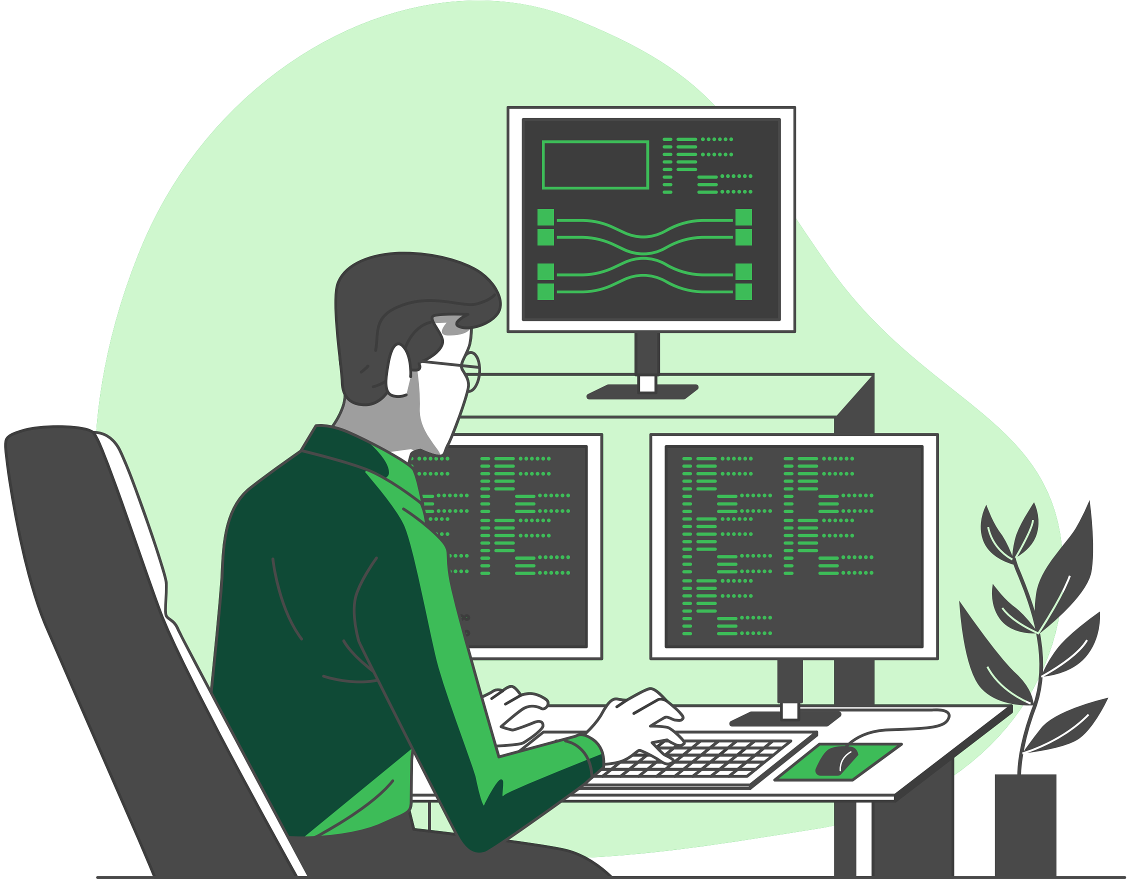
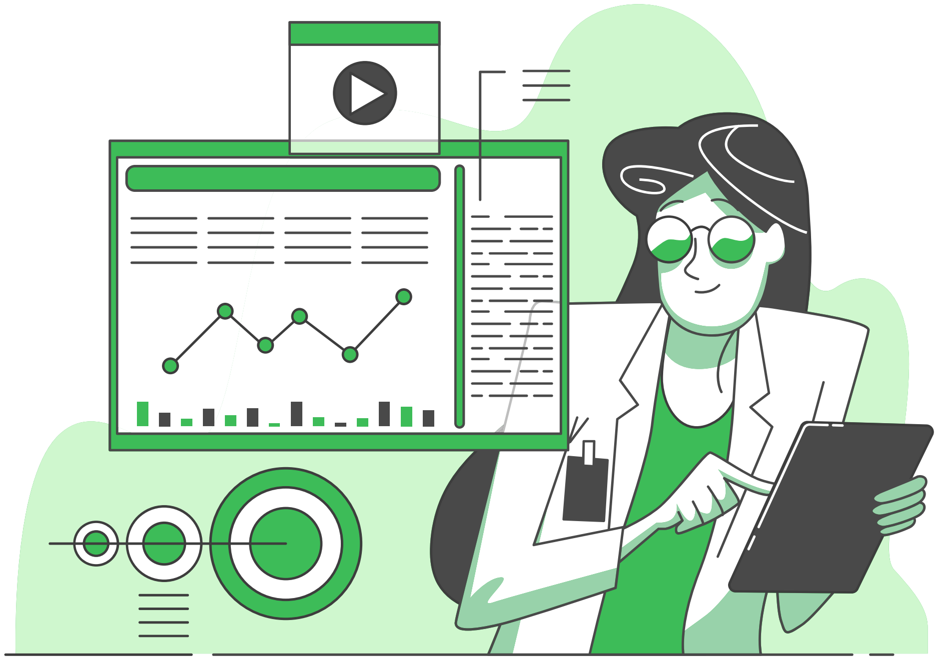


