Disjoint Sets

EECS 214

November 16, 2015

Take-aways

- What does the union-find ADT do?
- What might it be useful for?
- What are some possible data structures for union-find?
- How does the ranked, path-compressed forest union-find data structure work?
- Why is it efficient?

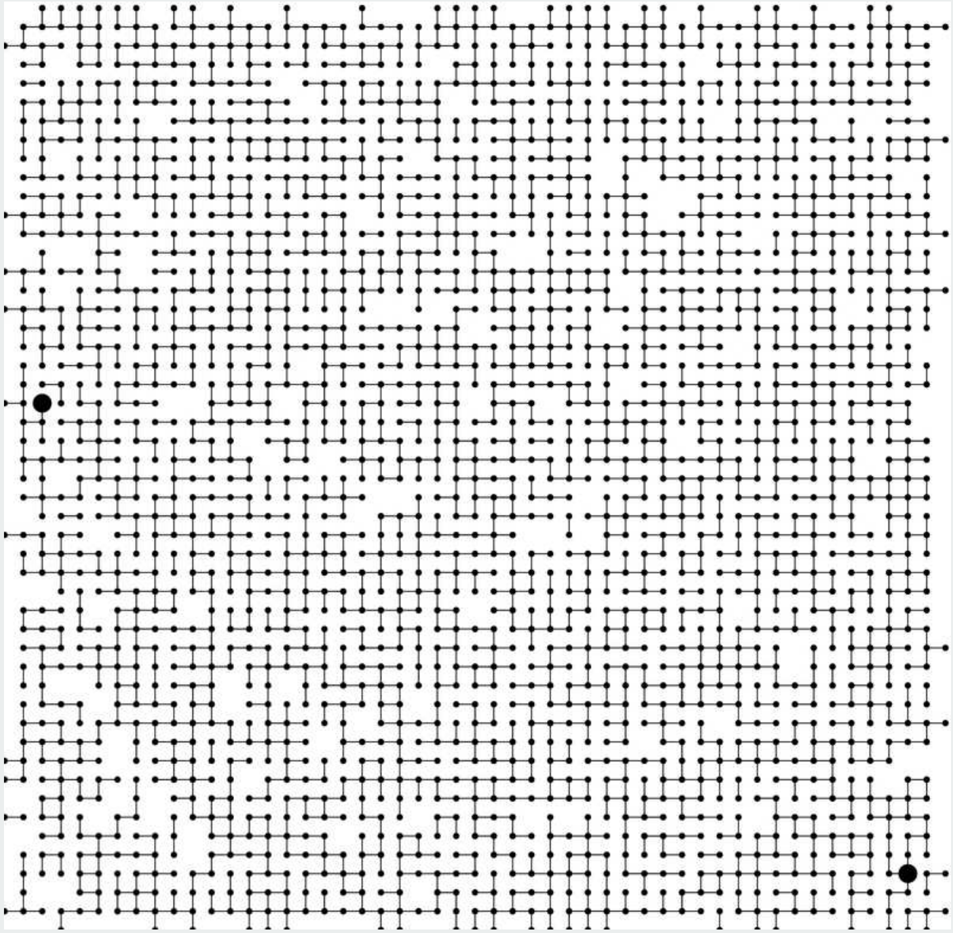
Following slides are from

https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf

Network connectivity

Basic abstractions

- set of objects
- union command: connect two objects
- find query: is there a path connecting one object to another?



Objects

Union-find applications involve manipulating objects of all types.

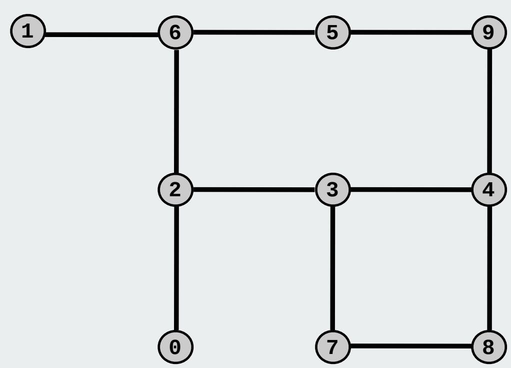
- Computers in a network.
- Web pages on the Internet.
- Transistors in a computer chip.
- Variable name aliases.
- Pixels in a digital photo.
- Metallic sites in a composite system.

stay tuned

When programming, convenient to name them 0 to N-1.

