http://algs4.cs.princeton.edu

1.5 Union-Find

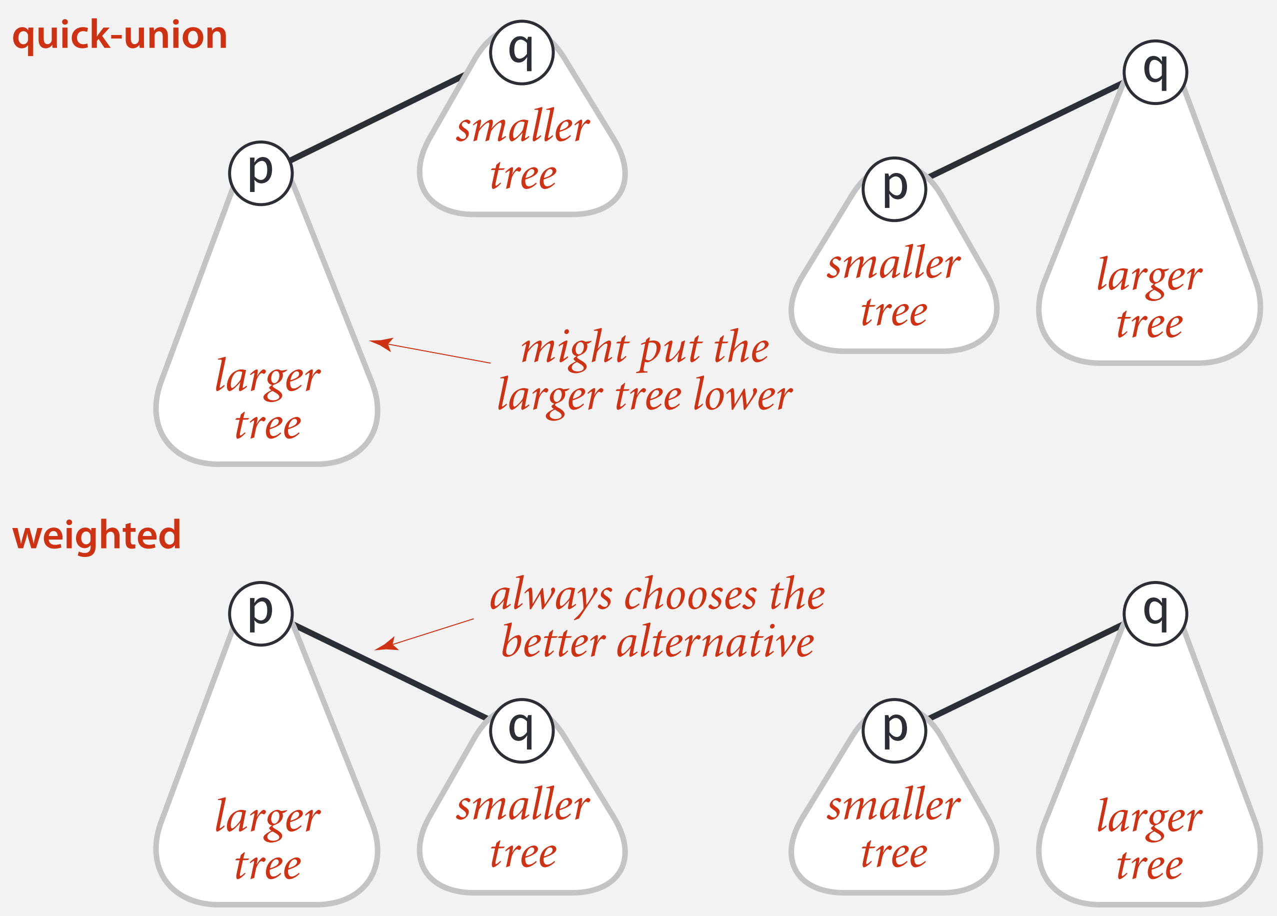
- *dynamic connectivity*
- *quick find*
- *quick union*
- ***improvements***
- *applications*

Improvement 1: weighting

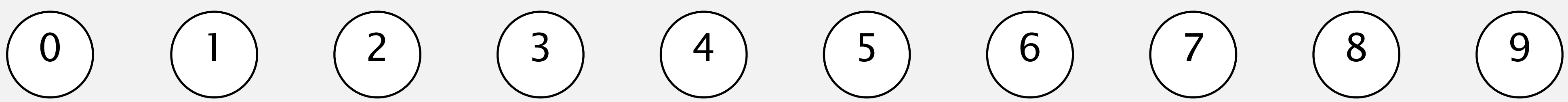
Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each tree (number of objects).
- Balance by linking root of smaller tree to root of larger tree.

