Disjoint Union / Find

CSE 373
Data Structures
Lecture 16
Reading

• Reading
  › Chapter 8 (you can skip Section 6)
Equivalence Relations

• A relation \( R \) is defined on set \( S \) if for every pair of elements \( a, b \in S \), \( a \, R \, b \) is either true or false.

• An equivalence relation is a relation \( R \) that satisfies the 3 properties:
  › Reflexive: \( a \, R \, a \) for all \( a \in S \)
  › Symmetric: \( a \, R \, b \) iff \( b \, R \, a \); \( a, b \in S \)
  › Transitive: \( a \, R \, b \) and \( b \, R \, c \) implies \( a \, R \, c \)
Equivalence Classes

• Given an equivalence relation $R$, decide whether a pair of elements $a, b \in S$ is such that $a R b$.

• The equivalence class of an element $a$ is the subset of $S$ of all elements related to $a$.

• Equivalence classes are disjoint sets.
Dynamic Equivalence Problem

- Starting with each element in a singleton set, and an equivalence relation, build the equivalence classes
- Requires two operations:
  - Find the equivalence class (set) of a given element
  - Union of two sets
- It is a dynamic (on-line) problem because the sets change during the operations and Find must be able to cope!
Disjoint Union - Find

- Maintain a set of pairwise disjoint sets.
  - \{3,5,7\}, \{4,2,8\}, \{9\}, \{1,6\}
- Each set has a unique name, one of its members
  - \{3,5,7\}, \{4,2,8\}, \{9\}, \{1,6\}
Union

- Union(x,y) – take the union of two sets named x and y
  - \{3,5,7\}, \{4,2,8\}, \{9\}, \{1,6\}
  - Union(5,1)
    - \{3,5,7,1,6\}, \{4,2,8\}, \{9\},
Find

- Find(x) – return the name of the set containing x.
  - \{3,5,7,1,6\}, \{4,2,8\}, \{9\},
  - Find(1) = 5
  - Find(4) = 8
An Application

- Build a random maze by erasing edges.
An Application (ct’d)

- Pick Start and End

```
Start
```
```
```
```
```
```
```
```
```
```
End
```
An Application (ct’d)

- Repeatedly pick random edges to delete.
Desired Properties

- None of the boundary is deleted
- Every cell is reachable from every other cell.
- There are no cycles – no cell can reach itself by a path unless it retraces some part of the path.
A Cycle (we don’t want that)
A Good Solution
Good Solution: A Hidden Tree
Number the Cells

We have disjoint sets $S = \{ \{1\}, \{2\}, \{3\}, \{4\}, \ldots, \{36\} \}$ each cell is unto itself. We have all possible edges $E = \{ (1,2), (1,7), (2,8), (2,3), \ldots \}$ 60 edges total.

<table>
<thead>
<tr>
<th>Start</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>
Basic Algorithm

- \( S = \) 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 \( S \)
  pick a random edge \((x,y)\) and remove from \( E \)
  \( u := \text{Find}(x); \ v := \text{Find}(y); \)
  if \( u \neq v \) then
    \( \text{Union}(u,v) \) //knock down the wall between the cells (cells in
    //the same set are connected)
  else
    add \((x,y)\) to Maze //don’t remove because there is already
    //a path between \( x \) and \( y \)

All remaining members of \( E \) together with Maze form the maze
Example Step

Pick (8,14)

<table>
<thead>
<tr>
<th>Start</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

End

S
{1,2,7,8,9,13,19}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{14,20,26,27}
{15,16,21}

...
Example

\[ S \]
\[ \{1,2,7,8,9,13,19\} \]
\[ \{3\} \]
\[ \{4\} \]
\[ \{5\} \]
\[ \{6\} \]
\[ \{10\} \]
\[ \{11,17\} \]
\[ \{12\} \]
\[ \{14,20,26,27\} \]
\[ \{15,16,21\} \]
\[ \ldots \]
\[ \{22,23,24,29,39,32\} \]
\[ 33,34,35,36\} \]