- Hide details not relevant to union-find.
- Integers allow quick access to object-related info.
- Could use symbol table to translate from object names

use as array index



Union-find abstractions

Simple model captures the essential nature of connectivity.

- Objects.

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

grid points

- Disjoint sets of objects.

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

subsets of connected grid points

- Find query: are objects 2 and 9 in the same set?

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

are two grid points connected?

- Union command: merge sets containing 3 and 8.

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

add a connection between two grid points

Connected components

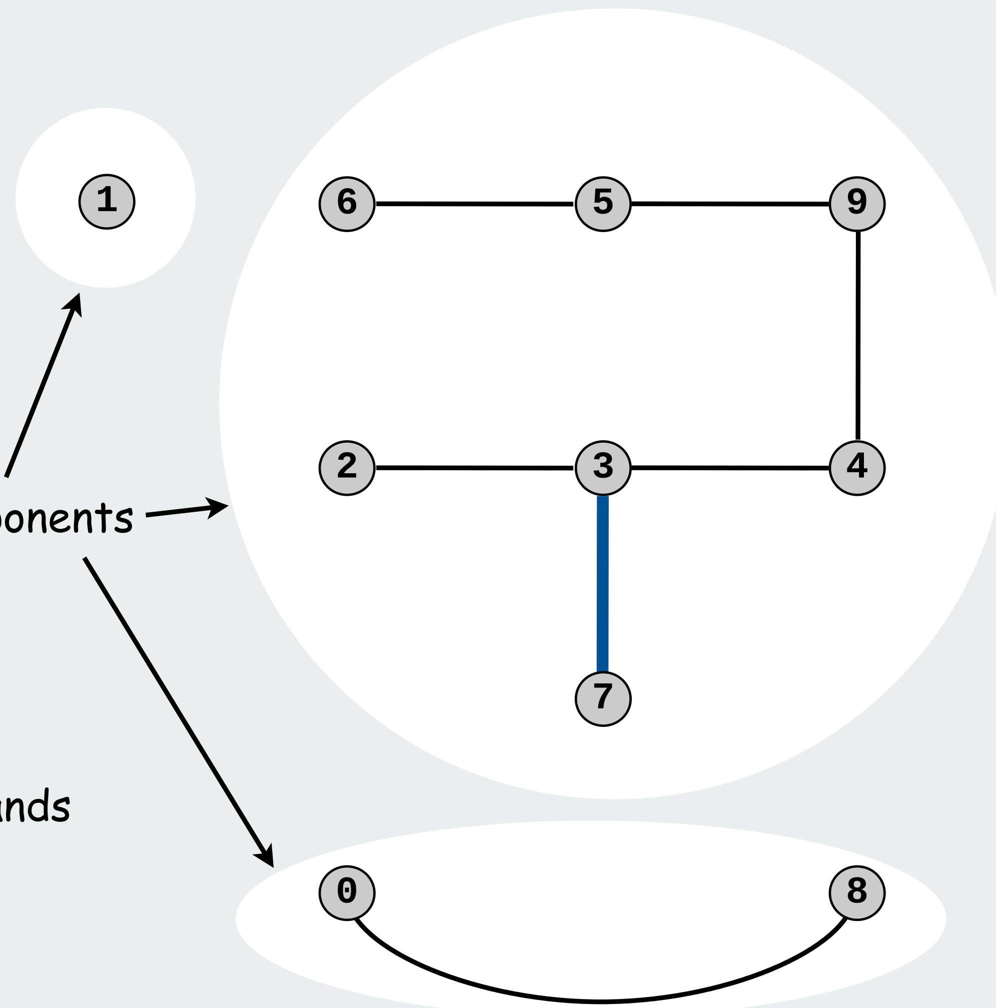
Connected component: set of mutually connected vertices