reasonable alternatives:
union by height or "rank"

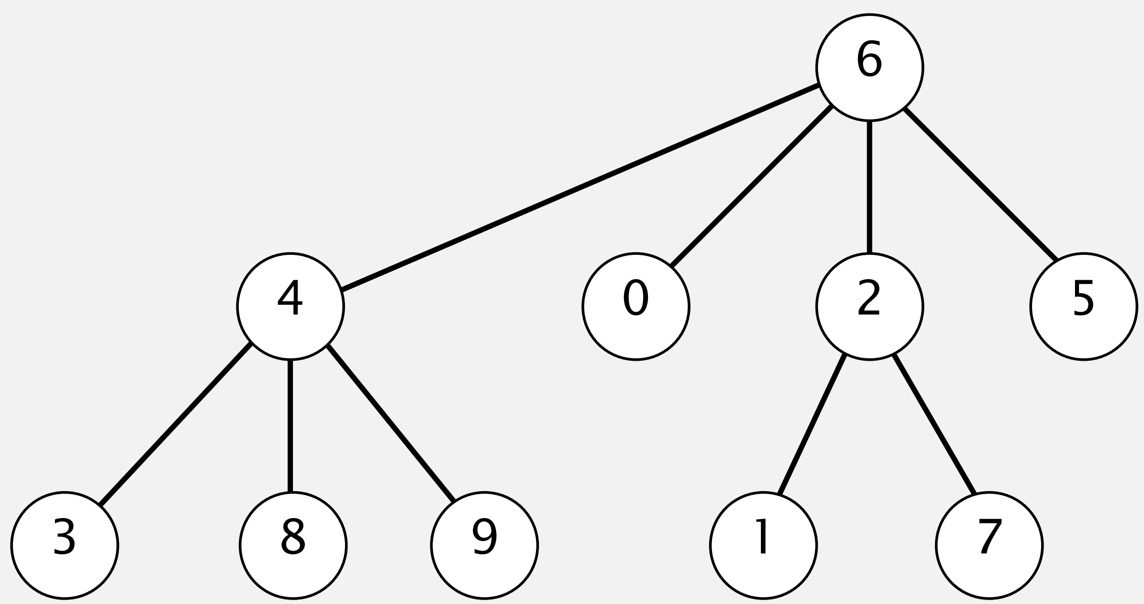


Weighted quick-union demo



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

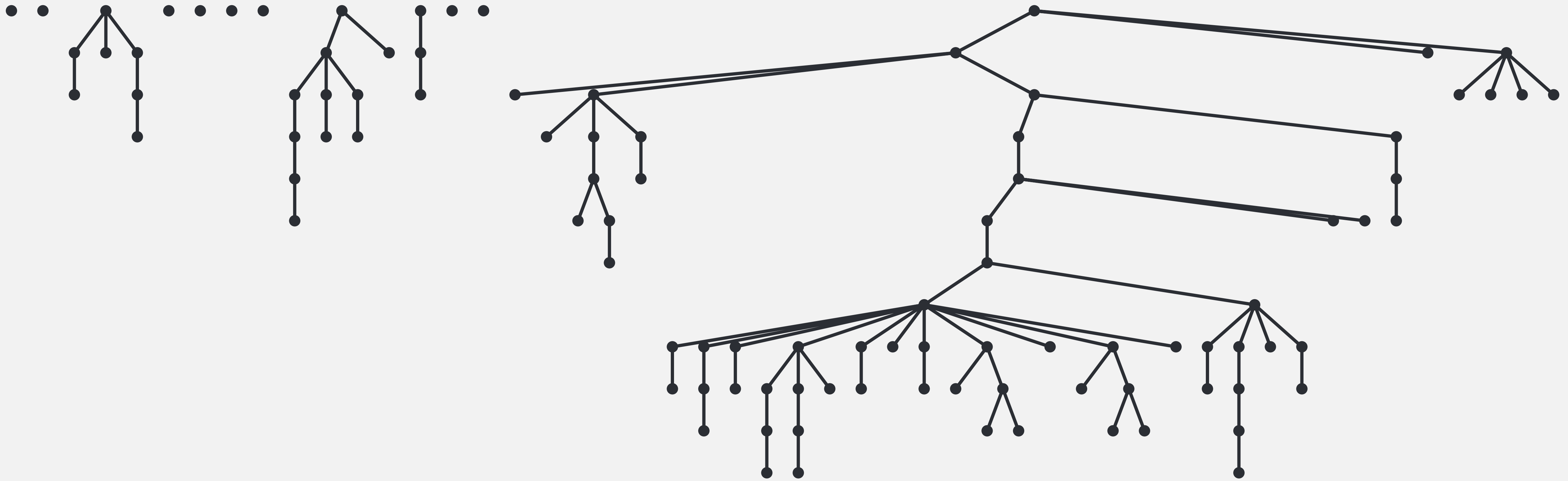
Weighted quick-union demo



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

Quick-union and weighted quick-union example

quick-union



average distance to root: 5.11

weighted

average distance to root: 1.52

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

Weighted quick-union: Java implementation

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