Union–Find

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents



01

Union-Find

Solving the dynamic connectivity problem

Dynamic Connectivity

Input: a sequence of pairs of integers.

A pair (p, q) means "p is connected to q" where connectivity is:

- **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.

Connectivity is an equivalence relation, which can separate objects into equivalence classes (i.e., here, two objects are in the same equivalence class if and only if they are connected.)

Goal: Filter out extraneous pairs (i.e., ignore any pair (p, q) where p and q are **ALREADY** in the same equivalence class)

Some Applications

01 Computer Networking

Determine if two computer are connected.

02 Variable Equivalency

Determine if two variables refer to the same object.

03 Sets (Mathematics)

If p and q are connected, then they are in the same set.

04 Graph Connectivity

A trajectory between a pair of vertices in a graph.

Union–Find

- Also knows as the **Disjoint Set**.
- Stores a collection of **disjoint** sets.
- Provides operations for **adding** new sets, **merging** sets, and **finding** a representative member of a set.

Some Conventions

- The Disjoint Set maintains an array called id that keeps track of the component of each vertex. (i.e., if $\mathrm{id}[i] = 4$, then vertex i is in the component labeled 4).
- Initially, each vertex is its own component. So, $\mathrm{id}[i] = i, \forall i$.
- Maintain a count of the number of components. That is, the starting number of components is $n = |V|$.

02

Quick-Find

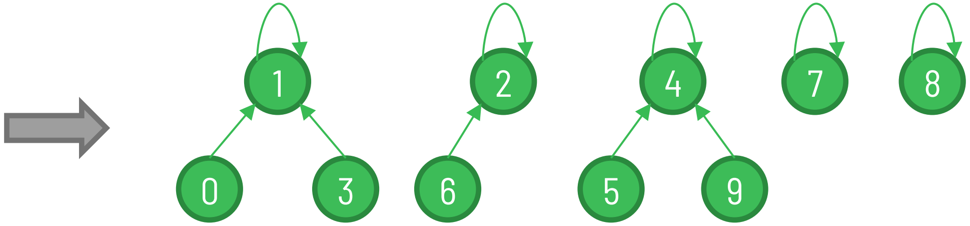
Let’s prioritize the find function

Quick-Find

```
UF(n)
   count ← n
   for i from 0 to n-1 do
      id[i] ← i
   end for

function union(p:item, q:item)
   exit if id[p] = id[q]
   idP ← id[p]
   idQ ← id[q]
   for i from 0 to n-1 do
      if id[i] = idP then
         id[i] ← idQ
      end if
   end for
   count ← count – 1
end function

function find(p:item)
   return id[p]
end function

function connected(p:item, q:item)
   return id[p] = id[q]
end function

function count()
   return count
End function
```

Quick-Find: (6, 2), (9, 5), (3, 0), (9, 4), (3, 1)