Each union command reduces by 1 the number of components

in	out
3 4	3 4
4 9	4 9
8 0	8 0
2 3	2 3
5 6	5 6
2 9	
5 9	5 9
7 3	7 3

7 union commands

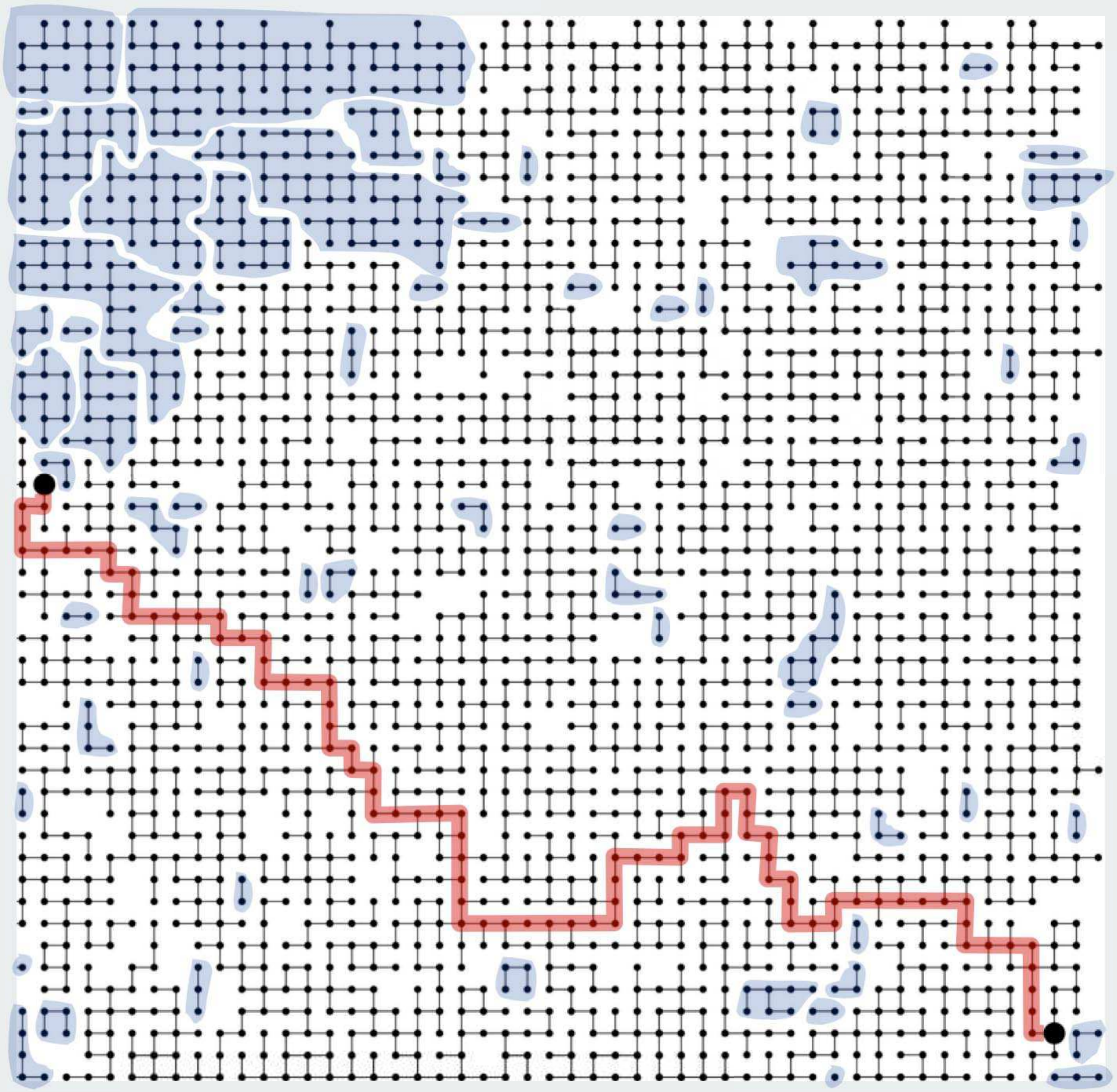
3 = 10-7 components



Network connectivity: larger example

find(u, v) ?