Find/connected. Identical to quick-union.

Union. Modify quick-union to:
- Link root of smaller tree to root of larger tree.
- Update the `sz[]` array.

```
int i = find(p);
int j = find(q);
if (i == j) return;
if  (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else                { id[j] = i; sz[i] += sz[j]; }
```

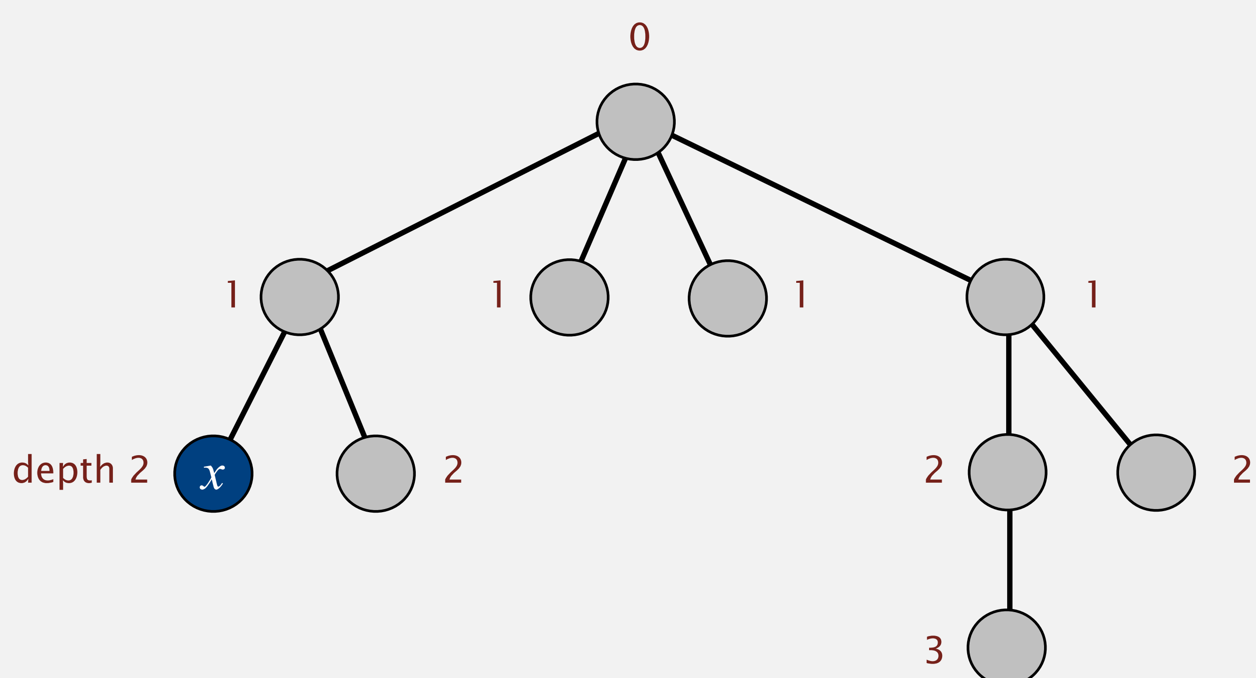
Weighted quick-union analysis

Running time.

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

lg = base-2 logarithm

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



N = 10
depth(x) = 3 ≤ lg N

Weighted quick-union analysis

Running time.

