Have you heard of tetration?

It’s the fourth hyperoperation.
Have you heard of tetration?

It’s the fourth hyperoperation.

\[ b + n \triangleq b + 1 + 1 + \cdots + 1 \]

Exponentiation associates to the right, so for example \( b^b^b \) means \( b \times b \times \cdots \times b \), not \( ((b^b)^b)^b \). Why?
Have you heard of tetration?

It’s the fourth hyperoperation.

\[
\begin{align*}
    b + n & \triangleq b + 1 + 1 + \cdots + 1 \\
    b \times n & \triangleq b + b + \cdots + b
\end{align*}
\]

Exponentiation associates to the right, so for example \( 4^b \) means \( b \times (b \times b) \), not \((b \times b) \times b\). Why?

Which is bigger?

1
Have you heard of tetration?

It’s the fourth hyperoperation.

\[
\begin{align*}
  b + n & \triangleq b + 1 + 1 + \cdots + 1 \\
  b \times n & \triangleq b + b + \cdots + b \\
  b^n & \triangleq b \times b \times \cdots \times b
\end{align*}
\]
Have you heard of tetration?

It’s the fourth hyperoperation.

\[
\begin{align*}
b + n & \triangleq b + 1 + 1 + \cdots + 1 \\
b \times n & \triangleq b + b + \cdots + b \\
b^n & \triangleq b \times b \times \cdots \times b \\
nb & \triangleq b b \cdots b
\end{align*}
\]
Have you heard of tetration?

It’s the fourth hyperoperation.

\[
\begin{align*}
  b + n & \triangleq b + 1 + 1 + \cdots + 1 \\
  b \times n & \triangleq b + b + \cdots + b \\
  b^n & \triangleq b \times b \times \cdots \times b \\
  n b & \triangleq b^{b^{\cdots b}} \\
\end{align*}
\]

Exponentiation associates to the right, so for example \(4 b\) means \(b^{b^{b^{b}}}}\), not \((b^b)^b\).
Have you heard of tetration?

It’s the fourth hyperoperation.

\[
\begin{align*}
    b + n & \triangleq b + 1 + 1 + \cdots + 1 \\
    b \times n & \triangleq b + b + \cdots + b \\
    b^n & \triangleq b \times b \times \cdots \times b \\
    n b & \triangleq b^{b^{\cdots^{b}}}_n
\end{align*}
\]

Exponentiation associates to the right, so for example \(4b\) means \(b^{b^{b_{b^{b}}}}\), not \((b^{b^{b}})^{b}\). Why?
Have you heard of tetration?

It’s the fourth hyperoperation.

\[
\begin{align*}
\text{Exponentiation } & \text{ associates to the right, so for example } 4^b \text{ means } b^{b^{b^{b\ldots}}}, \text{ not } (b^b)^b. \text{ Why? Which is bigger?}
\end{align*}
\]
Q: How fast does it grow?

A: Real fast.

Q: Does it have an inverse?

A: Yeah, 2—which do you want?

Q: Is one better than the other?

A: Maybe, you decide:

\[
\begin{align*}
b^n &= a \\
a &= b^\frac{n}{\log_b n}
\end{align*}
\]

Q: Why should we care?

A: Its inverse grows as slow as its self grows fast, and tetration grows real fast.
Tetration FAQ

Tetration FAQ

Q: Does it have an inverse?
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?

If
\[ b^n = a \]
then
\[ b = \sqrt[n]{a} \]
and also
\[ b^n = a \]
\[ b = \frac{a}{n} \]
\[ b^n = a \]
\[ b = \log_a n \]
Tetration FAQ

Q: How fast does it grow?  
A: Real fast.

Q: Does it have an inverse?  
A: Yeah, 2—which do you want?