Network connectivity: larger example



find(u, v) ?

true

63 components

Union-find abstractions

- Objects.
- Disjoint sets of objects.
- Find queries: are two objects in the same set?
- Union commands: replace sets containing two items by their union

Goal. Design efficient data structure for union-find.

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

Quick-find [eager approach]

Data structure.

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

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

5 and 6 are connected
2, 3, 4, and 9 are connected

Quick-find [eager approach]

Data structure.

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

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

5 and 6 are connected
2, 3, 4, and 9 are connected

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

id[3] = 9; id[6] = 6
3 and 6 not connected

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

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

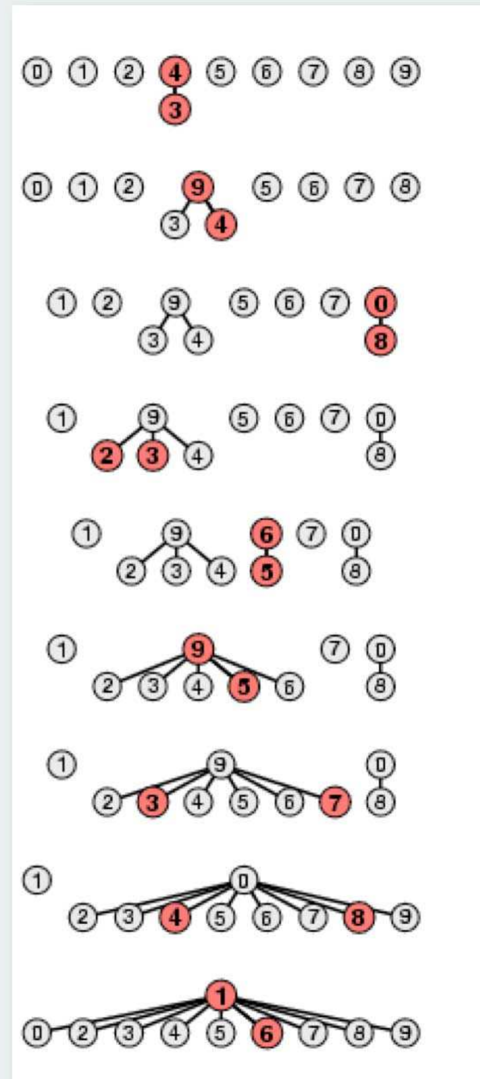
union of 3 and 6
2, 3, 4, 5, 6, and 9 are connected

problem: many values can change

Quick-find example

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

problem: many values can change



Quick-find is too slow

Quick-find algorithm may take ~MN steps to process M union commands on N objects

Rough standard (for now).

- 10^9 operations per second.
- 10^9 words of main memory.
- Touch all words in approximately 1 second.

a truism (roughly) since 1950 !

Ex. Huge problem for quick-find.

- 10^{10} edges connecting 10^9 nodes.
- Quick-find takes more than 10^{19} operations.
- 300+ years of computer time!

Paradoxically, quadratic algorithms get worse with newer equipment.

- New computer may be 10x as fast.
- But, has 10x as much memory so problem may be 10x bigger.
- With quadratic algorithm, takes 10x as long!

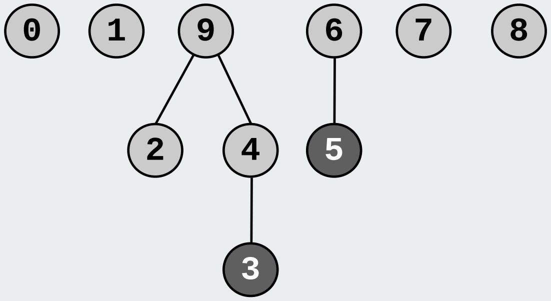
Quick-union [lazy approach]

Data structure.