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

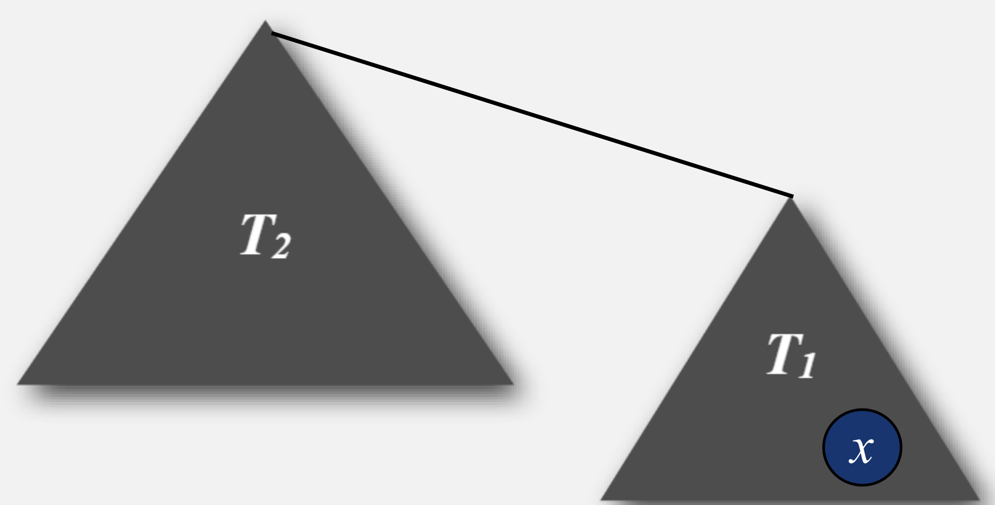
lg = base-2 logarithm

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

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

Increases by 1 when tree T_1 containing x is merged into another tree T_2.

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



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

Weighted quick-union analysis

Running time.

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

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

algorithm	initialize	union	find	connected
quick-find	N	N	1	1
quick-union	N	N †	N	N
weighted QU	N	lg N †	lg N	lg N

