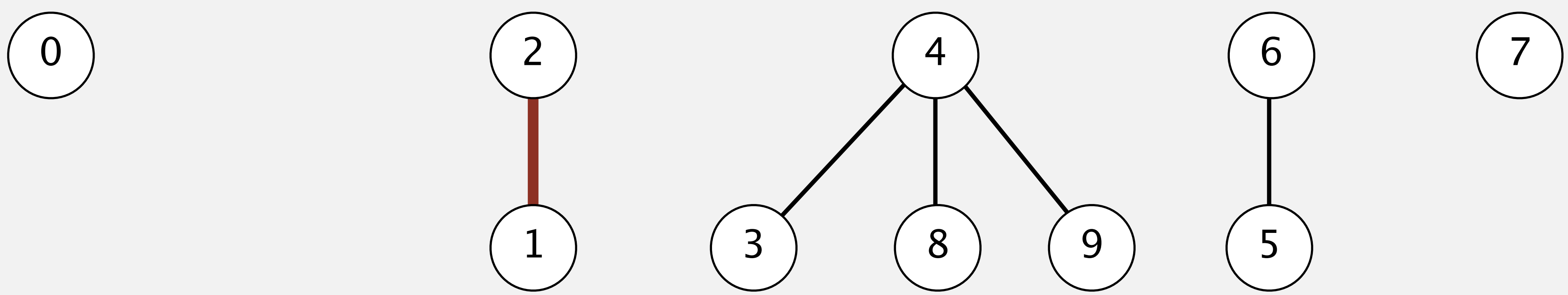
union(2, 1)



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

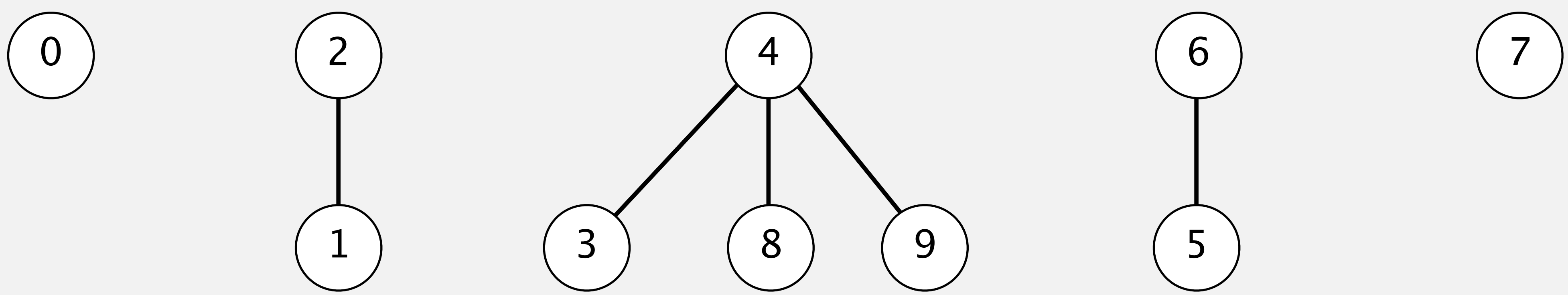
Weighted quick-union with path compression demo

union(2, 1)



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

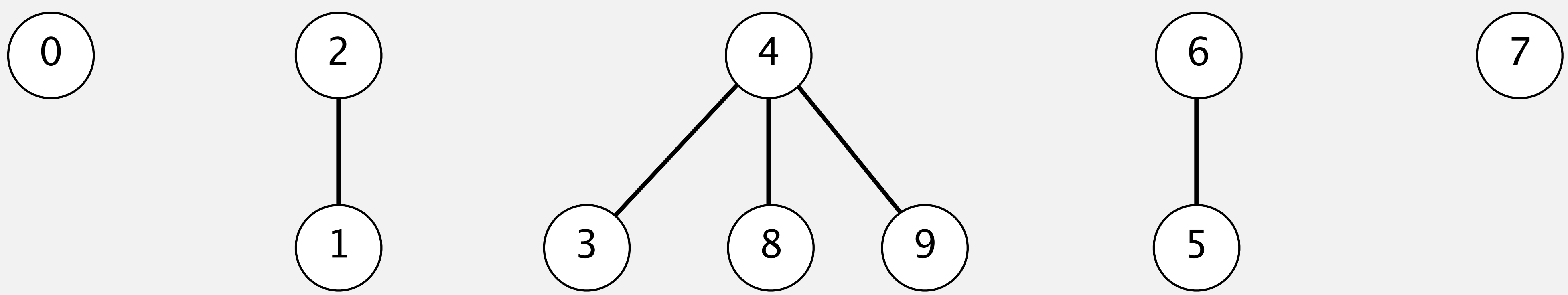
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

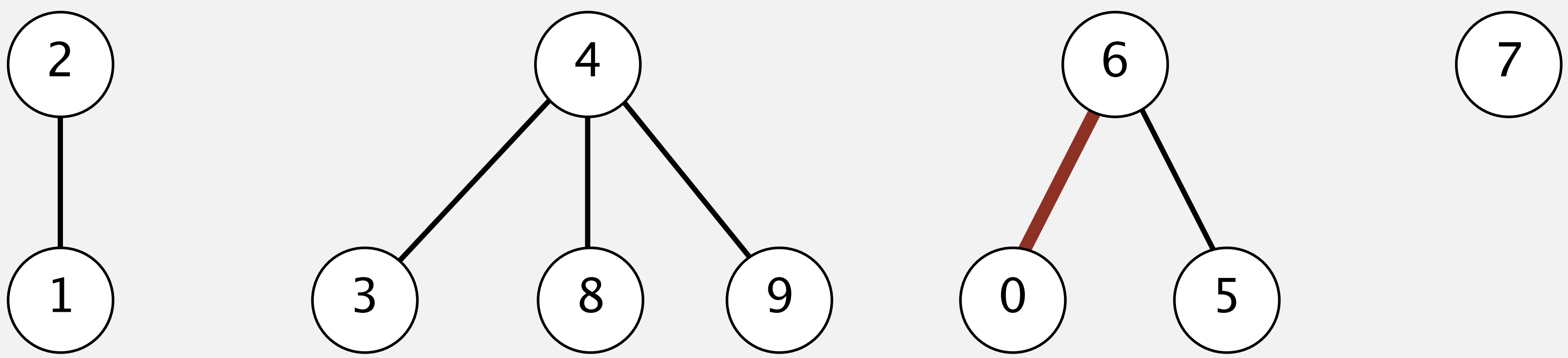
union(5, 0)