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



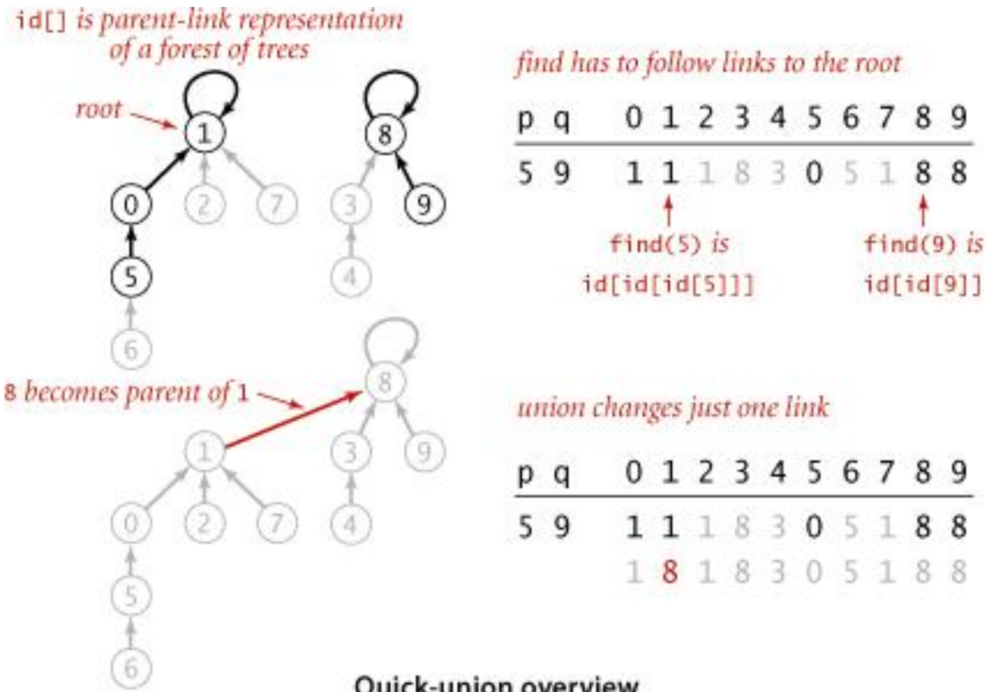
03

Quick-Union

Let’s prioritize the union function

Quick-Union

- id is set up like a tree so that $\mathrm{id}[i]$ gives you its parent, and so on, until you get to a value that points to itself (the root).
- Only one update is needed to union two sets but how many items are checked to find the root?



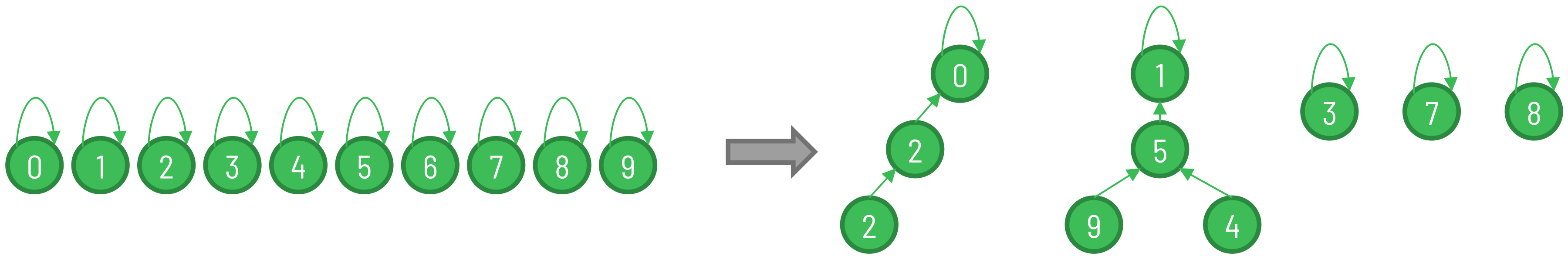
Quick-union overview

Quick-Union

```
UF(n):
   count ← n
   for i from 0 to n-1 do
      id[i] ← i
   end for

function union(p:item, q:item)
   idP ← find(p)
   idQ ← find(q)
   exit if idP = idQ
   id[idP] ← idQ
   count ← count – 1
end function

function find(p:item)
   while p ≠ id[p] do
      p ← id[p]
   end while
   return p
end function

function connected(p:item, q:item)
   return find(p) = find(q)
end function

function count()
   return count
end function
```

Quick-Union: (6, 2), (9, 5), (2, 0), (4, 9), (5, 1)

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



About Quick-Union

- As connections are added, you get fewer but larger trees (correspond to components).
- If the runtime of key operations depends on the height of the tree, what is the worst case?