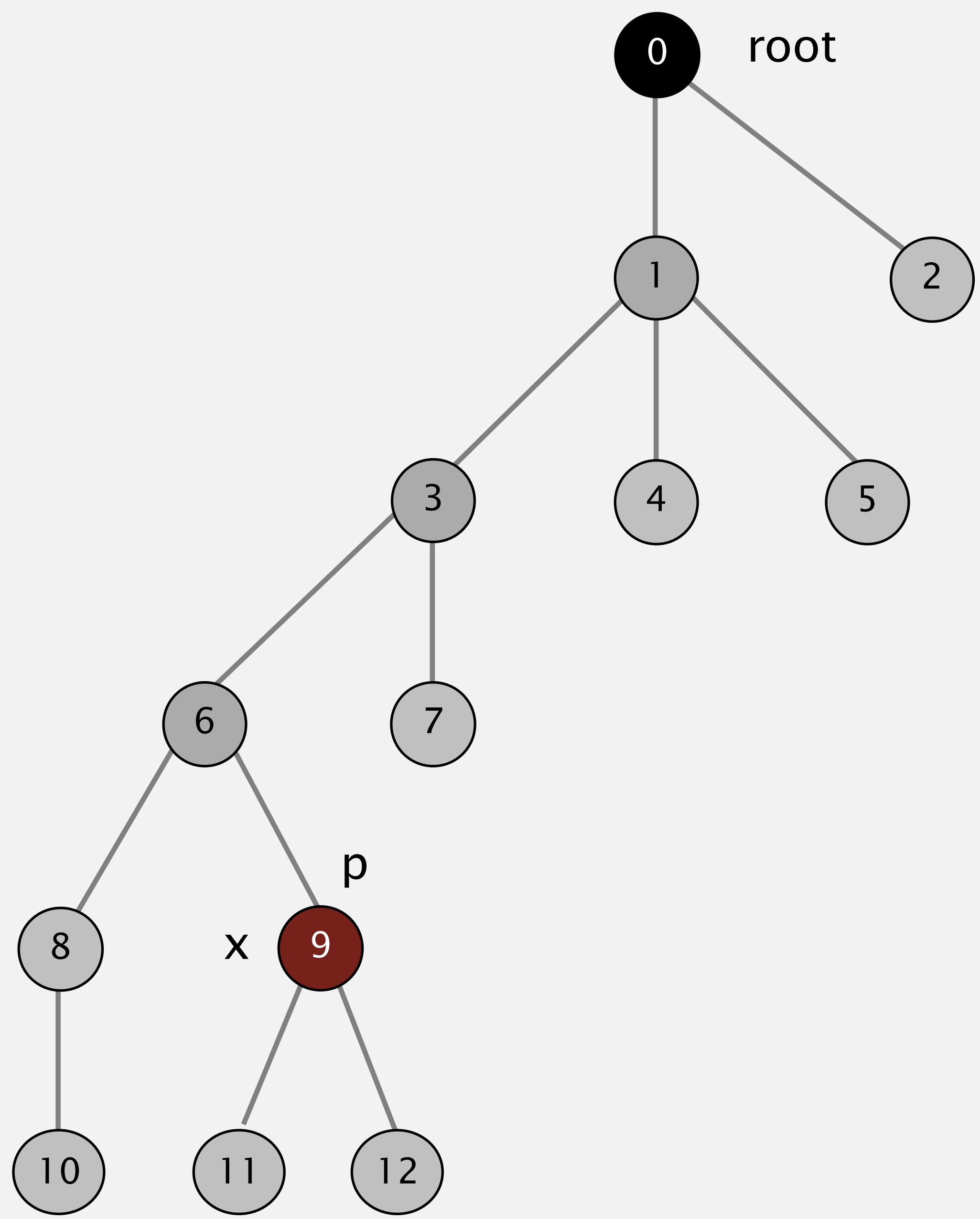
† includes cost of finding roots

Q. Stop at guaranteed acceptable performance?

A. No, easy to improve further.

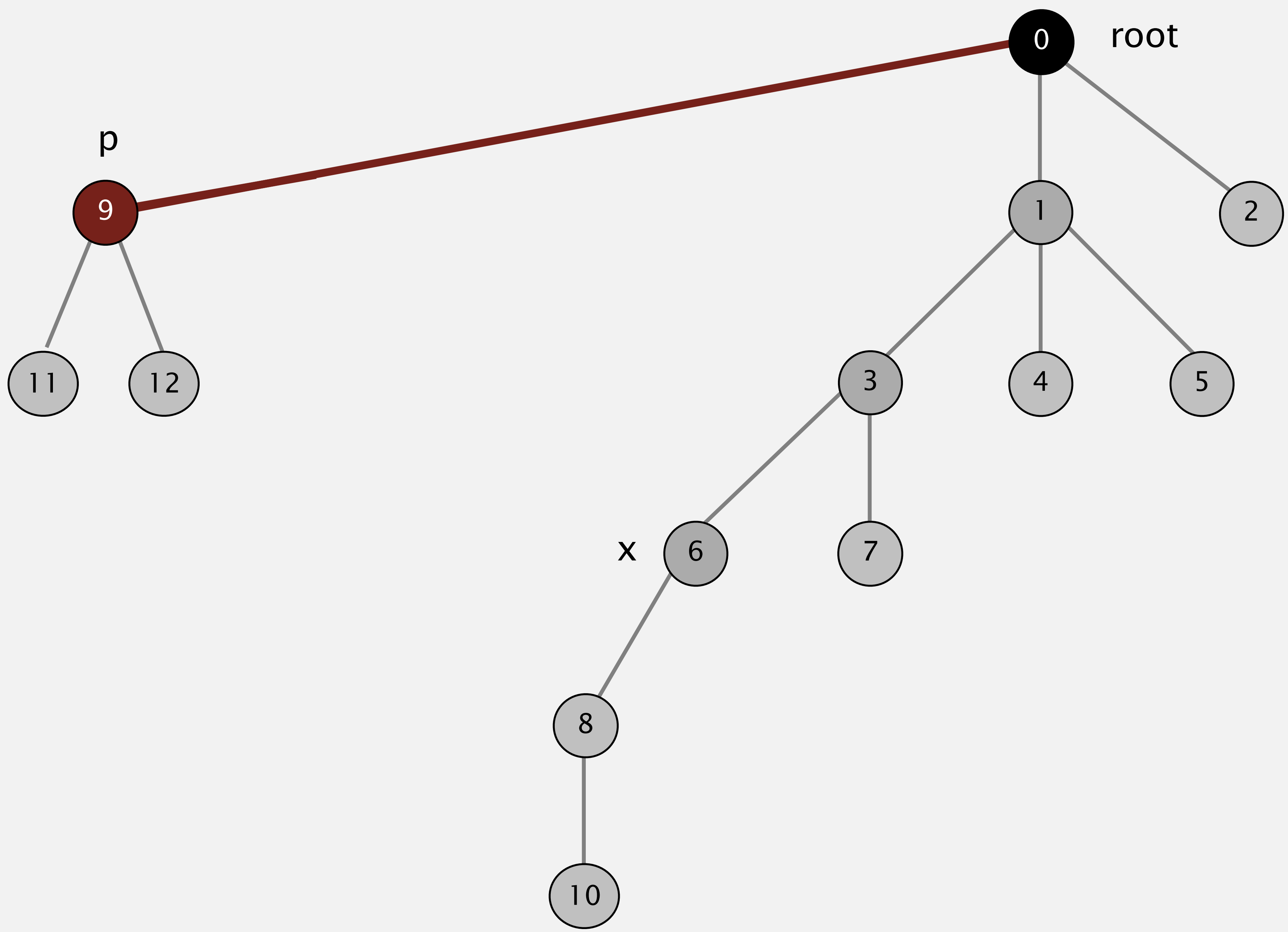
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the `id[]` of each examined node to point to that root.