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

keep going until it doesn't change

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



3's root is 9; 5's root is 6

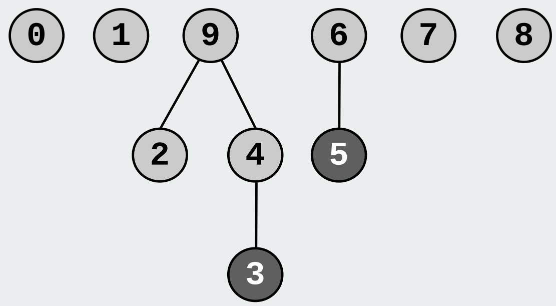
Quick-union [lazy approach]

Data structure.

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

keep going until it doesn't change

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



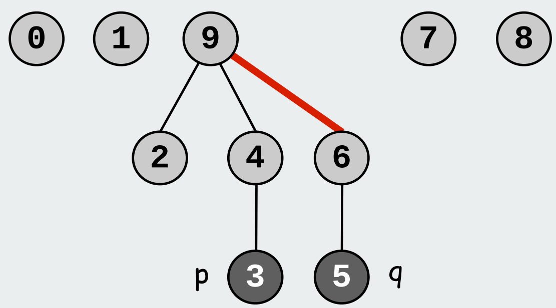
3's root is 9; 5's root is 6
3 and 5 are not connected

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

Union. Set the id of q's root to the id of p's root.

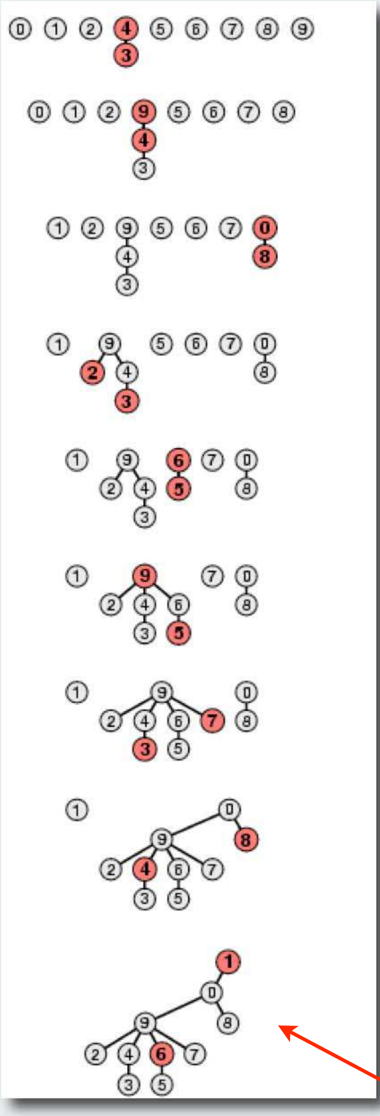
i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	9	6	9	7	8	9

only one value changes



Quick-union example

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



problem: trees can get tall

Quick-union is also too slow

Quick-find defect.

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

Quick-union defect.

- Trees can get tall.
- Find too expensive (could be N steps)
- Need to do find to do union

algorithm	union	find
Quick-find	N	1
Quick-union	N*	N ← worst case

* includes cost of find

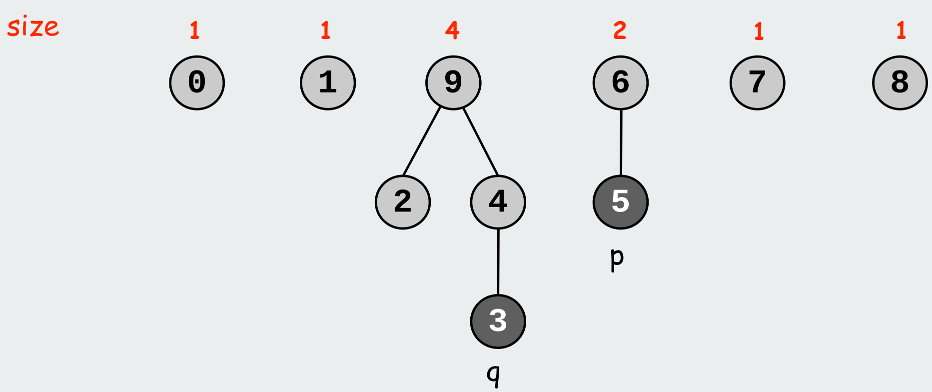
Improvement 1: Weighting

Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each component.
- Balance by linking small tree below large one.

