Introduction to Algorithms

Subtext of lecture

Steps to developing a usable algorithm.

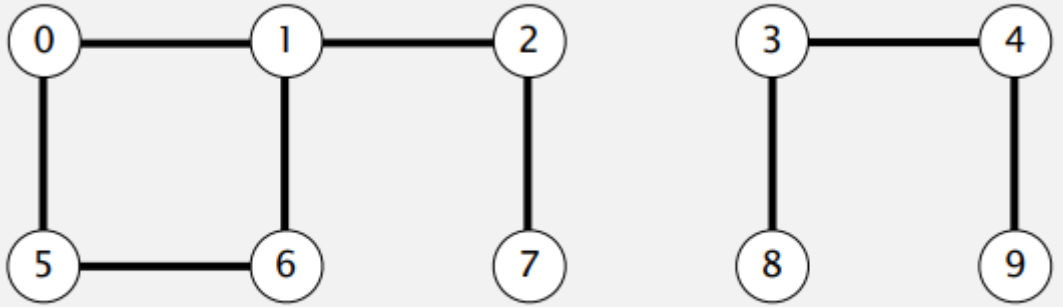
- Model the problem.
- Find an algorithm to solve it.
- Fast enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

Dynamic Connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

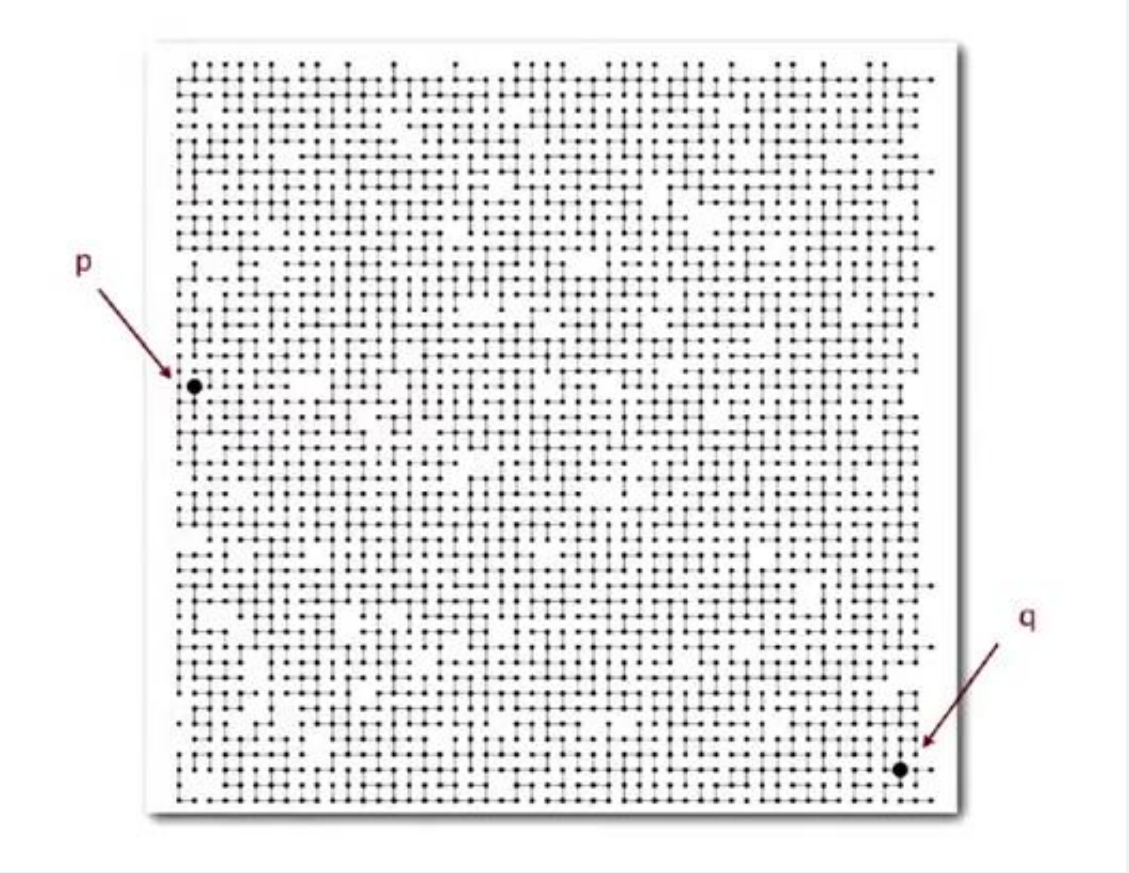
```
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
```

