CSE 326 Data Structures

Dave Bacon

Disjoint Sets
Logistics

- Homework 5 on web due Friday
- Project 3 out, “Shake-n-Bacon” code due Mon, Feb 26
- Reading: finish Weiss Chapter 8, start Chapter 7
- Lecturer Friday: Ruth Anderson
- Holiday Monday!
Find Operation

Find(x) - follow x to the root and return the root

Find(6) = 7
Union Operation

Union(x, y) - assuming x and y are roots, point y to x.

1, 2, 4, 5, 6, 7 8, 3, 3

Union(1, 7)
A Bad Case
A Bad Case

Union(2, 1)
Union(3, 2)
Union(n, n-1)

Find(1) n steps!!
Now this doesn’t look good 😞

Can we do better? Yes!

1. Improve union so that find only takes $O(\log n)$
   - Union-by-size
   - Reduces complexity to $O(m \log n + n)$

2. Improve find so that it becomes even better!
   - Path compression
   - Reduces complexity to almost $O(m + n)$
Weighted Union

- Weighted Union
  - Always point the smaller (total # of nodes) tree to the root of the larger tree
Example Again

W-Union(2,1)
W-Union(3,2)
\vdots
W-Union(n,2)

Find(1) constant time
Find(1)

smaller \downarrow larger
Analysis of Weighted Union

With weighted union an up-tree of height \( h \) has weight at least \( 2^h \).

- Proof by induction
  - **Basis**: \( h = 0 \). The up-tree has one node, \( 2^0 = 1 \)
  - **Inductive step**: Assume true for all \( h' < h \).

Minimum weight up-tree of height \( h \) formed by weighted unions

\[ W(T) \geq W(T_1) \geq 2^{h-1} \]

- Weighted union
- Induction hypothesis

\[ W(T_1) + W(T_2) = W(T) \geq 2^{h-1} + 2^{h-1} = 2^h \]
Analysis of Weighted Union (cont)

Let $T$ be an up-tree of weight $n$ formed by weighted union. Let $h$ be its height.

\[ n \geq 2^h \]
\[ \log_2 n \geq h \]
\[ h \leq \log_2 n \]

- Find($x$) in tree $T$ takes $O(\log n)$ time.
  - Can we do better?
Worst Case for Weighted Union

Binomial Tree
Worst Case for Weighted Union

\[ \frac{n}{2} \text{ Weighted Unions} \]

\[ \frac{n}{4} \text{ Weighted Unions} \]
Example of Worst Case (cont’)

After \( n/2 + n/4 + \ldots + 1 \) Weighted Unions:

- \( n \) elements
- Height \( O(\log n) \)

If there are \( n = 2^k \) nodes then the longest path from leaf to root has length \( k \).
Array Implementation

Union (1, 3)

Matrix:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>-1</td>
</tr>
</tbody>
</table>

Weights:

<table>
<thead>
<tr>
<th>up</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Diagram:

- Nodes 1, 2, 3, 4, 5, 6, 7
- Edges connecting nodes
- Union operation connecting nodes 1 and 3
Weighted Union

\[
W-\text{Union}(i, j : \text{index}) \{
    \text{// } i \text{ and } j \text{ are roots}
    \text{wi := weight}[i];
    \text{wj := weight}[j];
    \text{if } wi < wj \text{ then}
        \text{up}[i] := j;
        \text{weight}[j] := wi + wj;
    \text{else}
        \text{up}[j] := i;
        \text{weight}[i] := wi + wj;
\}
\]

new runtime for \text{Union}():
\[O(1)\]

new runtime for \text{Find}():
\[O(1)\]

runtime for \(m\) finds and \(n-1\) unions =
\[O(m \log n + n)\]
Nifty Storage Trick

- Use the same array representation as before
- Instead of storing $-1$ for the root, simply store $-\text{size}$

[Read section 8.4, page 276]
How about Union-by-height?

- Can still guarantee \(O(\log n)\) worst case depth

*Left as an exercise!*

- Problem: Union-by-height doesn’t combine very well with the new find optimization technique we’ll see next

“Path Compression”
Path Compression

- On a Find operation point all the nodes on the search path directly to the root.
Path Compression

- On a Find operation point all the nodes on the search path directly to the root.
Self-Adjustment Works
Path Compression Find

PC-Find(i : index) {
    r := i;
    while up[r] ≠ -1 do // find root//
        r := up[r];
    if i ≠ r then // compress path//
        k := up[i];
        while k ≠ r do
            up[i] := r;
            i := k;
            k := up[k]
        return(r)
    }

Interlude: A Really Slow Function

Ackermann’s function is a **really** big function $A(x, y)$ with inverse $\alpha(x, y)$ which is **really** small.

How fast does $\alpha(x, y)$ grow?

$\alpha(x, y) = 4$ for $x$ far larger than the number of atoms in the universe ($2^{300}$).

$\alpha$ shows up in:

- Computation Geometry (surface complexity)
- Combinatorics of sequences
A More Comprehensible Slow Function

\[ \log^* x = \text{number of times you need to compute} \ \log \ \text{to bring value down to at most 1} \]

E.g. \[ \log^* 2 = 1 \]
\[ \log^* 4 = \log^* 2^2 = 2 \]
\[ \log^* 16 = \log^* 2^{2^2} = 3 \]
\[ \log^* 65536 = \log^* 2^{2^{2^2}} = 4 \] (log \log \log \log 16 = 1)
\[ \log^* 2^{65536} = \ldots \ldots = 5 \] (log \log \log \log \log 65536 = 1)

Take this: \( \alpha(m,n) \) grows even slower than \( \log^* n \)!!
Complex Complexity of Union-by-Size + Path Compression

Tarjan proved that, with these optimizations, $p$ union and find operations on a set of $n$ elements have worst case complexity of $O(p \cdot \alpha(p, n))$. 

For all practical purposes this is amortized constant time:

$O(p \cdot 4)$ for $p$ operations!

$\frac{O(p \cdot 4)}{p} = O(1)$

• Very complex analysis – worse than splay tree analysis etc. that we skipped!
Disjoint Union / Find with Weighted Union and PC

- Worst case time complexity for a W-Union is $O(1)$ and for a PC-Find is $O(\log n)$.
- Time complexity for $m \geq n$ operations on $n$ elements is $O(m \log^* n)$ where $\log^* n$ is a very slow growing function.
  - $\log^* n < 7$ for all reasonable $n$. Essentially constant time per operation!
- Using “ranked union” gives an even better bound theoretically.
Amortized Complexity

• For disjoint union / find with weighted union and path compression.
  – average time per operation is essentially a constant.
  – worst case time for a PC-Find is $O(\log n)$.

• An individual operation can be costly, but over time the average cost per operation is not.