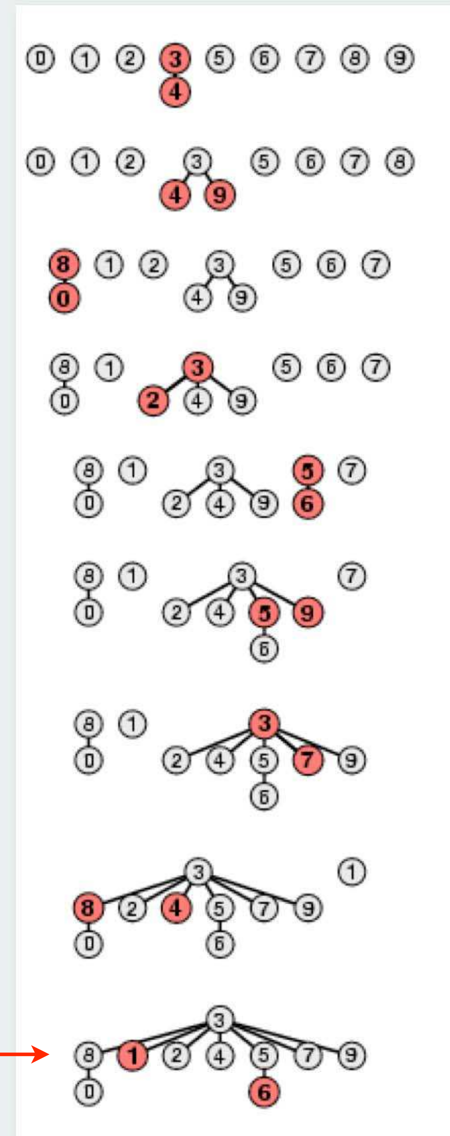
Ex. Union of 5 and 3.

- Quick union: link 9 to 6.
- Weighted quick union: link 6 to 9.



Weighted quick-union example

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



no problem: trees stay flat →

Weighted quick-union: Java implementation

Java implementation.

- Almost identical to quick-union.
- Maintain extra array `sz[]` to count number of elements in the tree rooted at i.

Find. Identical to quick-union.

Union. Modify quick-union to

- merge smaller tree into larger tree
- update the `sz[]` array.

```
if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else sz[i] < sz[j]  { id[j] = i; sz[i] += sz[j]; }
```

Weighted quick-union analysis

Analysis.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.
- Fact: depth is at most lg N. [needs proof]