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

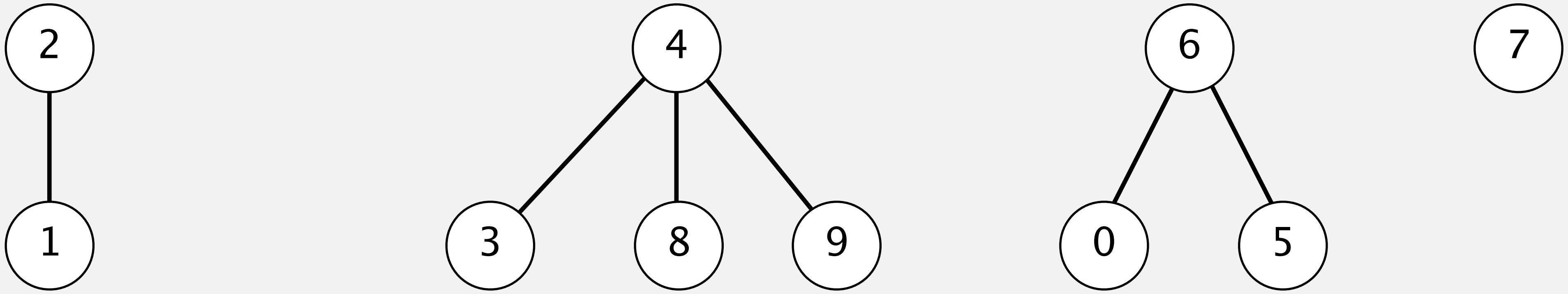
Weighted quick-union with path compression demo

union(5, 0)



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

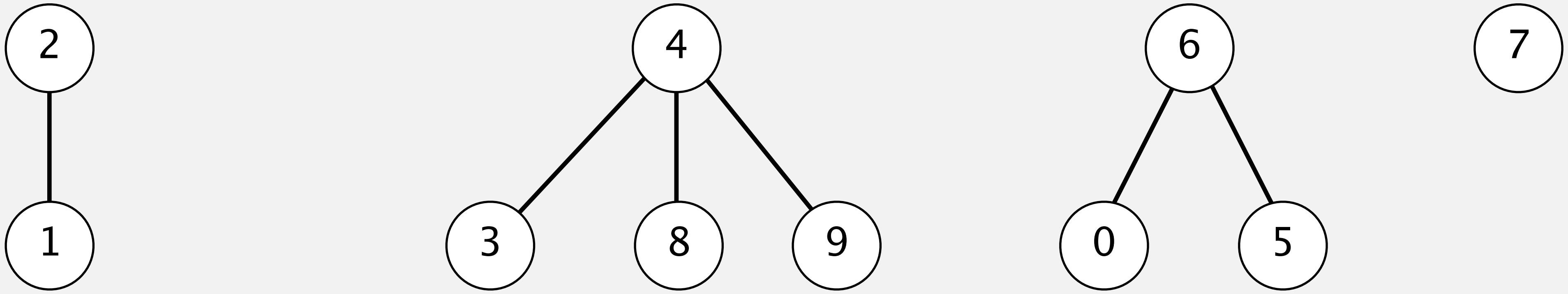
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

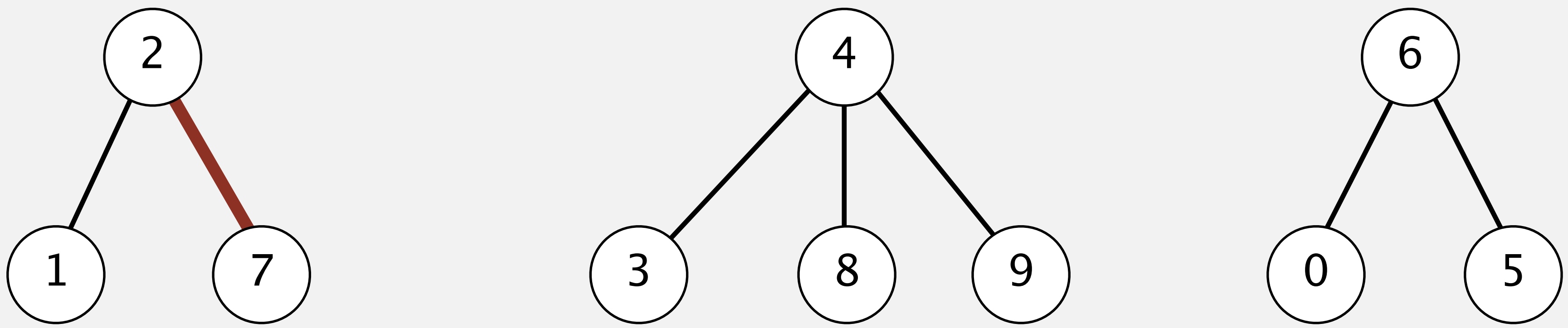
union(7, 2)