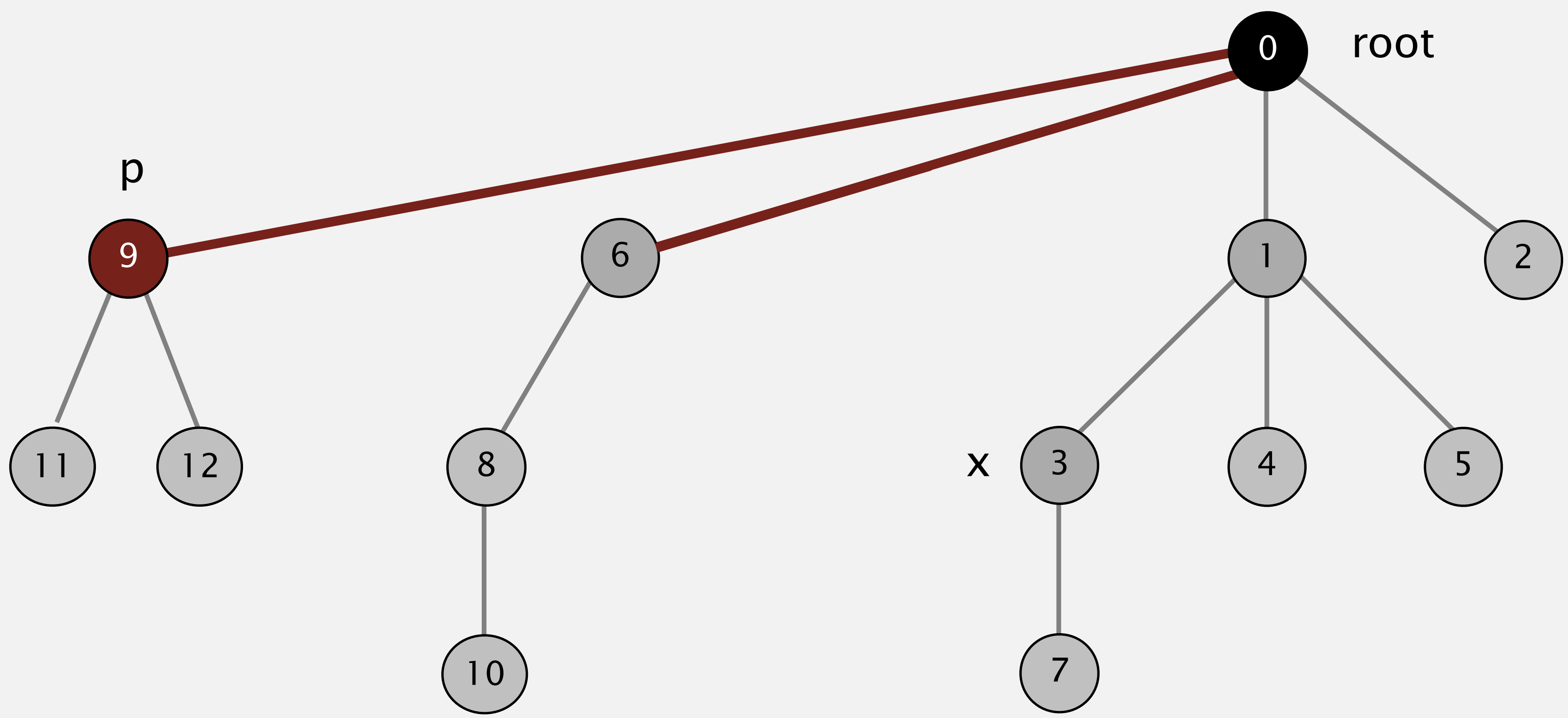
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the `id[]` of each examined node to point to that root.