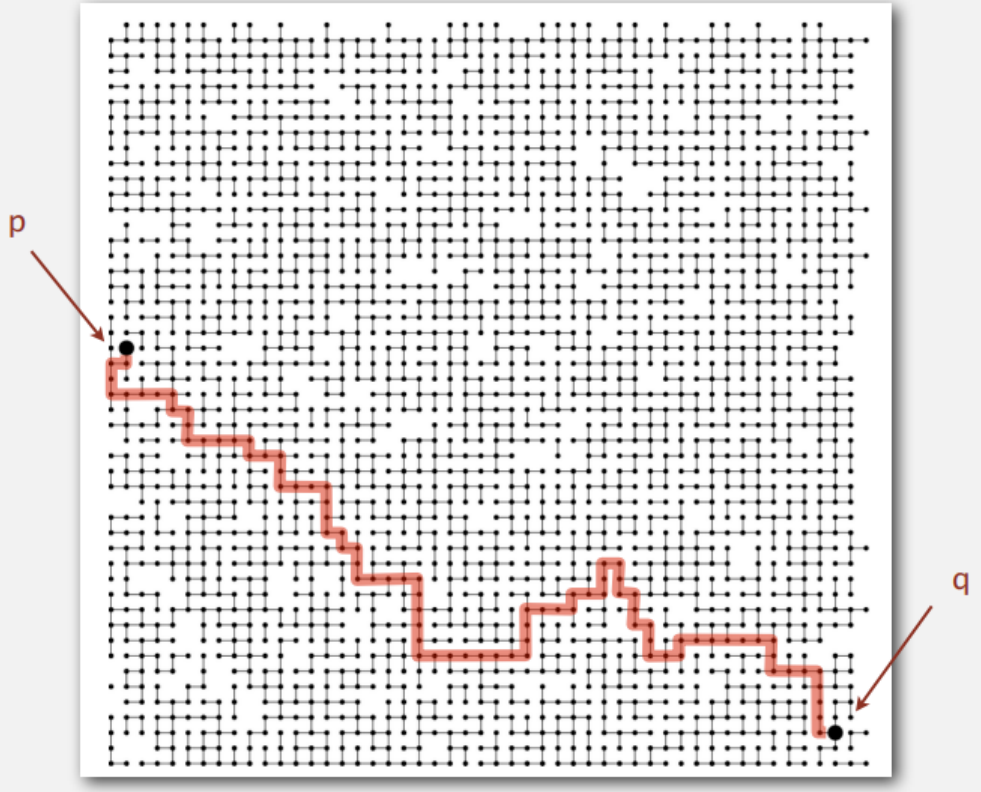



```
connected(0, 7)
connected(8, 9)
```

Dynamic Connectivity

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





Dynamic Connectivity

Applications involve manipulating objects of all types.

- Pixels in a digital photo.
- Computers in a network.
- Friends in a social network.
- Transistors in a computer chip.
- Elements in a mathematical set.
- Variable names in Fortran program.
- Metallic sites in a composite system.

Dynamic Connectivity

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

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

Dynamic Connectivity

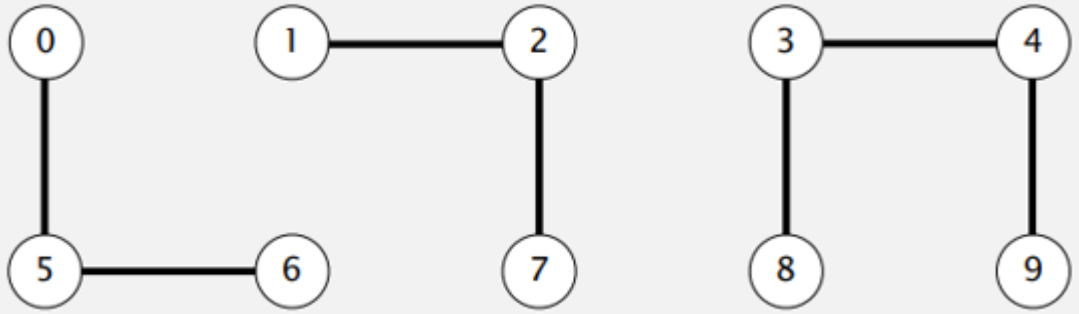
Goal. Design efficient data structure for union-find.

