Disjoint Sets and Dynamic Equivalence Relations

CSE 373
Data Structures and Algorithms

Today’s Outline

• Announcements
  – Assignment #4 due this Friday Feb 13th at the beginning of lecture.

• Today’s Topics:
  – Disjoint Sets & Dynamic Equivalence

Desired Properties

• None of the boundary is deleted
• Every cell is reachable from every other cell.
• Only one path from any one cell to another (There are no cycles – no cell can reach itself by a path unless it retraces some part of the path.)

Number the Cells

We have disjoint sets \( P = \{ \{1\}, \{2\}, \{3\}, \ldots, \{36\} \} \) each cell is unto itself.
We have all possible edges \( E = \{(1,2), (1,7), (2,8), (2,3), \ldots\} \) 60 edges total.

Basic Algorithm

• \( P \) = set of sets of connected cells
• \( E \) = set of edges
• \( \text{Maze} \) = set of maze edges (initially empty)

While there is more than one set in \( P \) {
  pick a random edge \((x,y)\) and remove from \( E \):
  \( u := \text{Find}(x) \);
  \( v := \text{Find}(y) \);
  if \( u \neq v \) then // removing edge \((x,y)\) connects previously non-
    // connected cells \( x \) and \( y \) leave this edge removed!
    \( \text{Union}(u,v) \);
  else // cells \( x \) and \( y \) were already connected, add this
    // edge to set of edges that will make up final maze.
    add \((x,y)\) to \( \text{Maze} \);
}
All remaining members of \( E \) together with \( \text{Maze} \) form the maze

Example Step

Pick \((8,14)\)
Example

\[ \text{Find}(8) = 7 \]
\[ \text{Find}(14) = 20 \]

Example

\[ \text{Pick} (19,20) \]

Example at the End

Implementing the Disjoint Sets ADT

- \( n \) elements.
- Total Cost of: \( m \) finds, \( \leq n-1 \) unions

Target complexity: \( O(m+n) \)
- \( O(1) \) worst-case for find as well as union would be great, but…

Known result: both find and union cannot be done in worst-case \( O(1) \) time

Up-Tree for Disjoint Union/Find

Initial state: \( 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \)

After several Unions:

Roots are the names of each set.

Find Operation

\[ \text{Find}(x) \text{ - follow } x \text{ to the root and return the root} \]
Union Operation

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

1. Union(1, 7)

Simple Implementation

- Array of indices

<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></td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Up[x] = 0 means x is a root.

Implementation

```c
int Find(int x) {
    while (up[x] != 0) {
        x = up[x];
    }
    return x;
}
```

```c
void Union(int x, int y) {
    up[y] = x;
}
```

runtime for Union():

runtime for Find():

runtime for m Finds and n-1 Unions:

Now this doesn’t look good 😞

Can we do better? Yes!

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

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

A Bad Case

```
1  2  3  ...  6
   \  /  \\
  2  3  ...  6
   /  \\
1  2  3  ...  6
   /    \\
1  2  3  ...  6
   /        \\
1  2  3  ...  6
   /          \\
1  2  3  ...  6
   /            \\
1  2  3  ...  6
```

Find(1) n steps!!

Weighted Union

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

```
2  1  3  4  7
   \   / \\
  2   1  3
   /   /
2   1  3  4
   /   /
2   1  3  4
   /   /
2   1  3  4
   /   /
2   1  3  4
   /   /
2   1  3  4
```

W-Union(1, 7)
Example Again

\[ \begin{align*}
&1 \quad 2 \quad 3 \quad \cdots \quad 6 \\
&\quad 2 \quad 3 \quad \cdots \quad 6 \\
&\quad \quad 2 \quad 3 \\
&\quad \quad \quad \quad 5 \\
&\text{W-Union}(2, 1) \\
&\text{W-Union}(3, 3) \\
&\vdots \\
&\text{W-Union}(n, 2) \\
&\text{Find}(1) \quad \text{constant time}
\end{align*} \]

Analysis of Weighted Union

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

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

\[ W(T) \geq W(T_1) + W(T_2) \geq 2^{h-1} + 2^{h-1} = 2^h \]

Analysis of Weighted Union (cont)

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

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

- **Find(x)** in tree \( T \) takes \( O(\log n) \) time.
  - Can we do better?

Worst Case for Weighted Union

Let \( n \) nodes be given and \( n/2 \) weighted unions:

- \( n/2 \) Weighted Unions
- \( n/4 \) Weighted Unions

Example of Worst Case (cont)

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

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

Array Implementation

\[ \begin{align*}
&2 \quad 1 \\
&\quad 2 \\
&\quad \quad 1 \\
&\text{Find} \\
&\text{up weight}
\end{align*} \]
Weighted Union

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

\text{new runtime for Union}: 

new runtime for Find():

runtime for m finds and n-1 unions =

Nifty Storage Trick

- Use the same array representation as before
- Instead of storing -1 for the root, simply store -size

[Read section 8.4, page 299]

How about Union-by-height?

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

\text{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

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

Student Activity:  

Draw the result of Find(e):

Self-Adjustment Works

PC-\text{Find}(x)
Path Compression Find

```c
PC-Find(i : index) {
    r := i;
    while up[r] ≠ -1 do //find root
        r := up[r];
    // Assert: r= the root, up[r] = -1
    if i ≠ r then // if i was not a root
        temp := up[i];
        while temp ≠ r do // compress path
            up[i] := r;
            i := temp;
            temp := up[temp]
        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. \( \log^* 2 = 1 \)
\( \log^* 4 = \log^* 2^2 = 2 \)
\( \log^* 16 = \log^* 2^4 = 3 \) \( \text{(log log 16 = 1)} \)
\( \log^* 65536 = \log^* 2^{16} = 4 \) \( \text{(log log log 65536 = 1)} \)
\( \log^* 2^{65536} = \ldots \ldots \ldots = 5 \)

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!

- Very complex analysis

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!

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.