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

\[ S \]
\[ \{1,2,7,8,9,13,19,14,20,26,27\} \]
\[ \{3\} \]
\[ \{4\} \]
\[ \{5\} \]
\[ \{6\} \]
\[ \{10\} \]
\[ \{11,17\} \]
\[ \{12\} \]
\[ \{15,16,21\} \]
\[ \ldots \]
\[ \{22,23,24,29,39,32\} \]
\[ 33,34,35,36\} \]

\[ \text{Union}(7,20) \]
### Example

Pick (19, 20)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

End

S
- \{1, 2, 7, 8, 9, 13, 19, 14, 20, 26, 27\}
  - \{3\}
  - \{4\}
  - \{5\}
  - \{6\}
  - \{10\}
  - \{11, 17\}
  - \{12\}
  - \{15, 16, 21\}
  - \{22, 23, 24, 29, 39, 32, 33, 34, 35, 36\}

2/25/03 Union/Find - Lecture 16
Example at the End

\[ S \{1,2,3,4,5,6,7,\ldots, 36\} \]

\[ \text{Start} \]

\[ \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
7 & 8 & 9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16 & 17 & 18 \\
19 & 20 & 21 & 22 & 23 & 24 \\
25 & 26 & 27 & 28 & 29 & 30 \\
31 & 32 & 33 & 34 & 35 & 36 \\
\end{array} \]

\[ \text{End} \]

\[ \text{E Maze} \]
Up-Tree for DU/F

Initial state

Intermediate state

Roots are the names of each set.
Find Operation

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

Find(6) = 7
Union Operation

- Union(i,j) - assuming i and j roots, point i to j.
Simple Implementation

- Array of indices (Up[i] is parent of i)

```
<table>
<thead>
<tr>
<th>up</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>7</th>
<th>7</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
</table>
```

$\text{Up}[x] = 0$ means $x$ is a root.
Union

Union(up[] : integer array, x,y : integer) : {
  //precondition: x and y are roots/
  Up[x] := y
}

Constant Time!
Find

- Design Find operator
  - Recursive version
  - Iterative version

```latex
\text{Find}(\text{up}[\cdot] : \text{integer array}, \ x : \text{ integer}) : \text{ integer} \{ \\
\quad \text{//precondition: } x \text{ is in the range 1 to size//} \\
\quad \text{ ??? } \\
\}
```
A Bad Case

Union(1,2)

Union(2,3)

...:

Union(n-1,n)

Find(1) n steps!!
Weighted Union

- Weighted Union (weight = number of nodes)
  - Always point the smaller tree to the root of the larger tree

```
W-Union(1,7)
```

```
1
  2
  1
  3

4
  5
  4

6
```

```
7
```

```
W-Union(1,7)
```
Example Again

Union(1,2)

Union(2,3)

Union(n-1,n)

Find(1)  constant time
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 $h' < h$.

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

• Let $T$ be an up-tree of weight $n$ formed by weighted union. Let $h$ be its height.
• $n \geq 2^h$
• $\log_2 n \geq h$
• $\text{Find}(x)$ in tree $T$ takes $O(\log n)$ time.
• Can we do better?
Worst Case for Weighted Union

n/2 Weighted Unions

n/4 Weighted Unions
Example of Worst Cast (cont’)

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

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

Can save the extra space by storing the complement of weight in the space reserved for the root
Weighted Union

\[ W-\text{Union}(i, j : \text{index})\{
//i and j are roots//
wi := \text{weight}[i];
wj := \text{weight}[j];
if wi < wj then
  up[i] := j;
  \text{weight}[j] := wi + wj;
else
  up[j] := i;
  \text{weight}[i] := wi + wj;
\} \]
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] ≠ 0 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)
}
Example
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.
Find Solutions

Recursive

Find(up[] : integer array, x : integer) : integer {
  //precondition: x is in the range 1 to size/
  if up[x] = 0 then return x
  else return Find(up,up[x]);
}

Iterative

Find(up[] : integer array, x : integer) : integer {
  //precondition: x is in the range 1 to size/
  while up[x] ≠ 0 do
    x := up[x];
  return x;
}