- Number of objects *N* can be huge.
- Number of operations *M* can be huge.
- Find queries and union commands may be intermixed.

Dynamic Connectivity

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

Your suggestions?



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

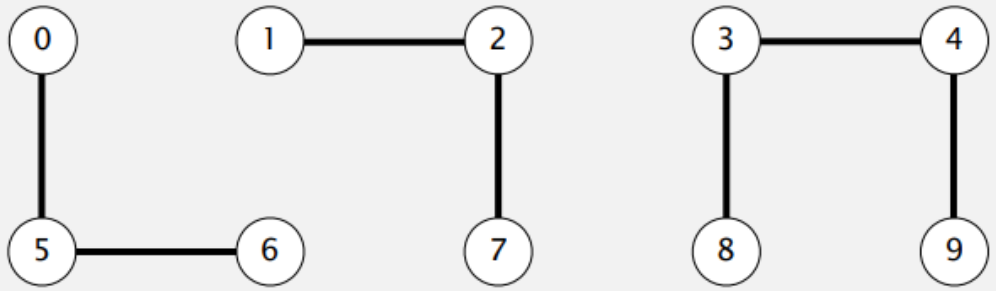
Quick Find

Data structure.

- Integer array *id[]* of length *N*.
- Interpretation: *p* and *q* are connected if and only if they have the same id.

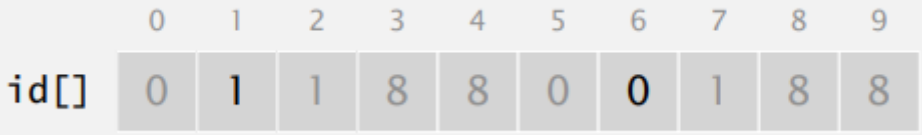
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	0	0	1	8	8

0, 5 and 6 are connected
1, 2, and 7 are connected
3, 4, 8, and 9 are connected



Quick Find

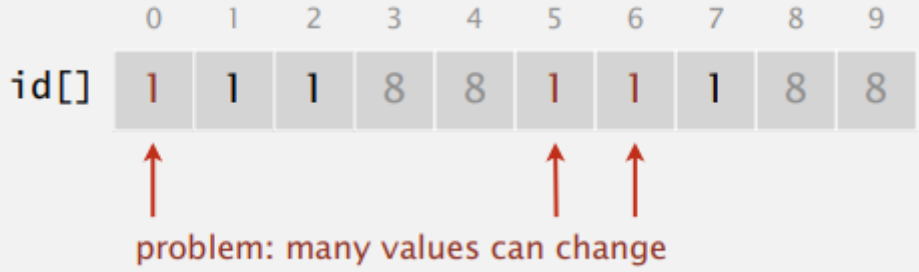
p and *q* are connected if and only if they have the same id.



Find. Check if *p* and *q* have the same id.

id[6] = 0; id[1] = 1
6 and 1 are not connected

Union. To merge components containing *p* and *q*, change all entries whose id equals *id[p]* to *id[q]*.



union(6, 1)

Quick Find Step-by-Step

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

Quick Find Effectiveness

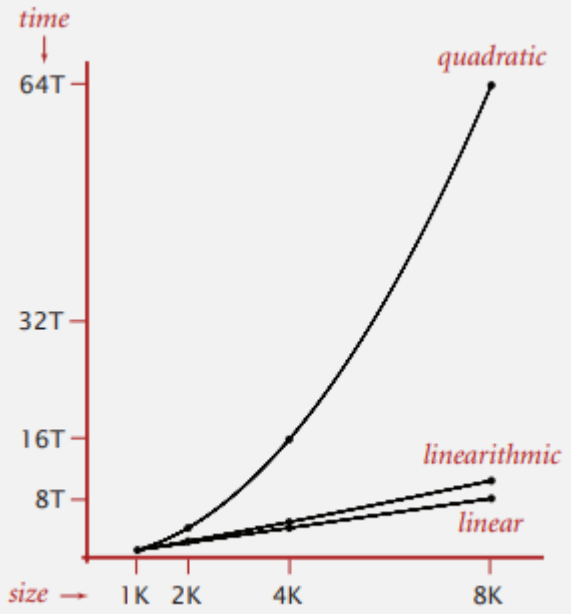
Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	N	N	1

order of growth of number of array accesses

Union is too expensive. It takes N^2 array accesses to process a sequence of N union commands on N objects.