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

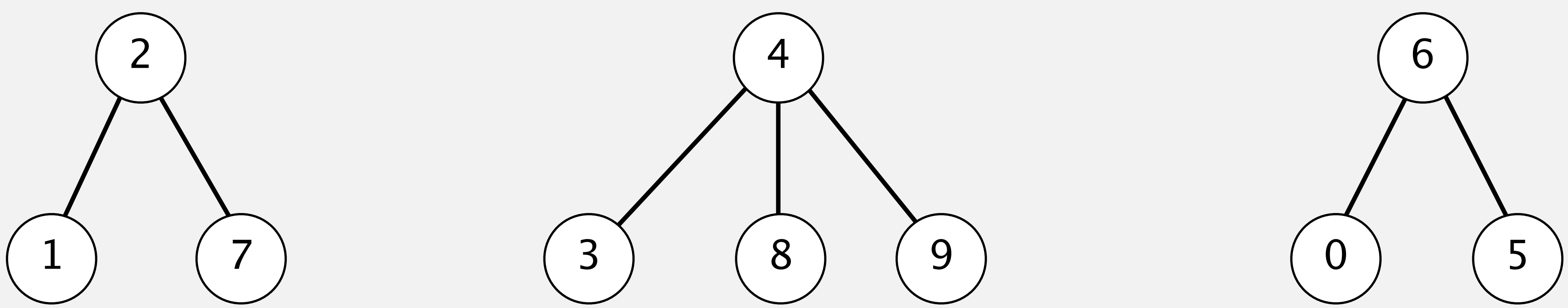
Weighted quick-union with path compression demo

union(7, 2)



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

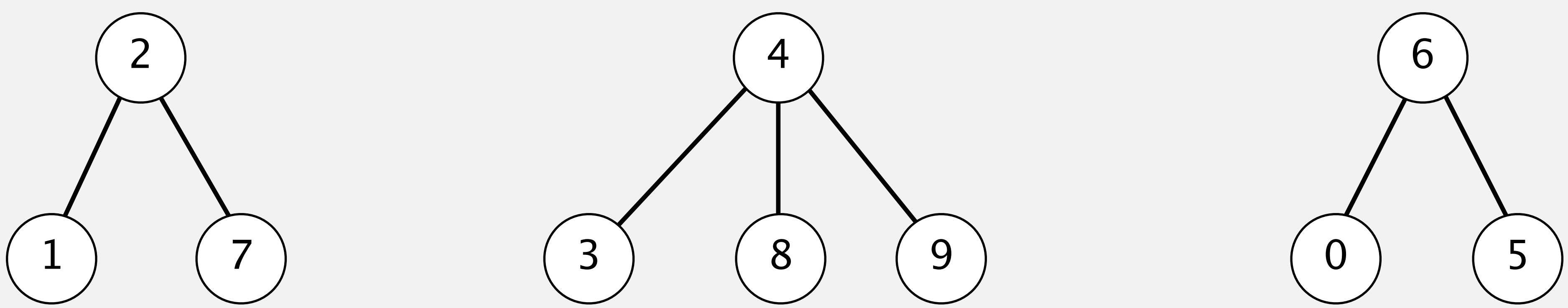
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

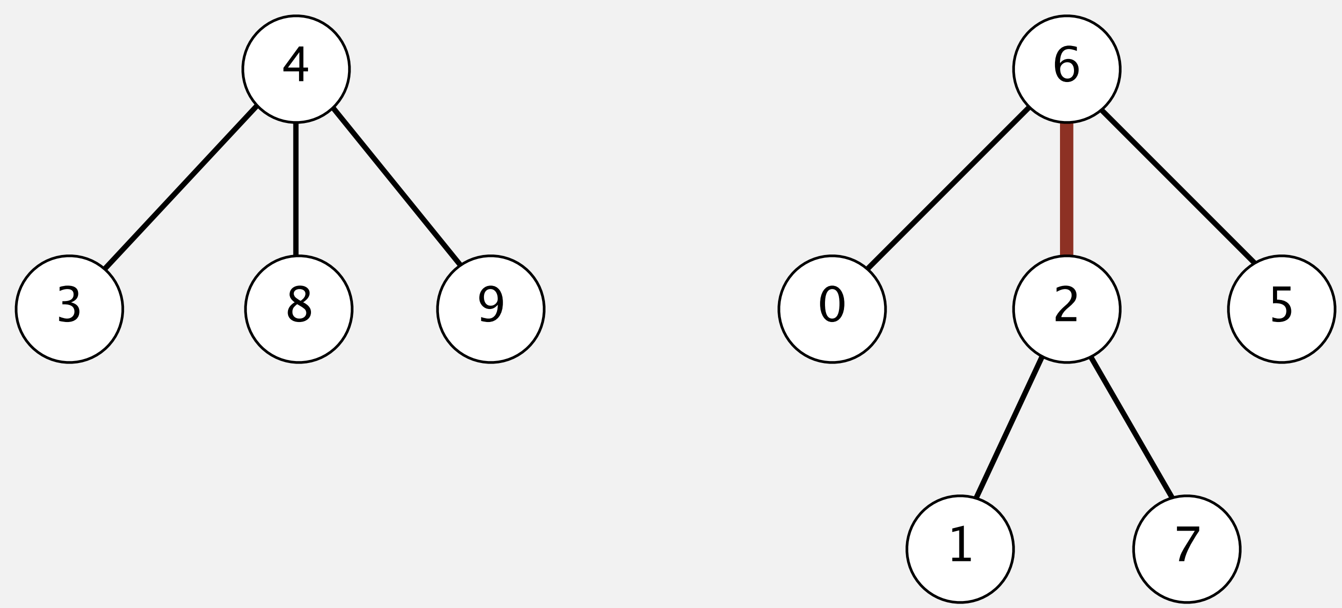
union(6, 1)