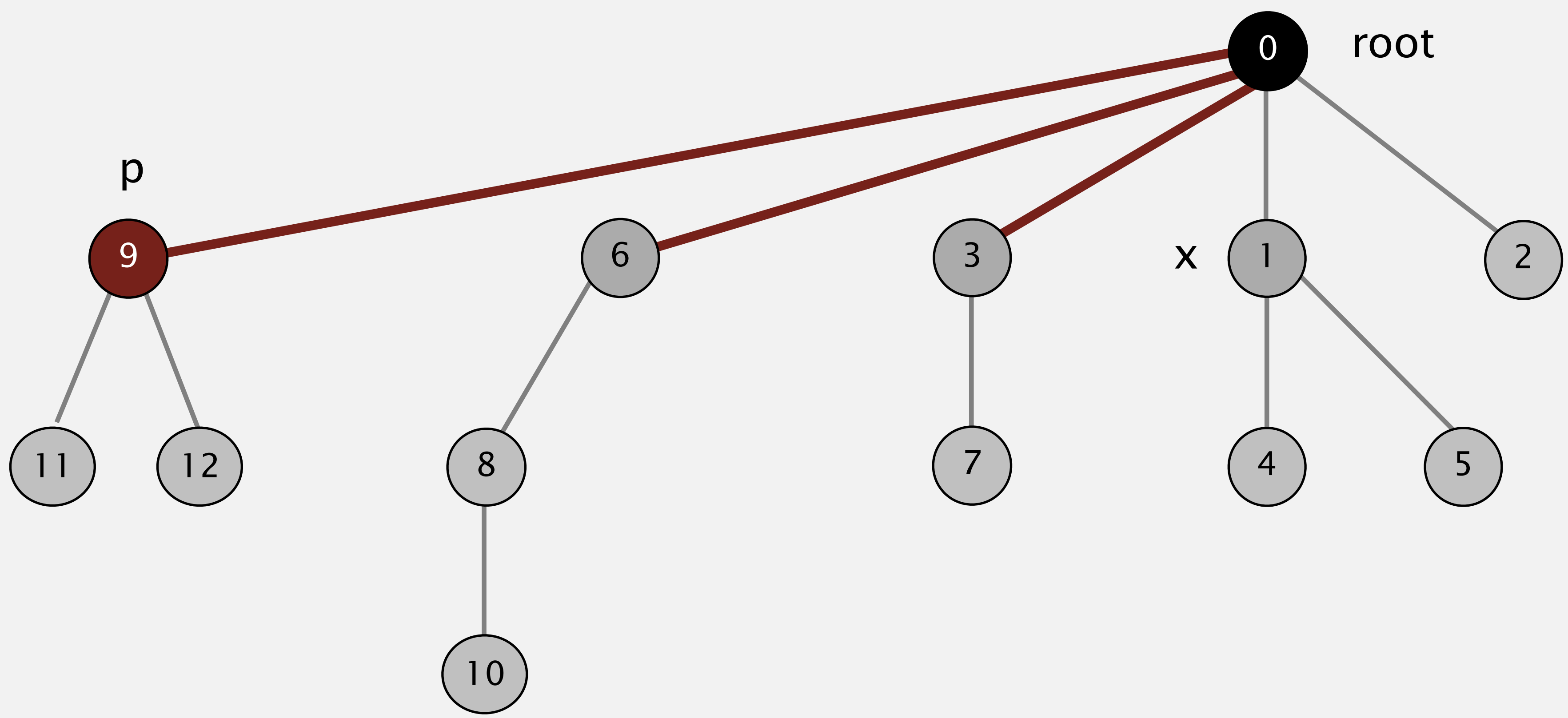
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the `id[]` of each examined node to point to that root.