What does N^2 mean?



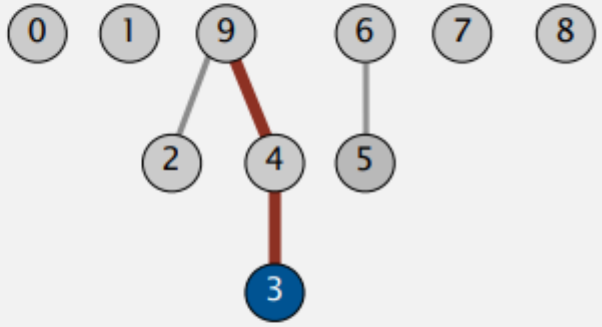
Quick Union

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change
(algorithm ensures no cycles)

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



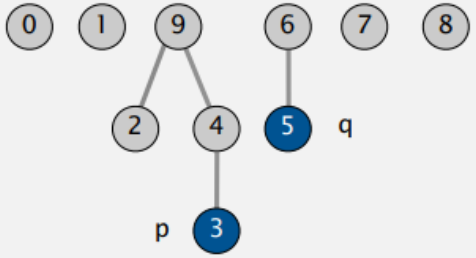
root of 3 is 9

Quick Union

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

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



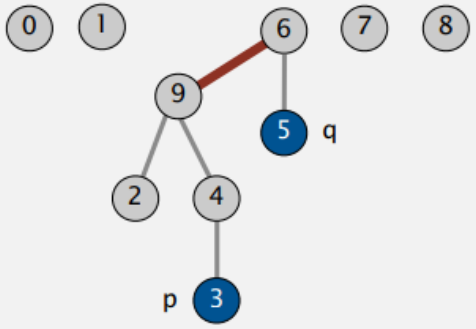
root of 3 is 9
root of 5 is 6
3 and 5 are not connected

Find. Check if `p` and `q` have the same root.

Union. To merge components containing `p` and `q`, set the id of `p`'s root to the id of `q`'s root.

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

only one value changes



Quick Union Step-by-Step

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

Quick Union Effectiveness

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

algorithm	initialize	union	find
quick-find	N	N	1
quick-union	N	N †	N

← worst case

† includes cost of finding roots

Quick-find defect.

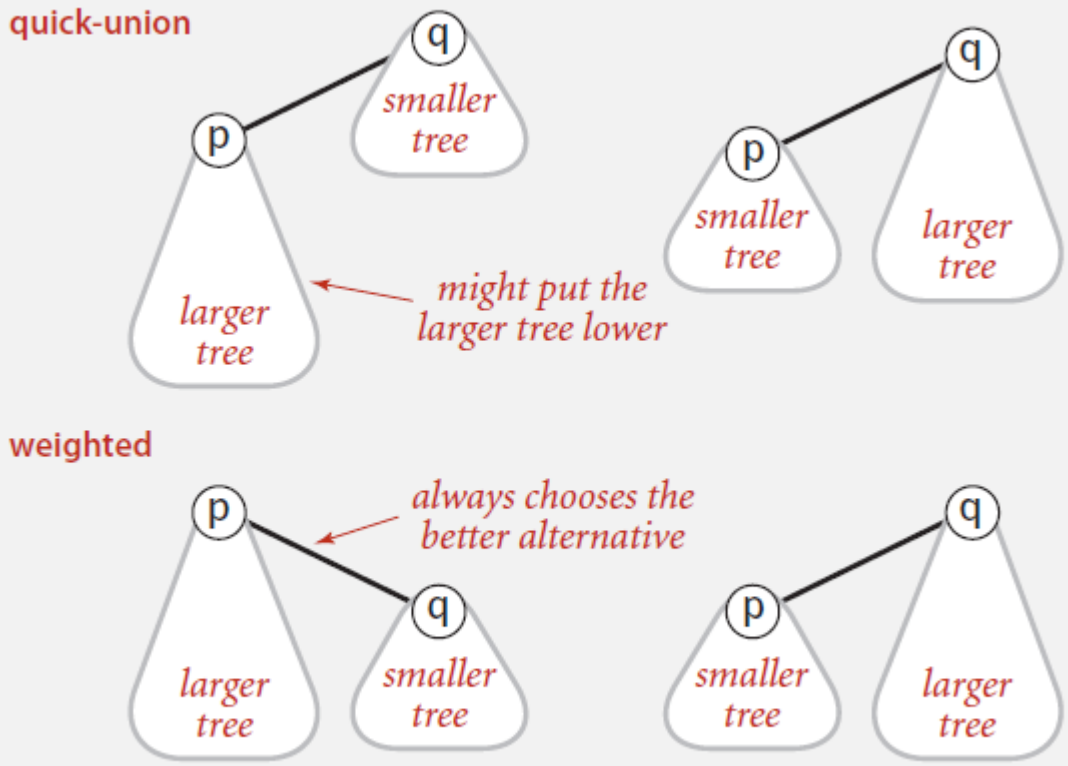
- Union too expensive (N array accesses).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.