04

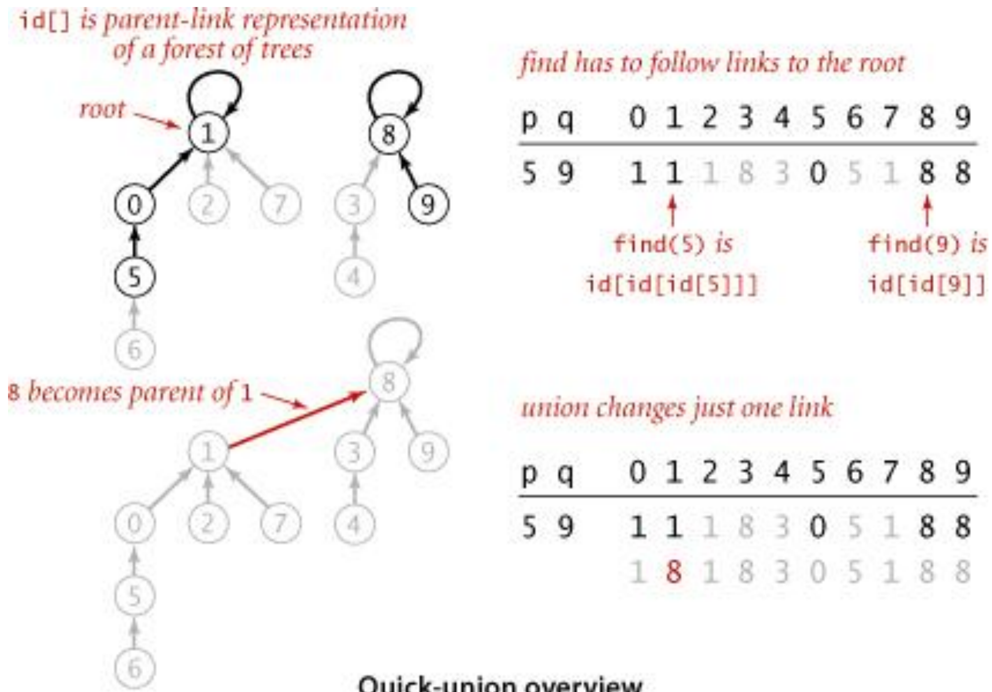
Improvements

Weighted Q-U and Path Compression

Remember Quick-Union

- id is set up like a tree so that $\mathrm{id}[i]$ gives you its parent, and so on, until you get to a value that points to itself (the root).

- Only one update is needed to union two sets but how many items are checked to find the root?

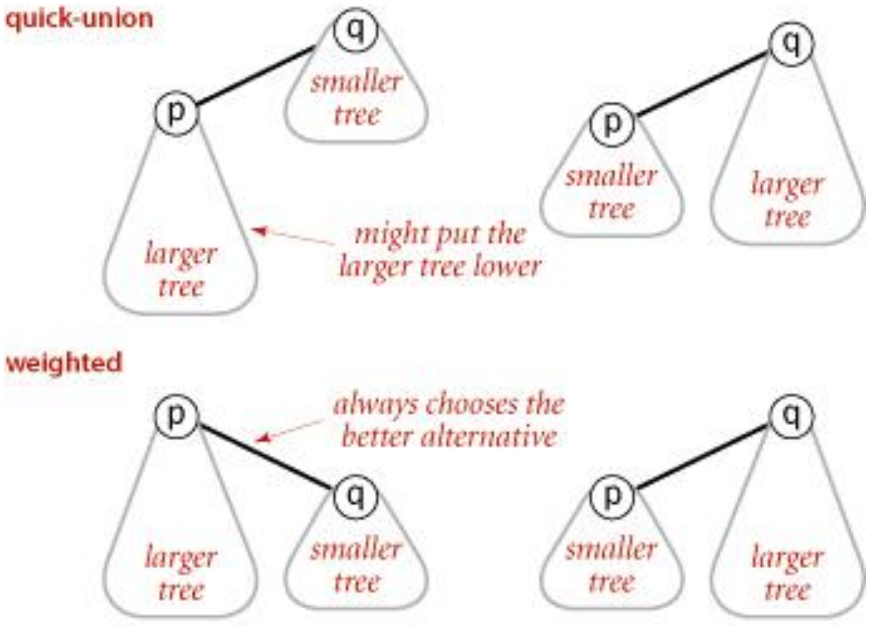


Quick-union overview

About Quick-Union

- As connections are added, you get fewer but larger trees (correspond to components).
- If the runtime of key operations depends on the height of the tree, what is the worst case?