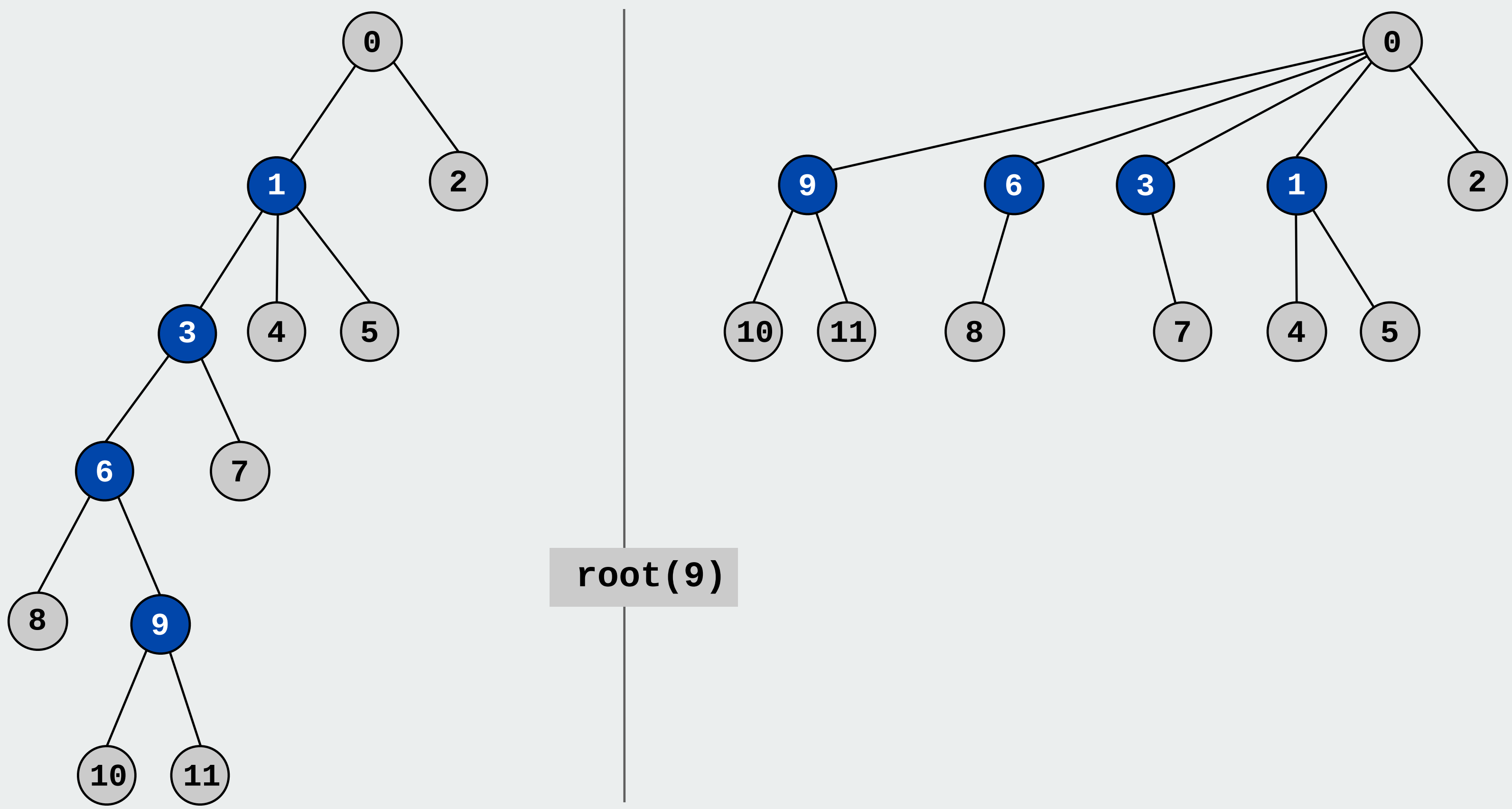
Data Structure	Union	Find
Quick-find	N	1
Quick-union	N *	N
Weighted QU	lg N *	lg N

* includes cost of find

Stop at guaranteed acceptable performance? No, easy to improve further.

Improvement 2: Path compression

Path compression. Just after computing the root of `i`, set the `id` of each examined node to `root(i)`.



Weighted quick-union with path compression

Path compression.

- Standard implementation: add second loop to `root()` to set the id of each examined node to the root.
- Simpler one-pass variant: make every other node in path point to its grandparent.