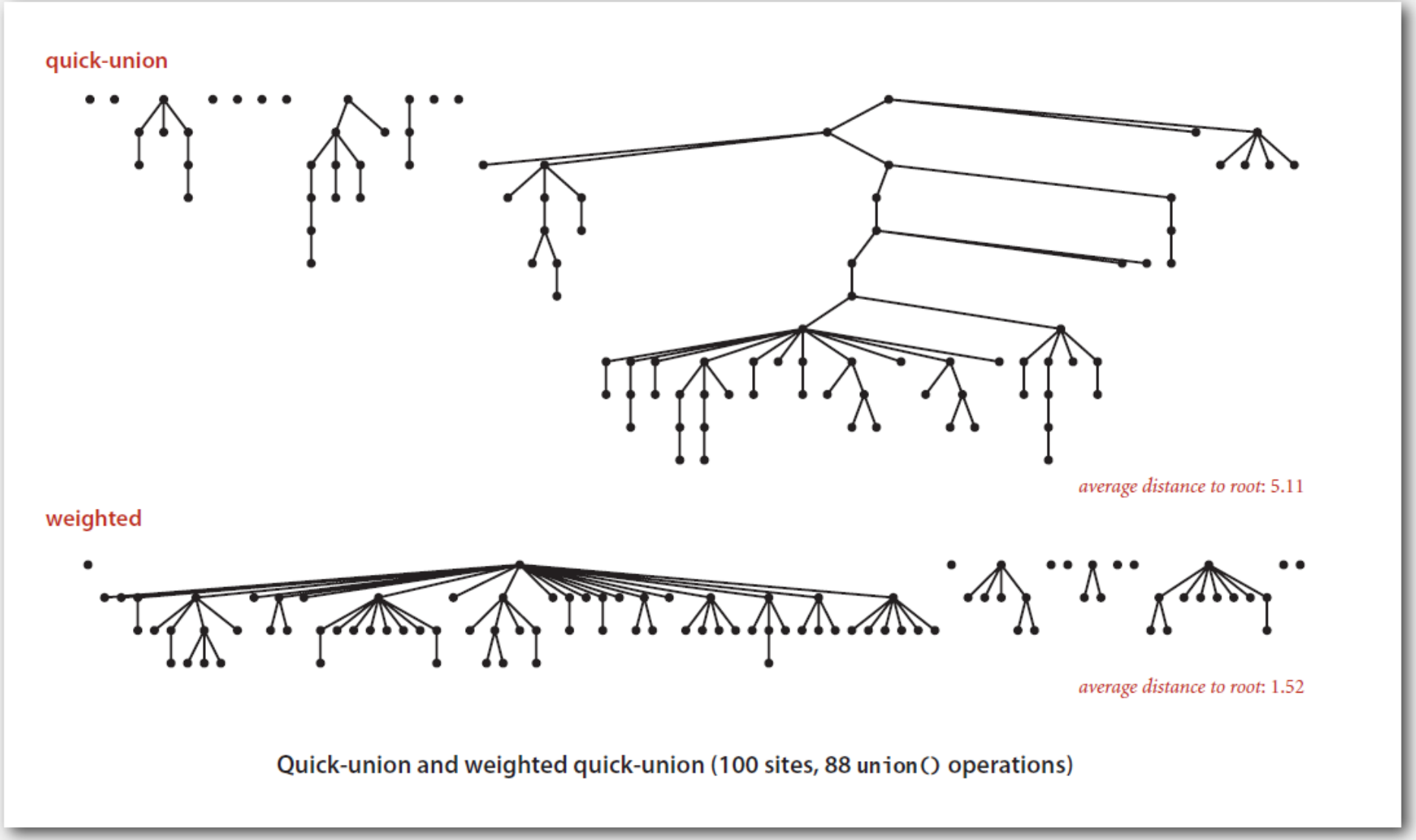
- Trees can get tall.
- Find too expensive (could be N array accesses).