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

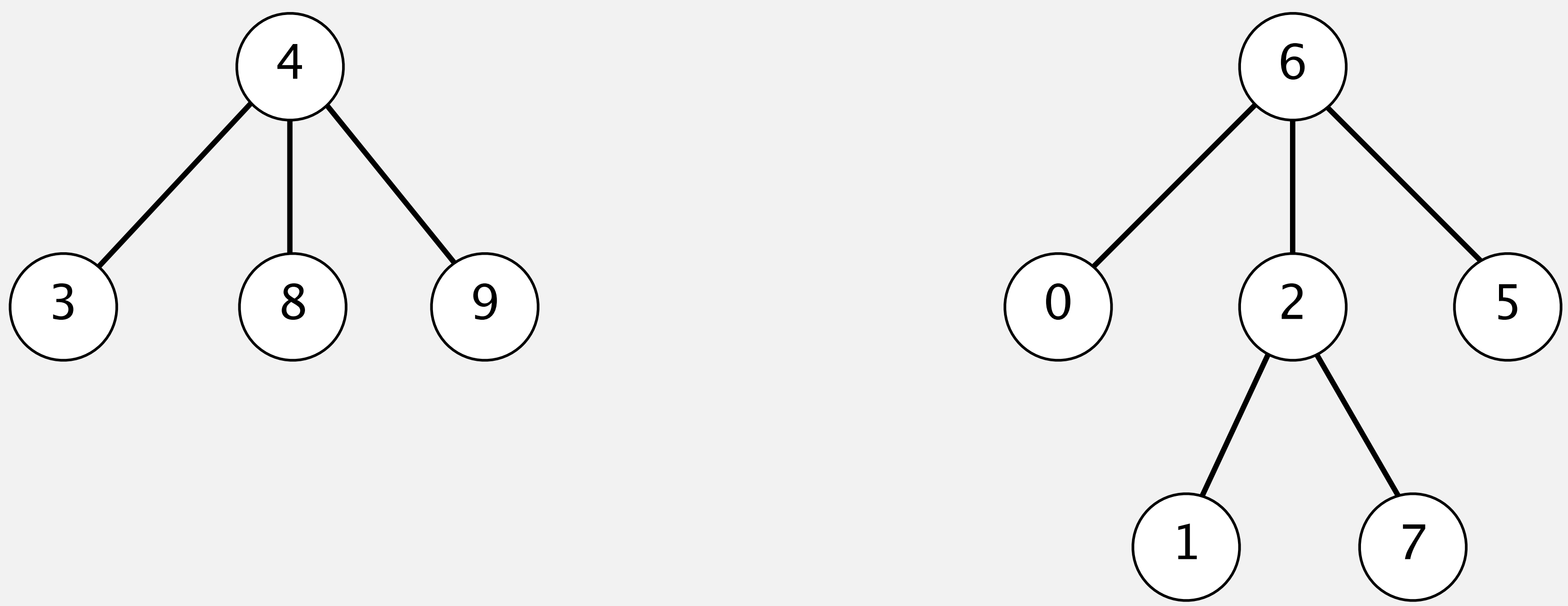
Weighted quick-union with path compression demo

union(6, 1)



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

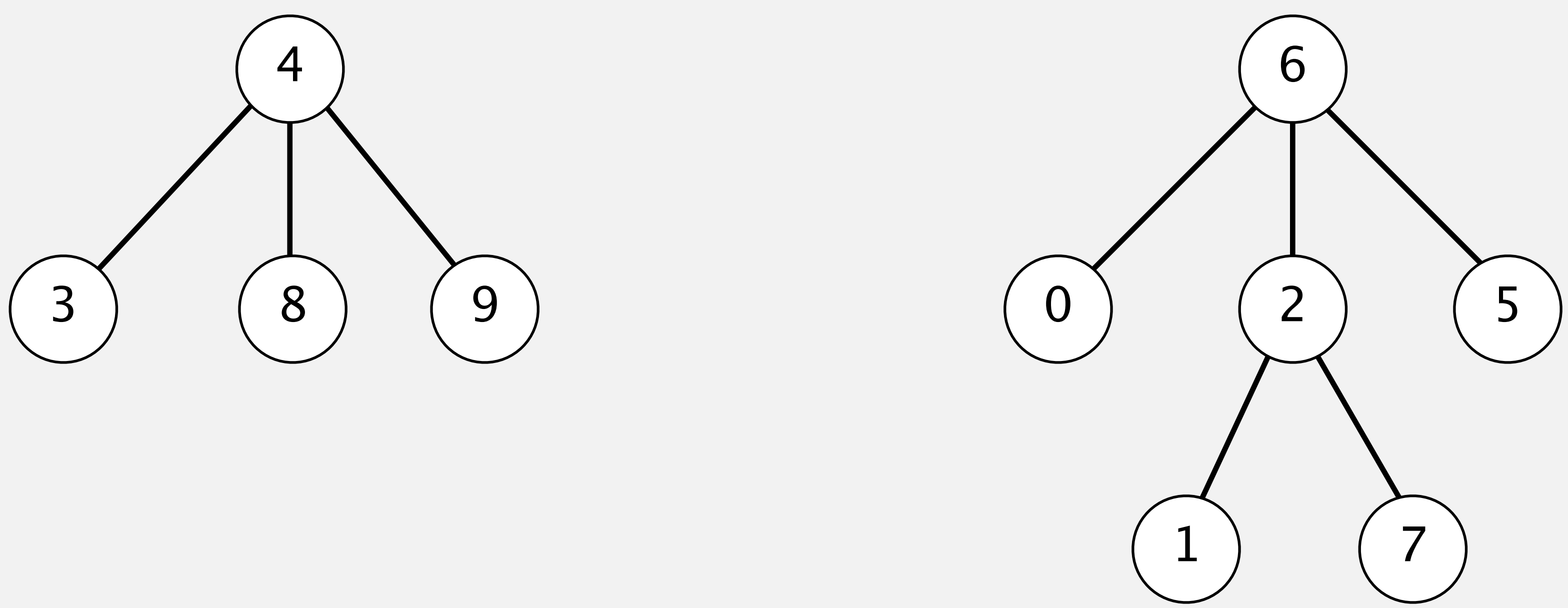
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