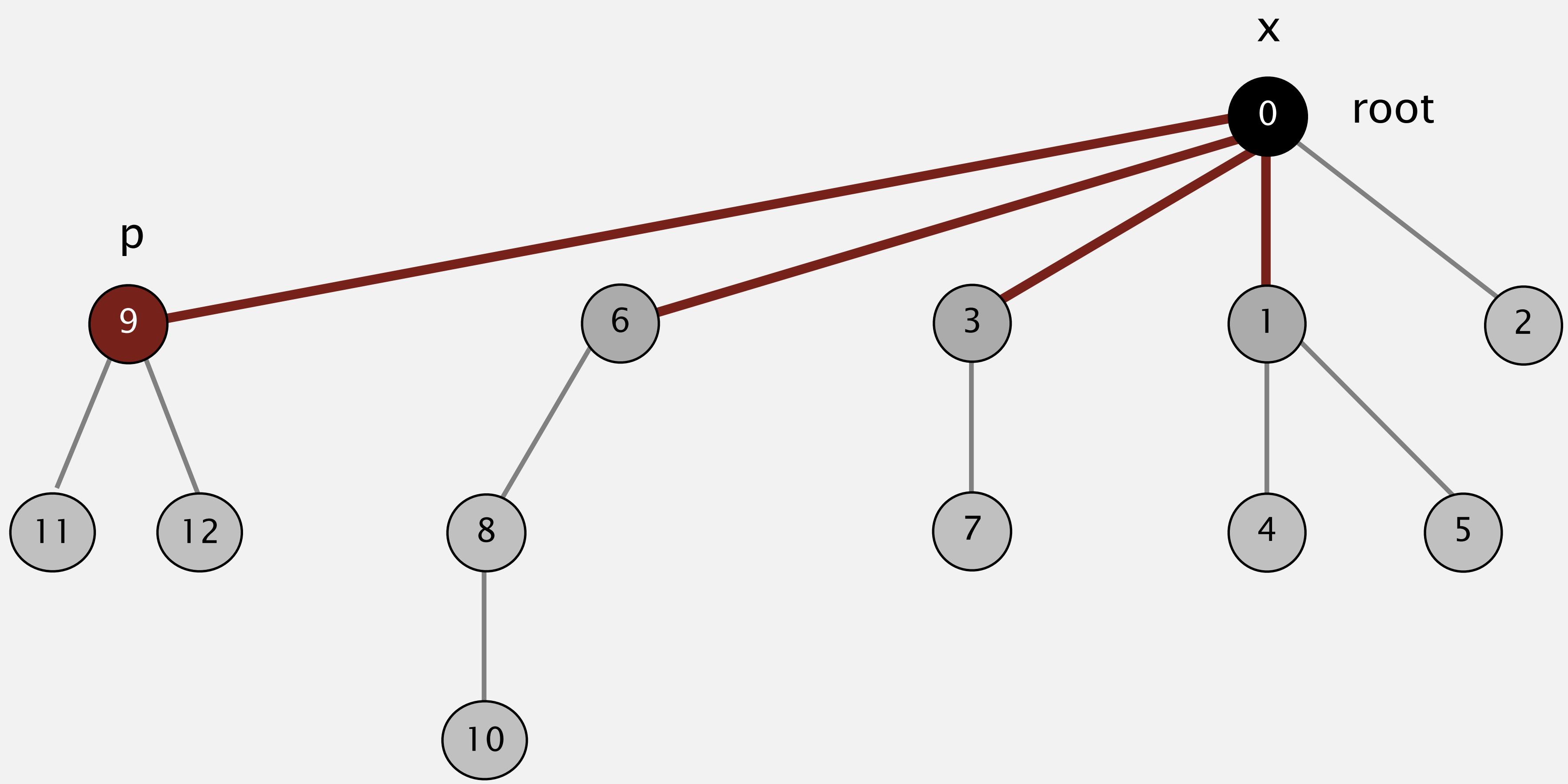
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the `id[]` of each examined node to point to that root.



Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the `id[]` of each examined node to point to that root.



Bottom line. Now, `find()` has the side effect of compressing the tree.

Path compression: Java implementation

Two-pass implementation: add second loop to `find()` to set the `id[]` of each examined node to the root.

Simpler one-pass variant (path halving): Make every other node in path point to its grandparent.

```
public int find(int i)
{
   while (i != id[i])
   {
      id[i] = id[id[i]];
      i = id[i];
   }
   return i;
}
```

only one extra line of code !

In practice. No reason not to! Keeps tree almost completely flat.

Weighted quick-union with path compression: amortized analysis

Proposition. [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of M union–find ops on N objects makes ≤ $c\,(\,N + M \lg^*N\,)$ array accesses.

- Analysis can be improved to $N + M\,\alpha(M, N)$.
- Simple algorithm with fascinating mathematics.

N	lg* N
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

iterated lg function

Linear-time algorithm for M union-find ops on N objects?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

Amazing fact. [Fredman-Saks] No linear-time algorithm exists.

in "cell-probe" model of computation

Summary

Key point. Weighted quick union (and/or path compression) makes it possible to solve problems that could not otherwise be addressed.

algorithm	worst-case time
quick-find	M N
quick-union	M N
weighted QU	N + M log N
QU + path compression	N + M log N
weighted QU + path compression	N + M lg* N

order of growth for M union-find operations on a set of N objects

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

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