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.



Weighted quick-union



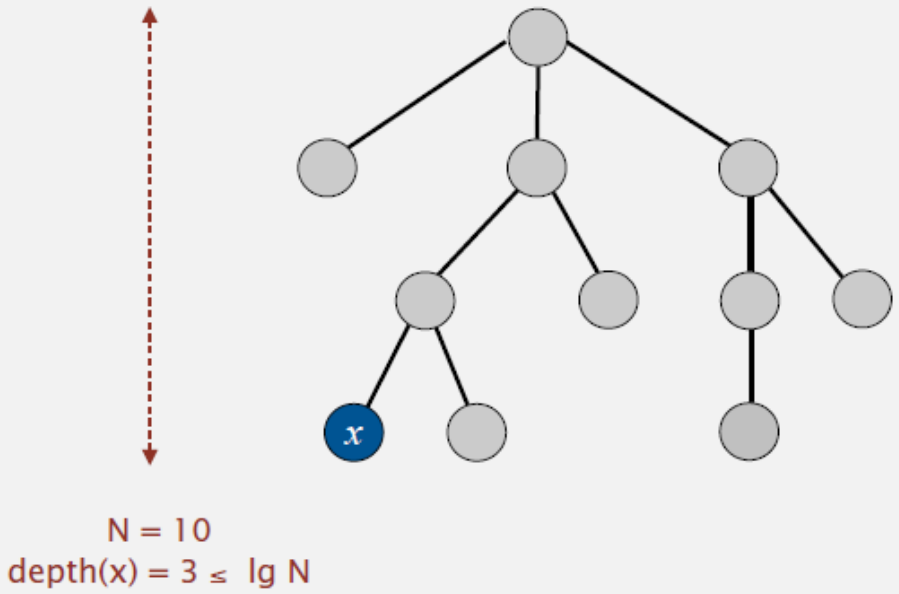
Quick-union and weighted quick-union (100 sites, 88 `union()` operations)

Weighted quick-union

Running time.

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

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



Weighted quick-union

Running time.

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

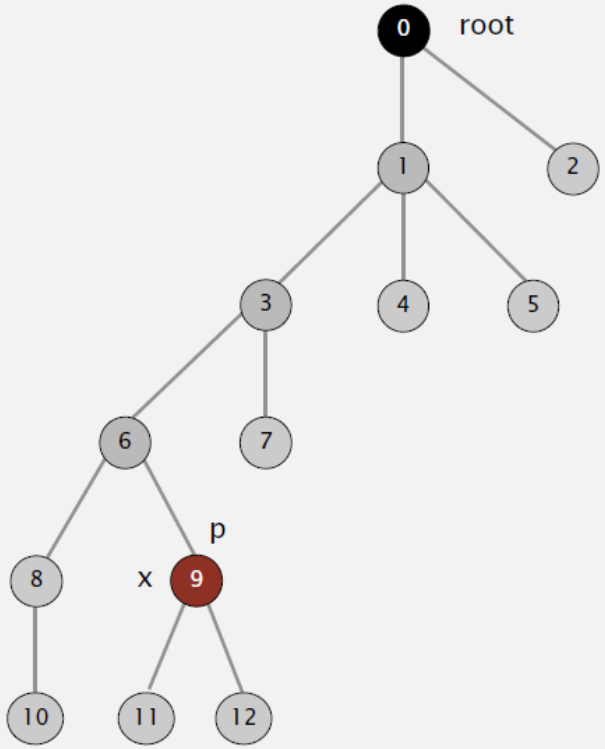
Proposition. Depth of any node *x* is at most lg *N*.