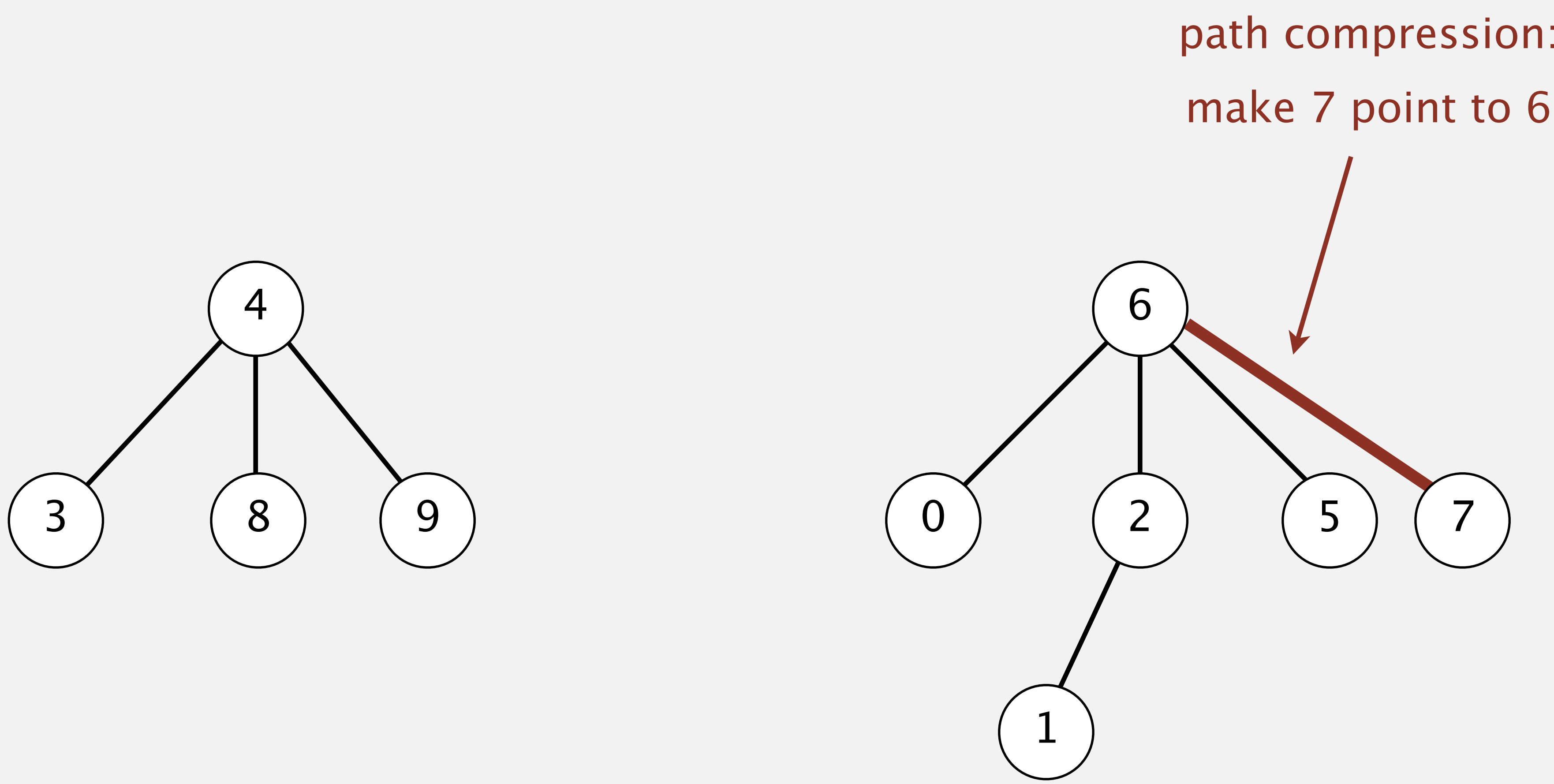
union(7, 3)



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

Weighted quick-union with path compression demo

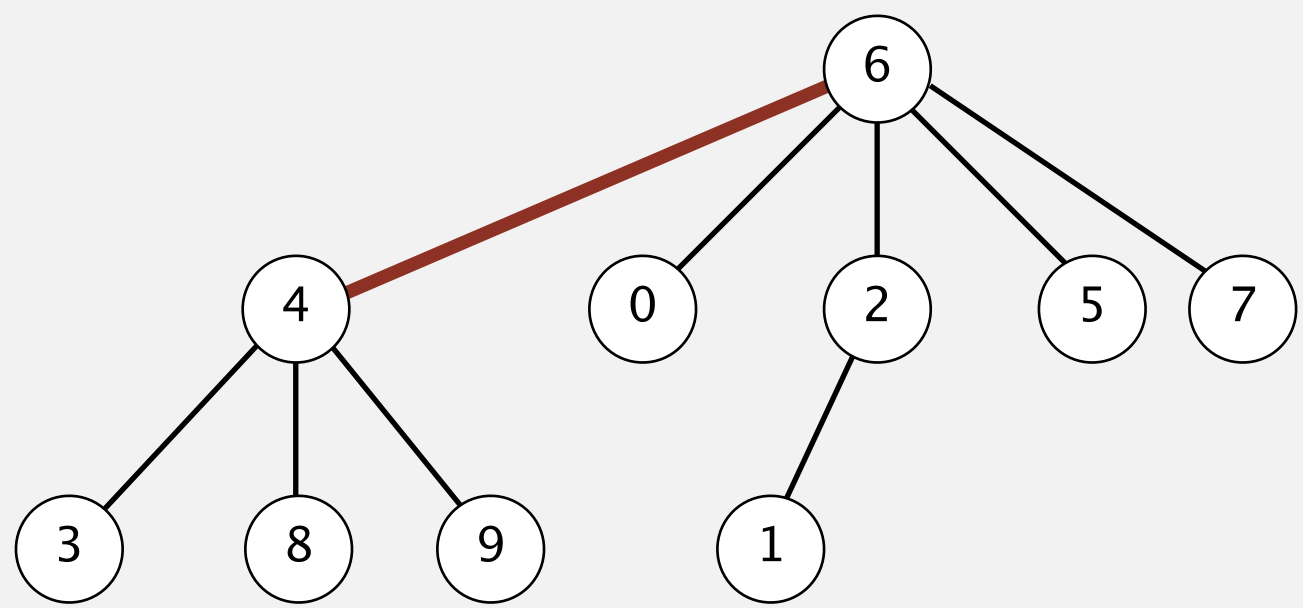
union(7, 3)