Q: Is one better than the other?
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: Maybe, you decide:

If \( \ldots \) then \( \ldots \), and also \( \ldots \):

\[
\begin{align*}
    b + n &= a \\
    b \times n &= a \\
    b^n &= a \\
    n^b &= a
\end{align*}
\]
Tetration FAQ

Q: Does it have an inverse? A: Yeah, 2—which do you want?
Q: Is one better than the other? A: Maybe, you decide:

If \( b + n = a \) then \( b = a - n \), and also \( n = a - b \)
\( b \times n = a \)
\( b^n = a \)
\( n^b = a \)

Q: Why should we care? A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

\[
\begin{array}{c|c|c|c}
n & 2 & 4 & 16 \\
\hline
2 & 2 & 2 & 2 \\
536 & 536 & 536 & 536 \\
\approx & 2 & 2 & 2 \\
19,728 & 19,728 & 19,728 & 19,728 \\
\end{array}
\]
Tetration FAQ


Q: Does it have an inverse?  A: Yeah, 2—which do you want?

Q: Is one better than the other?  A: Maybe, you decide:

If \( \ldots \) then \( \ldots \), and also \( \ldots \).

\[
\begin{align*}
& b + n = a & & b = a - n & & n = a - b \\
& b \times n = a & & b = a/n & & n = a/b \\
& b^n = a & & n^b = a \\
\end{align*}
\]
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: Maybe, you decide:

If ... then ..., and also . . . .

\begin{align*}
    b + n &= a \\
    b \times n &= a \\
    b^n &= a \\
    n^b &= a
\end{align*}

\begin{align*}
    b &= a - n \\
    b &= a/n \\
    b &= \sqrt[n]{a} \\
    n &= \log_b a
\end{align*}

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

\begin{align*}
    a &= 2 \quad \text{for } b = 2 \\
    a &= 536 \quad \text{for } b = 2
\end{align*}
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: Maybe, you decide:

\[
\begin{align*}
\text{If} & \quad \ldots \quad \text{then} \quad \ldots \quad , \quad \text{and also} \quad \ldots \\
\quad b + n &= a & \quad b &= a - n & \quad n &= a - b \\
\quad b \times n &= a & \quad b &= a/n & \quad n &= a/b \\
\quad b^n &= a & \quad b &= \sqrt[n]{a} & \quad n &= \log_b a \\
\quad n b &= a & \quad b &= \sqrt[n]{a} & \quad a &= n b
\end{align*}
\]

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?   A: Maybe, you decide:

If $b + n = a$ then $b = a - n$, and also

\begin{align*}
  b + n &= a & b &= a - n & n &= a - b \\
  b \times n &= a & b &= a/n & n &= a/b \\
  b^n &= a & b &= \sqrt[n]{a} & n &= \log_b a \\
  n \times b &= a & b &= \sqrt[n]{a} & n &= \log_b^* a
\end{align*}

Q: Why should we care?   A: Its inverse grows as slow as its self grows fast, and tetration grows real fast: $n^2, 12, 4, 16, 65, 536, 265, 536 \approx 22.0035 \times 10^{192}$.
Q: How fast does it grow?  
A: Real fast.

Q: Does it have an inverse?  
A: Yeah, 2—which do you want?

Q: Is one better than the other?  
A: Maybe, you decide:

If \( \ldots \) then \( \ldots \), and also \( \ldots \).

\[
\begin{align*}
  b + n &= a & b = a - n & n = a - b \\
  b \times n &= a & b = a/n & n = a/b \\
  b^n &= a & b = \sqrt[n]{a} & n = \log_b a \\
  n b &= a & b = \sqrt[n]{a} & n = \log_b^* a
\end{align*}
\]
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: Maybe, you decide:

If . . .  then . . . ,  and also . . .
\[ b + n = a \quad b = a - n \quad n = a - b \]
\[ b \times n = a \quad b = a/n \quad n = a/b \]
\[ b^n = a \quad b = n^{1/3} a \quad n = \log_3^a \]
\[ n^b = a \quad b = n^{1/4} a \quad n = \log_b^a \]
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: Maybe, you decide:

If . . .          then . . . ,  and also . . .
\[ b + n = a \quad b = \sqrt[n]{a_1} \quad n = \log_b^1 a \]
\[ b \times n = a \quad b = \sqrt[n]{a_2} \quad n = \log_b^2 a \]
\[ b^n = a \quad b = \sqrt[n]{a_3} \quad n = \log_b^3 a \]
\[ n^b = a \quad b = \sqrt[n]{a_4} \quad n = \log_b^4 a \]

