Disjoint Sets
We have a set of objects. We know some are equivalent to each other. Equivalence is

- **Reflexive** if $a$ is equiv. to $a$
- **Symmetric** if $a$ is equiv. to $b$, $b$ is equiv. to $a$
- **Transitive** if $a$ is equiv. to $b$, and $b$ is equiv. to $c$, $a$ is equivalent to $c$

Goal: For any two objects, are they equivalent?
Electrical circuit  If two solder points are connected by a wire, ignoring the wire’s resistance, the two points have the same voltage.

Roads  If two locations are connected by a road, you can travel between them.

Substitutes  If product A does a task and product B does the same task, they are substitutes.
Equivalence Sets

Given a set of equivalences, this partitions the elements into equivalence sets.
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

\[
\text{union}(a,b) \quad \text{make a equivalent to b}
\]

\[
\text{find}(a) \quad \text{return some indicator of a’s set}
\]
Need to answer a sequence of queries, as quickly as possible:

- union(a,b) make a equivalent to b
- find(a) return some indicator of a’s set

Example:
union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)
Need to answer a sequence of queries, as quickly as possible:

- `union(a,b)` make a equivalent to b
- `find(a)` return some indicator of a’s set

Example:

```
union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)
```
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

- `union(a,b)` make `a` equivalent to `b`
- `find(a)` return some indicator of `a`’s set

Example:

union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

- \texttt{union(a,b)} make a equivalent to b
- \texttt{find(a)} return some indicator of a’s set

Example:

\texttt{union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)}
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

- `union(a,b)` make a equivalent to b
- `find(a)` return some indicator of a’s set

Example:
union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)

```
  a -- d
     |
   b

  c
```

Shelton (UC Riverside)
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

- `union(a,b)` make a equivalent to b
- `find(a)` return some indicator of a’s set

Example:

union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

- \texttt{union(a,b)} make a equivalent to b
- \texttt{find(a)} return some indicator of a’s set

Example:
union(e,c) find(a) find(c) union(a,d) find(a) **find(d)** union(d,e) find(a) find(c)
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

union(a,b) make a equivalent to b
find(a) return some indicator of a’s set

Example:
union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

- \( \text{union}(a,b) \) make a equivalent to b
- \( \text{find}(a) \) return some indicator of a’s set

Example:

\( \text{union}(e,c) \) \( \text{find}(a) \) \( \text{find}(c) \) \( \text{union}(a,d) \) \( \text{find}(a) \) \( \text{find}(d) \) \( \text{union}(d,e) \) \( \text{find}(a) \) \( \text{find}(c) \)
Dynamic Equivalence

Need to answer a sequence of queries, as quickly as possible:

union(a,b) make a equivalent to b
find(a) return some indicator of a’s set

Example:
union(e,c) find(a) find(c) union(a,d) find(a) find(d) union(d,e) find(a) find(c)
Algorithm Types

Algorithm must be dynamic because the sets change.

Two types of dynamic algorithms:

- **offline**: The sequence is known in advance
- **online**: Each operation is given one at a time and must be answered before continuing
Algorithm must be dynamic because the sets change.

Two types of dynamic algorithms:

- **offline** The sequence is known in advance
- **online** Each operation is given one at a time and must be answered before continuing

We will considering online methods
Value returned by \texttt{find} does not matter, provided it is the same for elements in the same set (and different for elements in different sets).

Element values do not matter. Assume they are 0 through $n-1$. (Hash to values between 0 and $n-1$ if they are not.)
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds,

Running time? $O(n^2 + m)$

Better: keep each set in linked list with size

Change elements in smaller list.

Running time? $O(n \log n + m)$

Why? maximum of $\log n$ changes per element. When element is changed, it is merged with bigger set that doubles the size of the set (at least) can only double $\log n$ times before everything is one set
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

union(4,2) find(0) find(2) union(0,3) find(0) find(3) union(3,4) find(0) find(2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Methods

Should find or union be fast? (cannot both be fast)

find Keep a “name” (letter, number) with each element. Update by scanning through the array.

union(4,2) find(0) find(2) union(0,3) find(0) find(3) union(3,4) find(0) find(2)

A   B   E   D   E
0   1   2   3   4
Should find or union be fast? (cannot both be fast)

**find**  Keep a “name” (letter, number) with each element. Update by scanning through the array.

union(4,2)  **find(0)**  union(0,3)  **find(0)**  find(3)  union(3,4)  **find(0)**  **find(2)**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

union(4,2) find(0) **find(2)** union(0,3) find(0) find(3) union(3,4) find(0) find(2)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>E</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Methods

Should find or union be fast? (cannot both be fast)

- **find**: Keep a “name” (letter, number) with each element. Update by scanning through the array.

  union(4,2) find(0) find(2) union(0,3) find(0) find(3) union(3,4) find(0) find(2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>A</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Given $n$ elements (and $n-1$ unions) and $m$ finds, running time?