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

† includes cost of finding roots

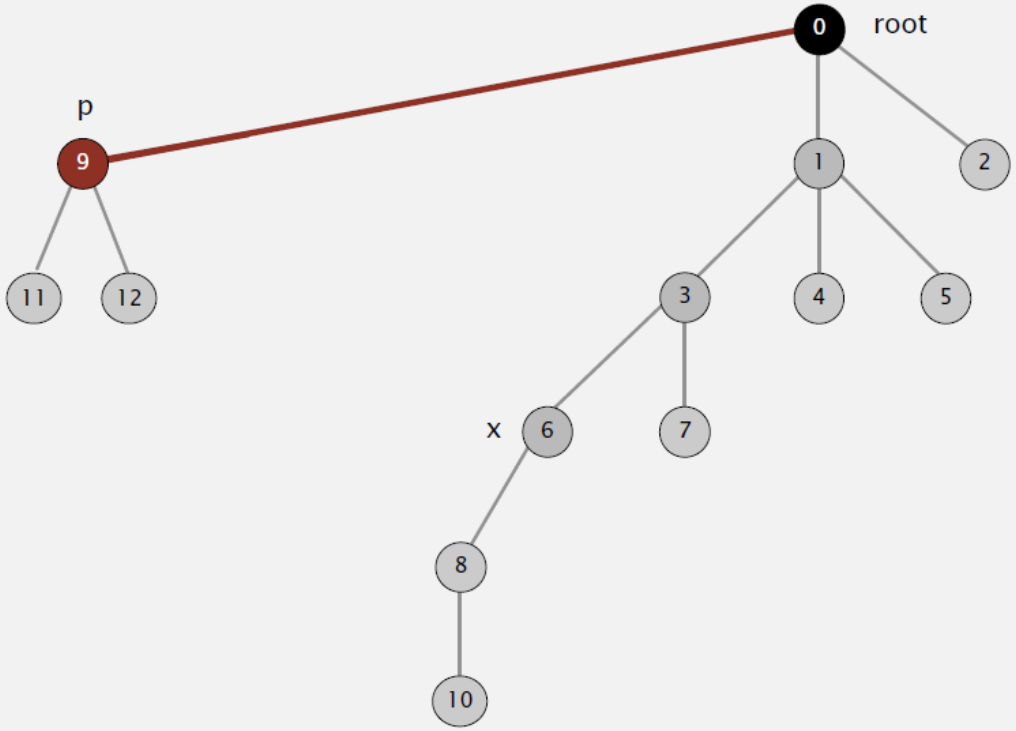
Path compression

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



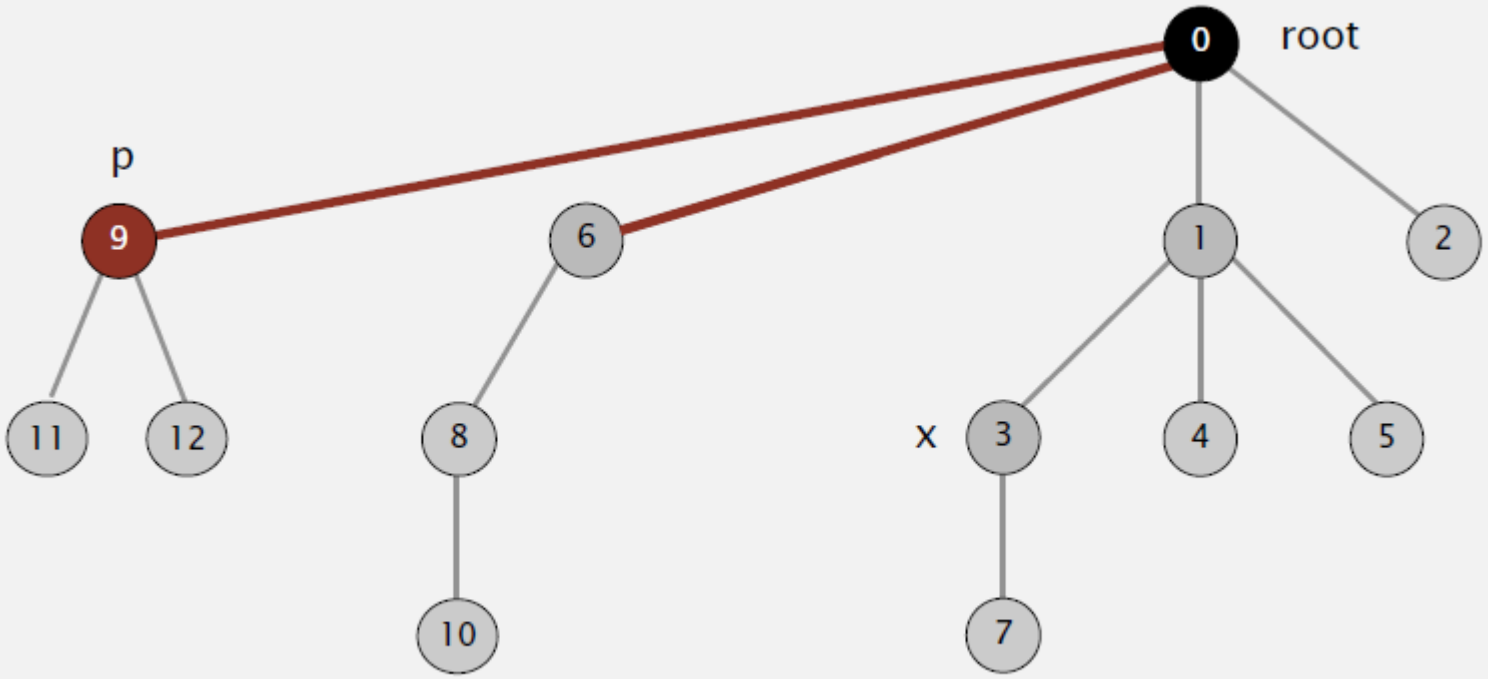
Path compression

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



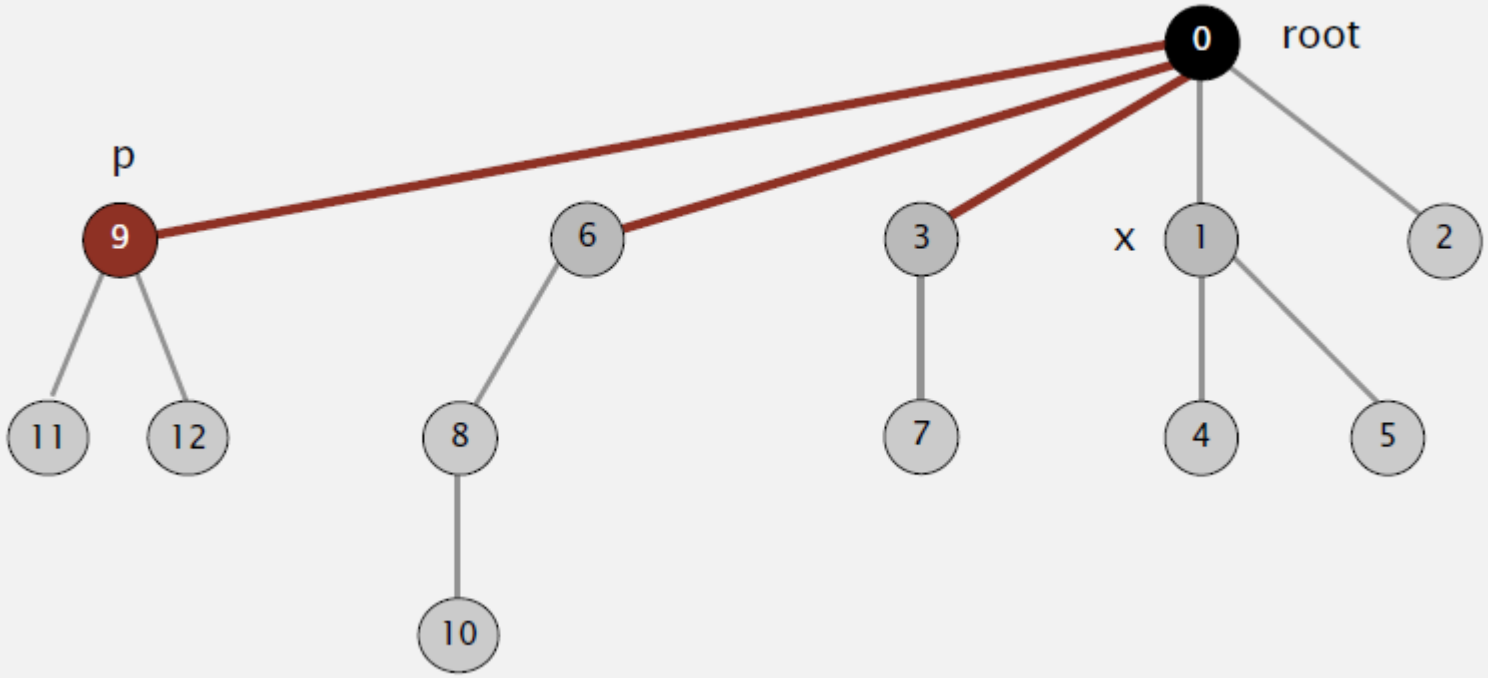
Path compression

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



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.



Summary

Bottom line. Weighted quick union (with path compression) makes it possible to solve problems that could not otherwise be addressed.

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

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

M union-find operations on a set of N objects