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

\[ \begin{array}{c|c|c}
\hline
n & 2 & 4 \\
\hline
2 & 1 & 2 \\
4 & 16 & 65,536 \\
\hline
\end{array} \]

\[ n \approx 2^{10^{19}} \times 10^{19} \]
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: Maybe, you decide:

\[
\begin{align*}
\text{If} & \quad \text{then} \ldots, \quad \text{and also} \ldots \\
1 + n &= a & b &= \sqrt[n]{a} & n &= \log_b^0 a \\
b + n &= a & b &= \sqrt[n]{a} & n &= \log_b^1 a \\
b \times n &= a & b &= \sqrt[n]{a} & n &= \log_b^2 a \\
b^n &= a & b &= \sqrt[n]{a} & n &= \log_b^3 a \\
nb &= a & b &= \sqrt[n]{a} & n &= \log_b^4 a
\end{align*}
\]

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:
Tetration FAQ

Q: Does it have an inverse? A: Yeah, 2—which do you want?
Q: Is one better than the other? A: Maybe, you decide:

If \( \ldots \) then \( \ldots \), and also \( \ldots \).

\[
\begin{align*}
H_0(b, n) &= a & b &= \sqrt[n]{a_0} & n &= \log_0^0 a \\
H_1(b, n) &= a & b &= \sqrt[n]{a_1} & n &= \log_1^1 a \\
H_2(b, n) &= a & b &= \sqrt[n]{a_2} & n &= \log_2^2 a \\
H_3(b, n) &= a & b &= \sqrt[n]{a_3} & n &= \log_3^3 a \\
H_4(b, n) &= a & b &= \sqrt[n]{a_4} & n &= \log_4^4 a
\end{align*}
\]
Tetration FAQ

Q: Does it have an inverse? A: Yeah, 2—which do you want?
Q: Is one better than the other? A: One we care about:

<table>
<thead>
<tr>
<th>If</th>
<th>then</th>
<th>and also</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b + n = a$</td>
<td>$b = a - n$</td>
<td>$n = a - b$</td>
</tr>
<tr>
<td>$b \times n = a$</td>
<td>$b = a/n$</td>
<td>$n = a/b$</td>
</tr>
<tr>
<td>$b^n = a$</td>
<td>$b = \sqrt[n]{a}$</td>
<td>$n = \log_b a$</td>
</tr>
<tr>
<td>$n b = a$</td>
<td>$b = \sqrt[n]{a}$</td>
<td>$n = \log^*_b a$</td>
</tr>
</tbody>
</table>

$log^*_b a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log^*_b \log_b a & \text{otherwise.}
\end{cases}$
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: One we care about:

\[ n^b = a \Rightarrow n = \log_b^* a \]

\[ \log_b^* a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log_b^* \log_b a & \text{otherwise}. 
\end{cases} \]

Q: Why should we care?
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: One we care about:

\[ n^b = a \quad \Rightarrow \quad n = \log_b^* a \]

\[ \log_b^* a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log_b \log_b a & \text{otherwise.}
\end{cases} \]

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</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>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: One we care about:

\[ n^b = a \Rightarrow n = \log_b^* a \]

\[ \log_b^* a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log_b^* \log_b a & \text{otherwise.} 
\end{cases} \]

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

\[
\begin{array}{c|cccccccc}
\hline
n & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
n^2 & 1 & 2 & 4 & & & & \\
\hline
\end{array}
\]
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?   A: One we care about:

\[ nb = a \implies n = \log_b^* a \]

\[ \log_b^* a = \begin{cases} 0 & \text{if } a \leq 1; \\ 1 + \log_b^* \log_b a & \text{otherwise.} \end{cases} \]

Q: Why should we care?   A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

<table>
<thead>
<tr>
<th>n</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n^2 )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: One we care about:

\[ n^b = a \implies n = \log_b^* a \]

\[ \log_b^* a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log_b^* \log_b a & \text{otherwise.}
\end{cases} \]

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

\[
\begin{array}{c|cccccc}
 n & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
 n^2 & 1 & 2 & 4 & 16 & 65,536 \\
\end{array}
\]
Tetration FAQ

Q: Does it have an inverse?    A: Yeah, 2—which do you want?
Q: Is one better than the other?    A: One we care about:

\[ n b = a \quad \Rightarrow \quad n = \log_b^* a \]

\[ \log_b^* a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log_b^* \log_b a & \text{otherwise.} 
\end{cases} \]