- $O(n^2 + m)$
- Better: keep each set in linked list with size
  - Change elements in smaller list.
  - Running time?
    - $O(n \log n + m)$

Why? maximum of $\log n$ changes per element
- When element is changed, it is merged with bigger set
- That doubles the size of the set (at least)
- Can only double $\log n$ times before everything is one set
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

union(4,2) find(0) find(2) union(0,3) **find(0)** find(3) union(3,4) find(0) find(2)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td>A</td>
<td>E</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Given $n$ elements (and $n-1$ unions) and $m$ finds, Running time?

$O(n^2 + m)$

Better: keep each set in linked list with size

Running time?

$O(n \log n + m)$

Why? maximum of $\log n$ changes per element

when element is changed, it is merged with bigger set

that doubles the size of the set (at least)

can only double $\log n$ times before everything is one set
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td>A</td>
<td>E</td>
</tr>
</tbody>
</table>

union(4,2) find(0) find(2) union(0,3) find(0) **find(3)** union(3,4) find(0) find(2)

Given $n$ elements (and $n - 1$ unions) and $m$ finds,

Running time? $O(n^2 + m)$

Better: keep each set in linked list with size $\log n$ Change elements in smaller list.

Running time? $O(n \log n + m)$

Why? maximum of $\log n$ changes per element when element is changed, it is merged with bigger set that doubles the size of the set (at least) can only double $\log n$ times before everything is one set
Should find or union be fast? (cannot both be fast)

- **find**: Keep a “name” (letter, number) with each element. Update by scanning through the array.

  - union(4,2) find(0) find(2) union(0,3) find(0) find(3) **union(3,4)** find(0) find(2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Methods

Should find or union be fast? (cannot both be fast)

find Keep a “name” (letter, number) with each element. Update by scanning through the array.

union(4,2) find(0) find(2) union(0,3) find(0) find(3) union(3,4) find(0) find(2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

union(4,2) find(0) find(2) union(0,3) find(0) find(3) union(3,4) find(0) find(2)
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds, Running time?

$$\mathcal{O}(n^2 + m)$$

Better: keep each set in linked list with size

Change elements in smaller list.

Running time?

$$\mathcal{O}(n \log n + m)$$

Why? maximum of $\log n$ changes per element

when element is changed, it is merged with bigger set

that doubles the size of the set (at least)

can only double $\log n$ times before everything is one set
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds, Running time? $O(n^2 + m)$
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds,

Running time? $O(n^2 + m)$

Better: keep each set in linked list with size
Change elements in smaller list.
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given \( n \) elements (and \( n - 1 \) unions) and \( m \) finds, Running time? \( O(n^2 + m) \)

Better: keep each set in linked list with size
Change elements in smaller list.

union(4,2) union(0,3) union(3,4)

<table>
<thead>
<tr>
<th>elements: (set, next)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,-1)</td>
<td>(B,-1)</td>
<td>(C,-1)</td>
<td>(D,-1)</td>
<td>(E,-1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sets: (head index, tail index, size)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,1)</td>
<td>(1,1,1)</td>
<td>(2,2,1)</td>
<td>(3,3,1)</td>
<td>(4,4,1)</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds, Running time? $O(n^2 + m)$

Better: keep each set in linked list with size
Change elements in smaller list.

**union(4,2)** union(0,3) union(3,4)

<table>
<thead>
<tr>
<th>elements: (set, next)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,-1)</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sets: (head index, tail index, size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,1)</td>
</tr>
</tbody>
</table>

Running time? $O(n \log n + m)$

Why? maximum of $\log n$ changes per element
When element is changed, it is merged with bigger set that doubles the size of the set (at least)

can only double $\log n$ times before everything is one set
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given \( n \) elements (and \( n - 1 \) unions) and \( m \) finds, Running time? \( O(n^2 + m) \)

Better: keep each set in linked list with size
Change elements in smaller list.

union(4,2) union(0,3) union(3,4)

<table>
<thead>
<tr>
<th>elements: (set, next)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,3)</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sets: (head index, tail index, size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,3,2)</td>
</tr>
</tbody>
</table>