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

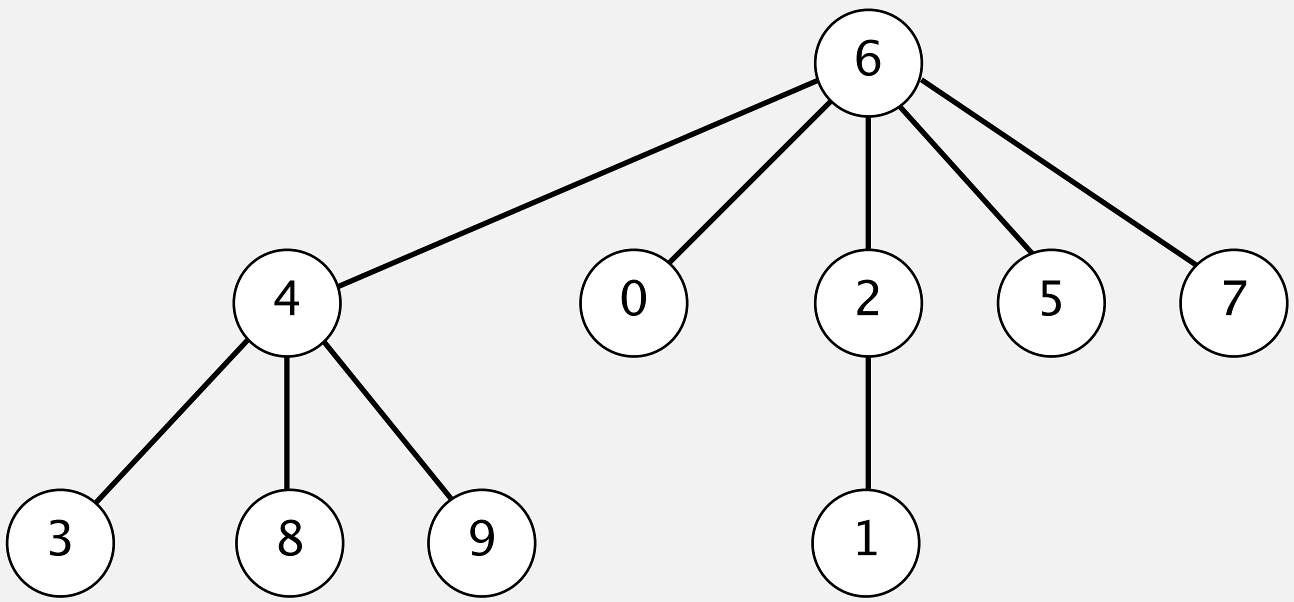
Weighted quick-union with path compression demo

union(7, 3)



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

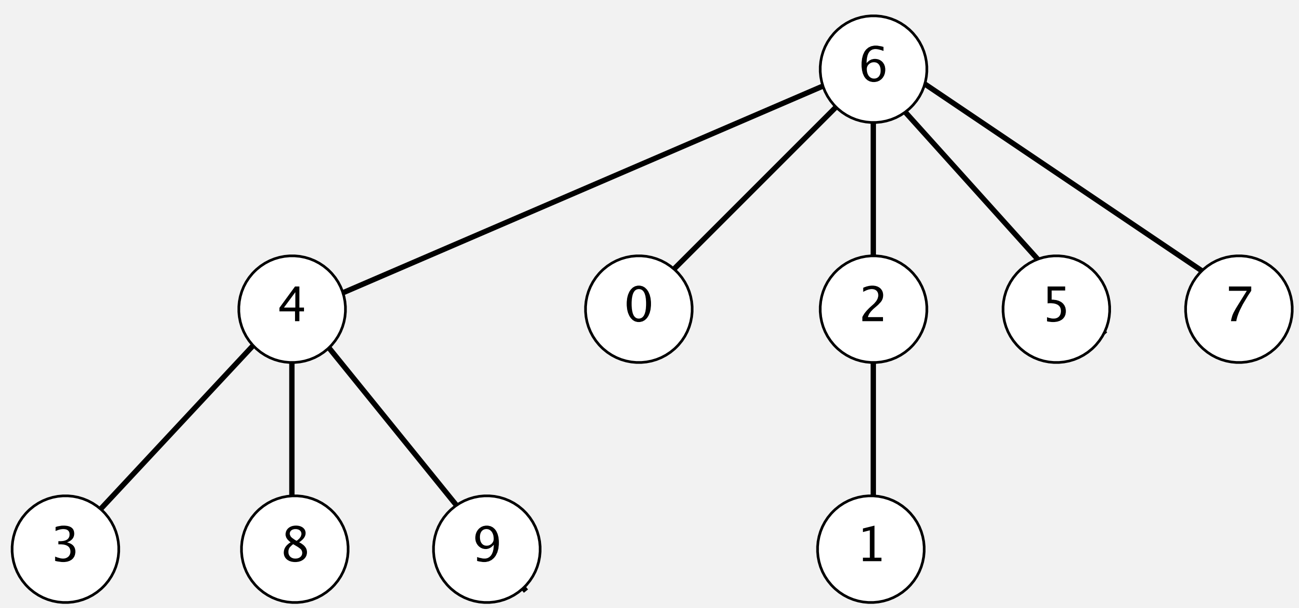
Weighted quick-union with path compression demo