Q: Why should we care?    A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n^2 )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>16</td>
<td>65,536</td>
<td>2^{65,536}</td>
<td>2^{65,536}</td>
</tr>
</tbody>
</table>
Tetration FAQ

Q: Does it have an inverse?  A: Yeah, 2—which do you want?
Q: Is one better than the other?  A: One we care about:

\[ nb = a \Rightarrow n = \log^*_b a \]

\[ \log^*_b a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log^*_b \log_b a & \text{otherwise.} 
\end{cases} \]

Q: Why should we care?  A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

\[ \begin{array}{cccccccc}
 n & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
 n^2 & 1 & 2 & 4 & 16 & 65,536 & 2^{65,536} & \approx 2^{2.0035 \times 10^{19,728}} \\
\end{array} \]
Tetration FAQ


Q: Does it have an inverse?  A: Yeah, 2—which do you want?

Q: Is one better than the other?    A: One we care about:

\[ \log_b^* a = \begin{cases} 
0 & \text{if } a \leq 1; \\
1 + \log_b \log_b a & \text{otherwise.} 
\end{cases} \]

Q: Why should we care?    A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

<table>
<thead>
<tr>
<th>( a \leq 1 )</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>16</th>
<th>65,536</th>
<th>( 2^{65,536} )</th>
<th>( 2^{2.0035 \times 10^{19,728}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \log^* a )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
We’re going to use the chalkboard from here on, but if you want union-find slides to read on your own then I suggest these slides from Robert Sedgewick and Kevin Wayne’s algorithms & data structures course at Princeton University. I’ve included a selection of those same union-find slides as the rest of this PDF, and the rest of their original lectures may be found here.
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 \( \text{id}[\cdot] \) of size \( N \).
- Interpretation: \( p \) and \( q \) are connected if they have the same id.

\[
\begin{array}{cccccccccc}
i & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\text{id}[i] & 0 & 1 & 9 & 9 & 9 & 6 & 6 & 7 & 8 & 9 \\
\end{array}
\]

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.

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$id[i]$</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$id[i]$</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$id[i]$</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

5 and 6 are connected
2, 3, 4, and 9 are connected
3 and 6 not connected
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 \( \sim MN \) 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.

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!

a truism (roughly) since 1950!
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]...]]]`.

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>id[i]</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

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

keep going until it doesn't change

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 $id[id[id[...id[i]...]]]$.

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>id[i]</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>id[i]</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

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

only one value changes

keep going until it doesn't change
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

<table>
<thead>
<tr>
<th>algorithm</th>
<th>union</th>
<th>find</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick-find</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>Quick-union</td>
<td>N*</td>
<td>N</td>
</tr>
</tbody>
</table>

* 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 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 \( \text{sz[]} \) 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 \( \text{sz[]} \) array.

```java
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 | \( N \ast \) | \( N \)
Weighted QU | \( \lg N \ast \) | \( \lg N \)

* 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 \( \text{root}(i) \).
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.

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

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 7 8 3
5–9  8 1 3 3 3 5 7 8 3
7–3  8 1 3 3 3 5 3 8 3
4–8  8 1 3 3 3 5 3 3 3
6–1  8 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!

Linear algorithm?

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

Amazing fact:

• In theory, no linear linking strategy exists

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\lg^* N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>65536</td>
<td>4</td>
</tr>
<tr>
<td>265536</td>
<td>5</td>
</tr>
</tbody>
</table>

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

because $\lg^* N$ is a constant in this universe
Summary

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.

Bottom line.
**WQUPC makes it possible to solve problems that could not otherwise be addressed**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick-find</td>
<td>$M \cdot N$</td>
</tr>
<tr>
<td>Quick-union</td>
<td>$M \cdot N$</td>
</tr>
<tr>
<td>Weighted QU</td>
<td>$N + M \log N$</td>
</tr>
<tr>
<td>Path compression</td>
<td>$N + M \log N$</td>
</tr>
<tr>
<td>Weighted + path</td>
<td>$(M + N) \lg^* N$</td>
</tr>
</tbody>
</table>