A B C D E

Running time? \( O(n \log n + m) \)

Why? maximum of \( \log n \) changes per element when element is changed, it is merged with bigger set that doubles the size of the set (at least) can only double \( \log n \) times before everything is one set
Methods

Should find or union be fast? (cannot both be fast)

find  Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given \( n \) elements (and \( n - 1 \) unions) and \( m \) finds, Running time?  \( O(n^2 + m) \)

Better: keep each set in linked list with size
Change elements in smaller list.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,3)</td>
<td>(B,-1)</td>
<td>(A,-1)</td>
<td>(A,4)</td>
<td>(A,2)</td>
</tr>
</tbody>
</table>

union(4,2) union(0,3) union(3,4)

elements: (set, next)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,2,4)</td>
<td>(1,1,1)</td>
<td>(-1,-1,0)</td>
<td>(-1,-1,0)</td>
<td>(-1,-1,0)</td>
</tr>
</tbody>
</table>

sets: (head index, tail index, size)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>

Running time?  \( O(n \log n + m) \)
Why? maximum of \( \log n \) changes per element when element is changed, it is merged with bigger set that doubles the size of the set (at least)
can only double \( \log n \) times before everything is one set

Shelton (UC Riverside)  Disjoint Sets  C5 14
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds, Running time? $O(n^2 + m)$

Better: keep each set in linked list with size
Change elements in smaller list.
Running time?
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds,

Running time? $O(n^2 + m)$

Better: keep each set in linked list with size

Change elements in smaller list.

Running time? $O(n \log n + m)$
Methods

Should find or union be fast? (cannot both be fast)

**find** Keep a “name” (letter, number) with each element. Update by scanning through the array.

Given $n$ elements (and $n - 1$ unions) and $m$ finds, Running time? $O(n^2 + m)$

Better: keep each set in linked list with size
Change elements in smaller list.
Running time? $O(n \log n + m)$
Why? maximum of $\log n$ changes per element

- when element is changed, it is merged with bigger set
- that doubles the size of the set (at least)
- can only double $\log n$ times before everything is one set
Methods

Should find or union be fast? (cannot both be fast)

union Represent each set as a tree.
Use root of the tree as the representative value.
Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree.
Use root of the tree as the representative value.
To merge two trees, make the root of one tree the child of the other.
Use only parent pointers.
Methods

Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree. Use root of the tree as the representative value. To merge two trees, make the root of one tree the child of the other. Use only parent pointers.

union(4,2) union(0,3) union(3,4)
Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree.
Use root of the tree as the representative value.
To merge two trees, make the root of one tree the child of the other.
Use only parent pointers.

union(4,2) union(0,3) union(3,4)
Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree.
Use root of the tree as the representative value.
To merge two trees, make the root of one tree the child of the other.
Use only parent pointers.

union(4,2) **union(0,3)** union(3,4)
Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree.
Use root of the tree as the representative value.
To merge two trees, make the root of one tree the child of the other.
Use only parent pointers.
union(4,2) union(0,3) union(3,4)
Methods

Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree.
Use root of the tree as the representative value.
To merge two trees, make the root of one tree the child of the other.
Use only parent pointers.

union(4,2) union(0,3) union(3,4)

```
      0
    /   |
   1-1   2
  /     |
 3-1   4
```

Shelton (UC Riverside)

Disjoint Sets
Methods

Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree.
Use root of the tree as the representative value.
To merge two trees, make the root of one tree the child of the other.
Use only parent pointers.

**union(4,2) union(0,3) union(3,4)**
Methods

Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree. Use root of the tree as the representative value. To merge two trees, make the root of one tree the child of the other. Use only parent pointers.

union(4,2)  union(0,3)  union(3,4)

```
0 1 2 3 4
-1 -1 -1 0 2
0 1 2 3 4
```
Should find or union be fast? (cannot both be fast)

**union** Represent each set as a tree.
Use root of the tree as the representative value.
To merge two trees, make the root of one tree the child of the other.
Use only parent pointers.
union(4,2) union(0,3) union(3,4)

\[
\begin{array}{c|c|c|c|c}
0 & 1 & 2 & 3 & 4 \\
\hline
2 & -1 & -1 & 0 & 2 \\
\end{array}
\]
Running time?