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

Weighted quick-union with path compression demo

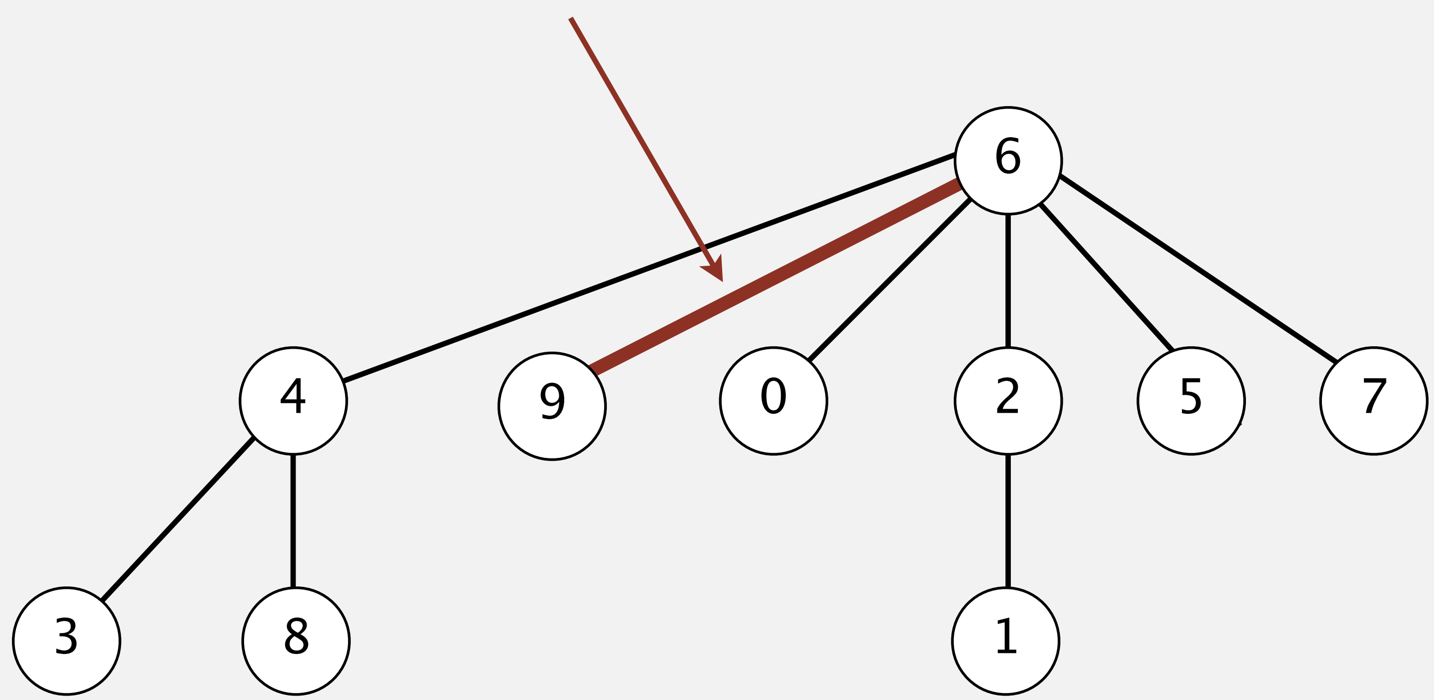
connected(9, 1) ✔



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

Weighted quick-union with path compression demo

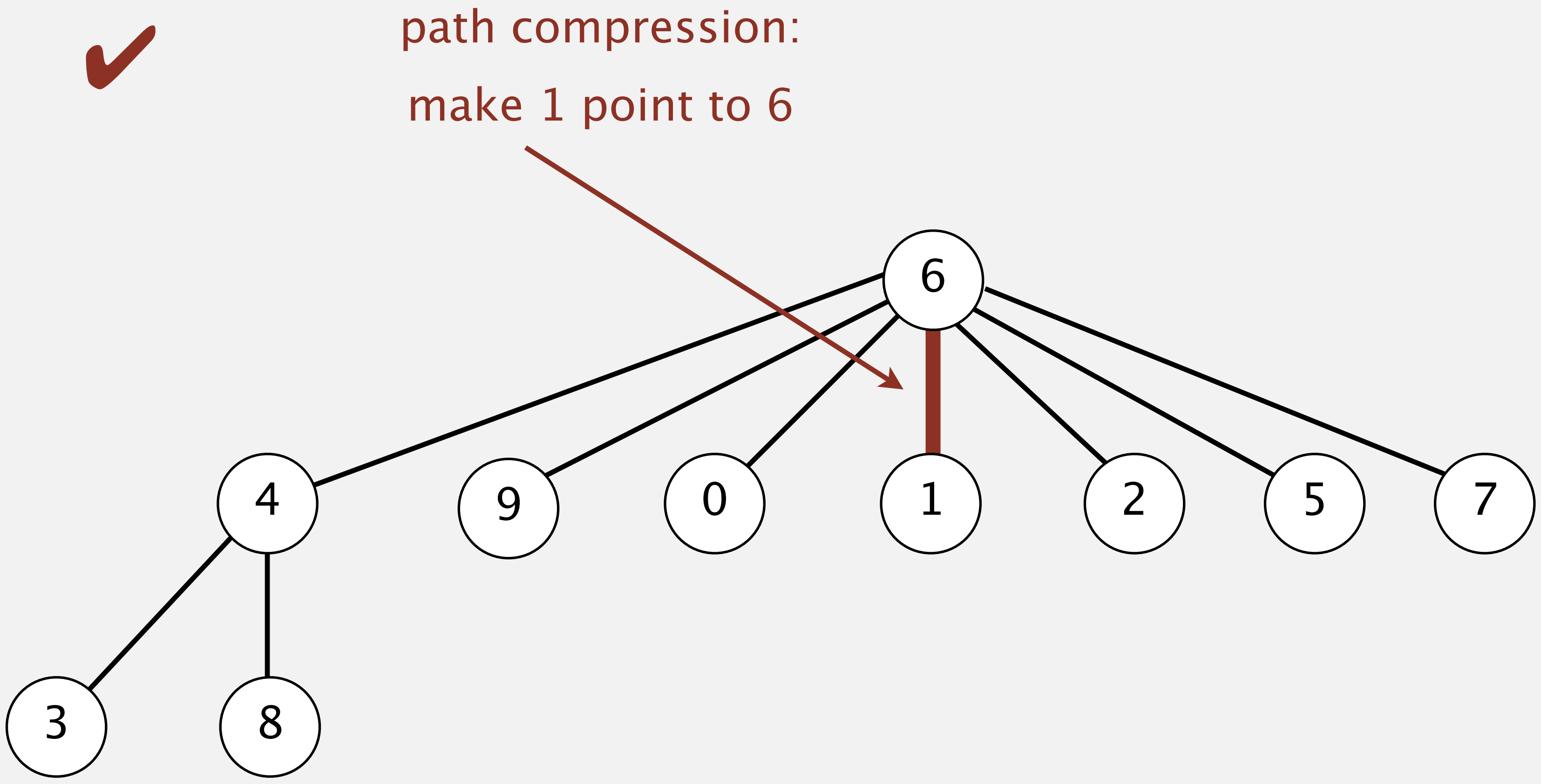
connected(9, 1) ✔



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

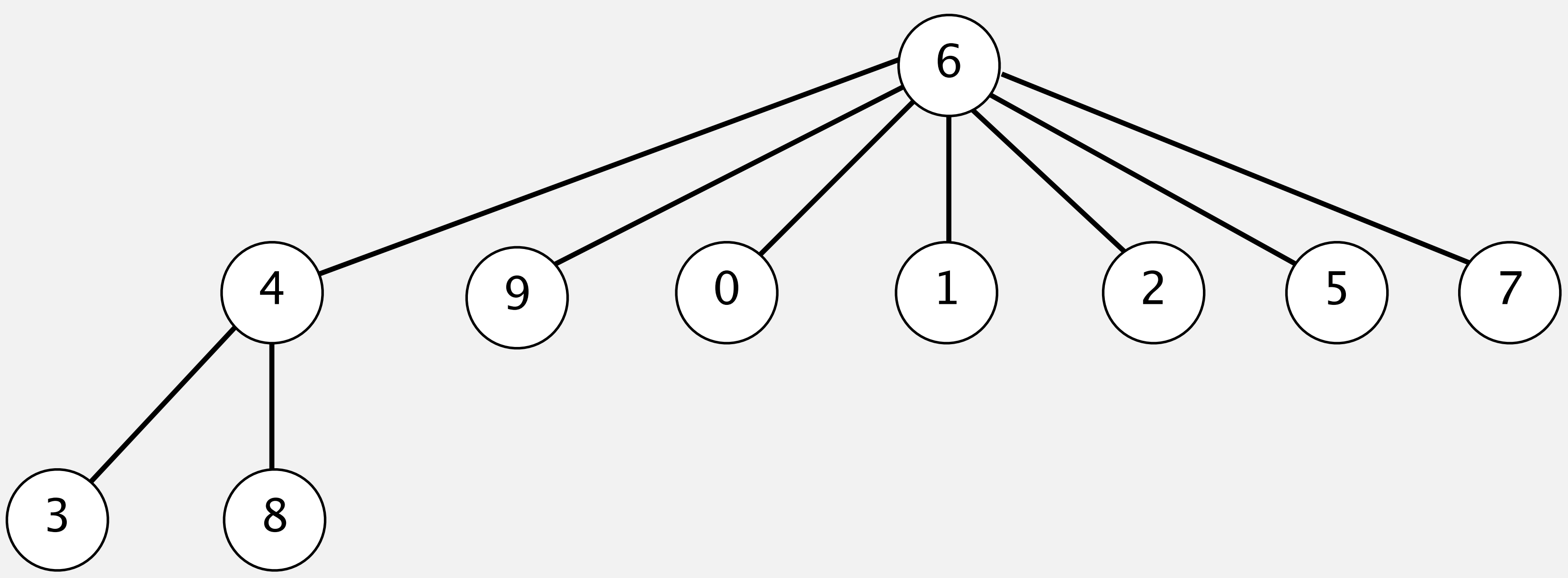
Weighted quick-union with path compression demo

connected(9, 1) ✔



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

Weighted quick-union with path compression demo



	0	1	2	3	4	5	6	7	8	9
id[]	6	6	6	4	6	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
build pq	1	E
delete-min	E	$\log E$
union	V	$\log^* V$ †
connected	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
build pq	1	E
delete-min	E	$\log E$
union	V	$\log^* V$ †
connected	E	$\log^* V$ †

† amortized bound using weighted quick union with path compression

recall: log* V ≤ 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	*optimal*	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	*optimal*	Pettie–Ramachandran
20xx	E	???

PRINCETON UNIVERSITY

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	*optimal*	Pettie–Ramachandran
20xx	E	???

PRINCETON UNIVERSITY

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