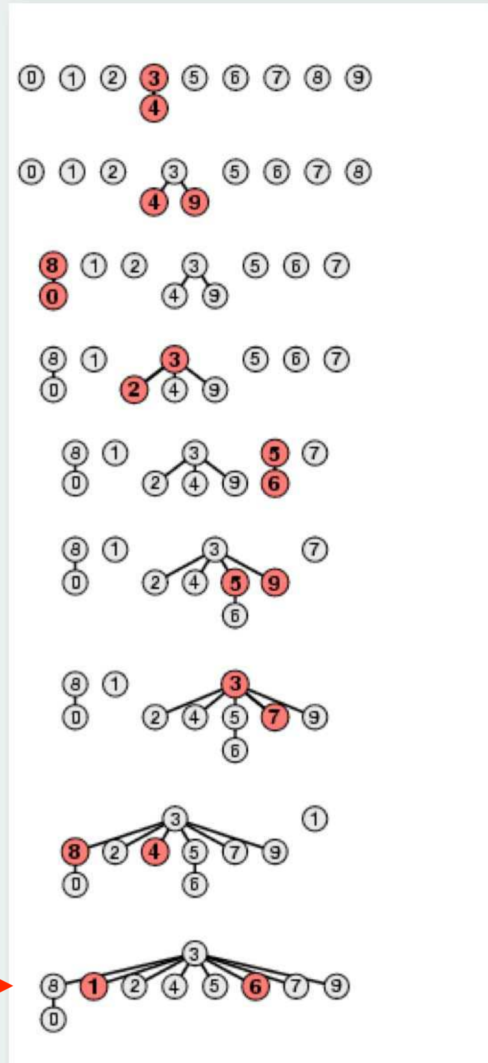
```
public int root(int i)
{
   while (i != id[i])
   {
      id[i] = id[id[i]];
      i = id[i];
   }
   return i;
}
```

only one extra line of code !

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

Weighted quick-union with path compression

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



no problem: trees stay VERY flat

WQUPC performance

Theorem. Starting from an empty data structure, any sequence of M union and find operations on N objects takes O(N + M lg* N) time.

- Proof is very difficult.
- But the algorithm is still simple!

number of times needed to take the lg of a number until reaching 1

Linear algorithm?

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

because lg* N is a constant in this universe

N	lg* N
1	0
2	1
4	2
16	3
65536	4
265536	5

Amazing fact:

- In theory, no linear linking strategy exists

Summary

Algorithm	Worst-case time
Quick-find	M N
Quick-union	M N
Weighted QU	N + M log N
Path compression	N + M log N
Weighted + path	(M + N) lg* N

M union-find ops on a set of N objects

Ex. Huge practical problem.

- 10^{10} edges connecting 10^{9} nodes.
- WQUPC reduces time from 3,000 years to 1 minute.
- Supercomputer won't help much.
- Good algorithm makes solution possible.

WQUPC on Java cell phone beats QF on supercomputer!

Bottom line.

WQUPC makes it possible to solve problems that could not otherwise be addressed

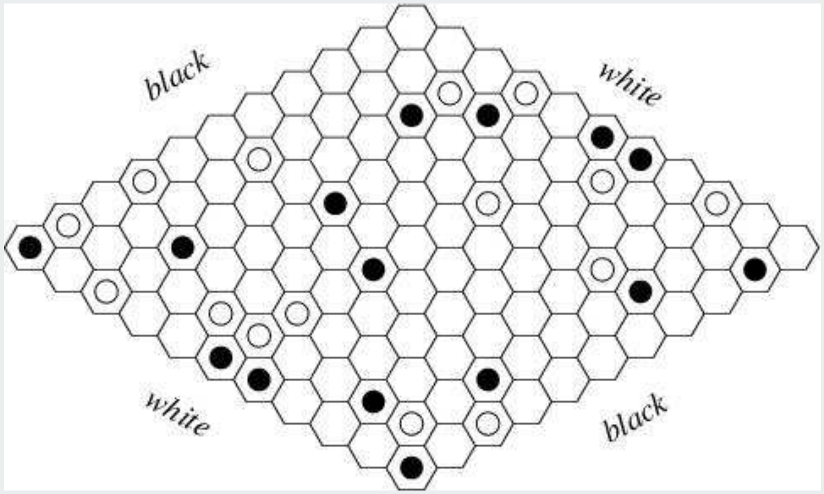
Union-find applications

- ✓ Network connectivity.
- Percolation.
- Image processing.
- Least common ancestor.
- Equivalence of finite state automata.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Games (Go, Hex)
- Compiling equivalence statements in Fortran.

Hex

Hex. [Piet Hein 1942, John Nash 1948, Parker Brothers 1962]

- Two players alternate in picking a cell in a hex grid.
- Black: make a black path from upper left to lower right.
- White: make a white path from lower left to upper right.



Reference: http://mathworld.wolfram.com/GameofHex.html

Union-find application. Algorithm to detect when a player has won.