**union** [roots are given]:

\[ O(1) \]

**find**

\[ O(\text{height of tree}) \]

\[ = O(n) \]

**union** [general]

\[ O(n) \]

Total running time of \( n - 1 \) unions and \( m \) finds?

\[ O(n^2 + mn) \]
Disjoint Set Union Find

Running time?
union [roots are given]: $O(1)$

Problem: Trees are too tall
Disjoint Set Union Find

Running time?

union [roots are given]: \( O(1) \)

find \( O(\text{height of tree}) \)

```
2 1 2 3 4
0 1 -1 2 0 2
```
Running time?

union [roots are given]: \( O(1) \)
find \( O(\text{height of tree}) = O(n) \)
union [general] \( O(n) \)
Disjoint Set Union Find

Running time?

union [roots are given]: $O(1)$
find $O($height of tree$) = O(n)$
union [general] $O(n)$

total running time of $n - 1$ unions and $m$ finds?

Problem: Trees are too tall

Shelton (UC Riverside)
Disjoint Set Union Find

Running time?

union [roots are given]: $O(1)$

find $O(\text{height of tree}) = O(n)$

union [general] $O(n)$

total running time of $n-1$ unions and $m$ finds? $O(n^2 + mn)$

Problem: Trees are too tall
Disjoint Set Union Find

Running time?

- union [roots are given]: $O(1)$
- find: $O(\text{height of tree}) = O(n)$
- union [general]: $O(n)$

Total running time of $n - 1$ unions and $m$ finds? $O(n^2 + mn)$

Problem: Trees are too tall
Disjoint Set Union Find

Solution: Merge the smaller one onto the larger one (need to keep tree size)

Example:
union(6,3) union(5,3) union(7,1) union(1,3)
Disjoint Set Union Find

Solution: Merge the smaller one onto the larger one (need to keep tree size)
Example:
union(6,3) union(5,3) union(7,1) union(1,3)
Disjoint Set Union Find

Solution: Merge the smaller one onto the larger one (need to keep tree size)

Example:
union(6,3) union(5,3) union(7,1) union(1,3)
Disjoint Set Union Find

Solution: Merge the smaller one onto the larger one (need to keep tree size)

Example:
union(6,3) union(5,3) union(7,1) union(1,3)
Disjoint Set Union Find

Solution: Merge the smaller one onto the larger one (need to keep tree size)
Example:

union(6,3) union(5,3) **union(7,1)** union(1,3)

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

```
<table>
<thead>
<tr>
<th>-1</th>
<th>-2</th>
<th>-1</th>
<th>-3</th>
<th>-1</th>
<th>3</th>
<th>3</th>
<th>1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
```
Disjoint Set Union Find

Solution: Merge the smaller one onto the larger one (need to keep tree size)

Example:
union(6,3) union(5,3) union(7,1) union(1,3)
Disjoint Set Union Find

Running time?

union [roots are given]:

\[ O(1) \]

find:

\[ O(\text{height of tree}) = O(\log n) \]

Why? Number of links in path to root is number of times this node was a member of the smaller set in a union. Can happen a maximum of \[ \log n \] times.

union [general]:

\[ O(\log n) \]

total running time of \( n - 1 \) unions and \( m \) finds:

\[ O(n \log n + m \log n) \]

Not bad, but we had \[ O(n \log n + m \log n) \] before!
Disjoint Set Union Find

Running time?

\begin{align*}
\text{union } \text{[roots are given]} & : \mathcal{O}(1) \\
\text{find} & : \mathcal{O}(\text{height of tree}) = \mathcal{O}(\log n)
\end{align*}

Why? Number of links in path to root is number of times this node was a member of the smaller set in a union. Can happen a maximum of \(\log n\) times.

\begin{align*}
\text{union } \text{[general]} & : \mathcal{O}(\log n) \\
\text{total running time of } n - 1 \text{ unions and } m \text{ finds} & : \mathcal{O}(n \log n + m \log n)
\end{align*}

Not bad, but we had \(\mathcal{O}(n \log n + m \log n)\) before!
Disjoint Set Union Find

Running time?

union [roots are given]: $O(1)$

find $O(\text{height of tree})$
Running time?

**union** [roots are given]: $O(1)$

**find** $O(\text{height of tree}) = O(\log n)$

Why? Number of links in path to root is number of times this node was a member of the smaller set in a union. Can happen a maximum of $\log n$ times.

**union** [general] $O(\log n)$
Disjoint Set Union Find

Running time?

\textbf{union} [roots are given]: \(O(1)\)

\textbf{find} \(O(\text{height of tree}) = O(\log n)\)

Why? Number of links in path to root is number of times this node was a member of the smaller set in a union. Can happen a maximum of \(\log n\) times.