Weighted Quick-Union



Weighted quick-union

Algorithms, 4th edition, Sedgewick and Wayne. Official web site

Weighted Quick-Union

```
UF(n:ℤ⁺)
   count ← n
   for i from 0 to n-1 do
      id[i] ← i
      size[i] ← 1
   end for

function union(p:item, q:item)
   idP ← find(p)
   idQ ← find(q)
   exit if idP = idQ
   if size[idP] < size[idQ] then
      id[idP] ← idQ
      size[idQ] ← size[idQ] + size[idP]
   else
      id[idQ] ← idP
      size[idP] ← size[idP] + size[idQ]
   end if
   count ← count – 1
end function
```

Same as Quick-Union:

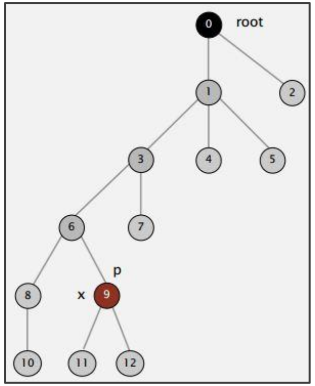
```
find(p)
connected(p, q)
count()
```

Path Compression

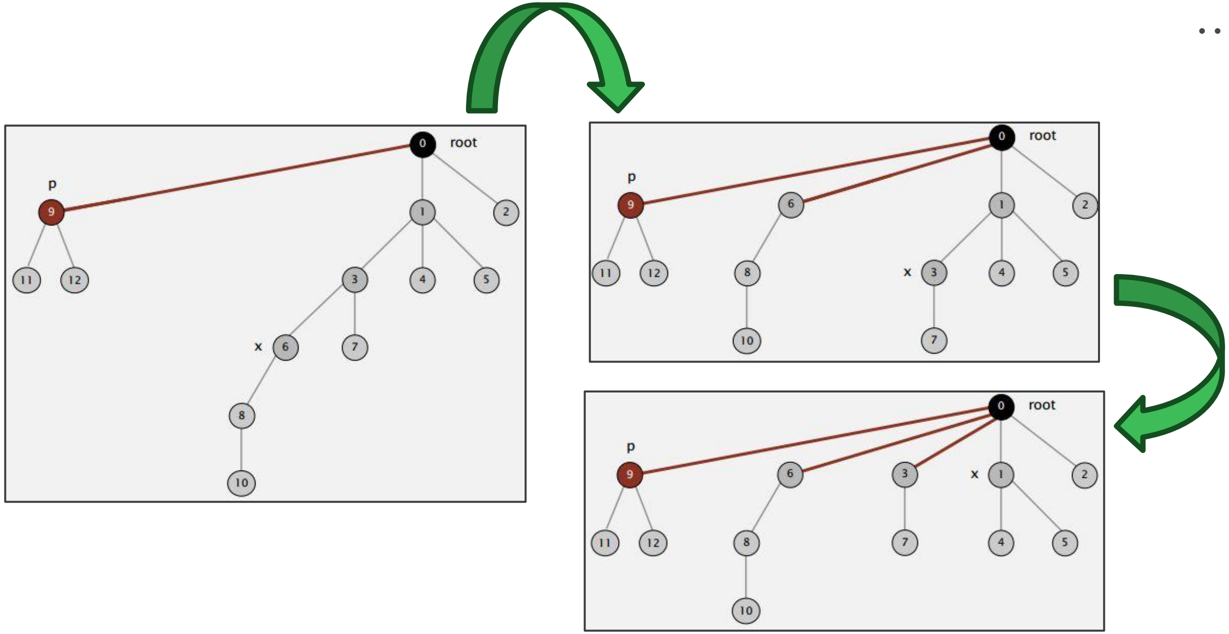
- **Main Idea:** Ideally, we want every node to link directly to its root.
- **How?** After we find the root of p, update the root for every element between p and the root (i.e., elements in the branch connect immediately to the root).
- But…it's expensive to change all the elements (remember Quick-Find?)
- **Solution:** Change the elements you examine as you look for the root. We can do this in multiple ways (e.g., recursion, memoization).
- What's the runtime?

Path Compression Goal

Find(9) returns 0



Set to 0 the id of all nodes in the path from 9 to 0



https://researchhubs.com/post/computing/algorithm-1/quick-union-improvement-compressing.html

Recursive Path Compression

```
function find(p:item)
   if p = id[p] then
      return p
   end if
   id[p] ← find(id[p])
   return id[p]
end function
```

Same as [Weighted] Quick-Union:

```
UF(n)
union(p, q)
connected(p, q)
count()
```

Iterative Path Compression

```
function find(p:item)
   idP ← p
   while idP ≠ id[idP] do
      idP ← id[idP]
   end while
   while p ≠ idP do
      t ← id[p]
      id[p] ← idP
      p ← t
   end while
   return idP
end function
```

Same as [Weighted] Quick-Union:

```
UF(n)
union(p, q)
connected(p, q)
count()
```

Finish

Do you have any questions?

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik and illustrations by Stories