\textbf{union} [general] \(O(\log n)\)

total running time of \(n - 1\) unions and \(m\) finds?
Disjoint Set Union Find

Running time?

union [roots are given]: $O(1)$

find $O(\text{height of tree}) = O(\log n)$

Why? Number of links in path to root is number of times this node was a member of the smaller set in a union.

Can happen a maximum of $\log n$ times.

union [general] $O(\log n)$

total running time of $n - 1$ unions and $m$ finds? $O(n \log n + m \log n)$

Not bad, but we had $O(n \log n + m)$ before!
Disjoint Set Union Find

Last trick: Path Compression
When executing find, adjust pointers along path to point to root.
Last trick: **Path Compression**

When executing `find`, adjust pointers along path to point to root.

Example:

`join(3,11)`
Disjoint Set Union Find

Last trick: **Path Compression**
When executing `find`, adjust pointers along path to point to root.
Example:
```
join(3, 11) ⇒ find(3) → 3, find(11) → 8, join(8, 3)
```
Disjoint Set Union Find

Last trick: **Path Compression**
When executing `find`, adjust pointers along path to point to root.
Example:

`join(3,11) ⇒ find(3) → 3, find(11) → 8, join(8,3)`

without path compression
Last trick: **Path Compression**
When executing `find`, adjust pointers along path to point to root.
Example:

\[ \text{join}(3,11) \Rightarrow \text{find}(3) \rightarrow 3, \quad \text{find}(11) \rightarrow 8, \quad \text{join}(8,3) \]

without path compression
Last trick: **Path Compression**
When executing `find`, adjust pointers along path to point to root.

Example:
\[\text{join}(3,11) \Rightarrow \text{find}(3) \rightarrow 3, \text{find}(11) \rightarrow 8, \text{join}(8,3)\]
without path compression
Last trick: **Path Compression**

When executing \texttt{find}, adjust pointers along path to point to root.

Example:

join(3,11) $\Rightarrow$ \texttt{find}(3) $\rightarrow$ 3, \texttt{find}(11) $\rightarrow$ 8, join(8,3)

with path compression
Last trick: **Path Compression**
When executing `find`, adjust pointers along path to point to root.

Example:
`join(3,11) ⇒ find(3) → 3, find(11) → 8, join(8,3)`

with path compression
Disjoint Set Union Find

Last trick: **Path Compression**
When executing `find`, adjust pointers along path to point to root.
Example:

join(3,11)  ⇒  find(3) → 3, find(11) → 8, join(8,3)
with path compression
Disjoint Set Union Find

Running time?

\[ f(n) \text{ is a function of } n \text{ that reduces } n. \]

\[ f^*(n) \text{ is the number of times you need to apply } f(n) \text{ to } n \text{ to get } 1: \]

\[ f(n) = n - 2^{n/2} \sqrt{n} \log n \log^* n \]

\[ f^*(n) = n/2 \log n \log \log n \log \log \log n \ldots \]

\[ \log^* n \text{ is the iterated logarithm and is defined as above (similarly for } \log^* n, \log^*^* n, \ldots \). \]
Running time?
Complicated. Need to first explain slow-growing functions:

\( f(n) \) is a function of \( n \) that reduces \( n \). \( f^*(n) \) is the number of times you need to apply \( f(n) \) to \( n \) to get 1:

\[
\begin{array}{c|ccccccc}
  f(n) & n - 2 & n/2 & \sqrt{n} & \log n & \log^* n \\
  f^*(n) & n/2 & \log n & \log \log n & \log^* n & \log^{**} n \\
\end{array}
\]

\( \log^* n \) is the iterated logarithm and is defined as above (similarly for \( \log^{**} n, \log^{***} n, \ldots \)).
Running time?
Section 8.6.2 shows a proof of $O(m + n \log \log n)$.

Section 8.6.3 betters that to $O(m + n \log^* n)$.

Section 8.6.4 betters that to $O(m \alpha(m, n))$
where $\alpha(m, n)$ is the inverse of the Ackermann function, $A(m, n)$

$A(m, n)$ grows absurdly quickly (and is hard to define other than with complicated generalization of exponentiation). Therefore, $\alpha(m, n)$ grows absurdly slowly. It can be defined as the minimum number of stars in $\log^{\cdots\cdots}(\log n)$ such that the quantity is less than $m/n$.

For all values of $m$ and $n$ you could possibly construct in this universe, it is less than 5.