CSE 326: Data Structures

Disjoint Union/Find

James Fogarty

Spring 2009
Announcements

• Homework 4 Due

• Course Feedback

• Homework 5 Released

• Project 3 Released
Making Connections

You have a set of nodes (numbered 1-9) on a network. You are given a sequence of pairwise connections between them:

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

Q: Are nodes 2 and 4 (indirectly) connected?
Q: How about nodes 3 and 8?
Q: Are any of the paired connections redundant due to indirect connections?
Q: How many sub-networks do you have?
Making Connections

Answering these questions is much easier if we create disjoint sets of nodes that are connected:

Start: {1} {2} {3} {4} {5} {6} {7} {8} {9}
3-5: {1} {2} {3, 5} {4} {6} {7} {8} {9}
4-2: {1} {2, 4} {3, 5} {6} {7} {8} {9}
1-6: {1, 6} {2, 4} {3, 5} {7} {8} {9}
5-7: {1, 6} {2, 4} {3, 5, 7} {8} {9}
4-8: {1, 6} {2, 4, 8} {3, 5, 7} {9}
3-7

Q: Are nodes 2 and 4 (indirectly) connected?
Q: How about nodes 3 and 8?
Q: Are any of the paired connections redundant due to indirect connections?
Q: How many sub-networks do you have?
Equivalence Relations

Relation \( R \):

- For every pair of elements \((a, b)\) in a set \( S \), \( a \, R \, b \) is either true or false.
- If \( a \, R \, b \) is true, then \( a \) is related to \( b \).

An equivalence relation satisfies:

1. (Reflexive) \( a \, R \, a \)
2. (Symmetric) \( a \, R \, b \) iff \( b \, R \, a \)
3. (Transitive) \( a \, R \, b \) and \( b \, R \, c \) implies \( a \, R \, c \)
Applications of Disjoint Sets

Maintaining disjoint sets in this manner arises in a number of areas

Today: A Cute Problem
(Maze Building)

Later: A Major Graph Problem
(Minimum Spanning Trees)
Disjoint Set ADT

• Data: set of pairwise disjoint sets.

• Required operations
  – **Union** – merge two sets to create their union
  – **Find** – determine which set an item appears in

• A common operation sequence:
  – Connect two elements if not already connected:
    \[
    \text{if } (\text{Find}(x) \neq \text{Find}(y)) \text{ then } \text{Union}(x, y)
    \]
Disjoint Sets and Naming

• Maintain a set of pairwise disjoint sets.
  – \{3,5,7\} , \{4,2,8\}, \{9\}, \{1,6\}

• Each set has a unique name: for convenience, 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\}
  - Union(9, 8)
    \{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
  - Find(9) = 9
Example

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

Find(8) = 7
Find(14) = 20
Union(7,20)

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

{22,23,24,29,30,32,33,34,35,36}
Cute Application

• Build a random maze by erasing edges.
Cute Application

- Pick Start and End
Cute Application

- Repeatedly pick random edges to delete.
Desired Properties

- None of the boundary is deleted
- Every cell 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

Start

End

Diagram of a cycle starting from Start and ending at End.
A 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></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
</tbody>
</table>

End
Maze Building with Disjoint Union/Find

Algorithm sketch:
1. Choose wall at random.
   → Boundary walls are not in wall list, because we cannot delete them
2. Erase wall if the neighbors are in disjoint sets.
   → Avoids cycles
3. Take union of those sets.
4. Repeat until there is only one set.
   → Every cell reachable from every other cell.
Pseudocode

• S = set of sets of connected cells
  – Initialize to {{1}, {2}, …, {n}}
• W = set of interior walls
  – Initialize to set of all interior walls {{1,2},{1,7}, …}
• Maze = set of walls included in maze (initially empty)

While there is more than one set in S
  Pick a random non-boundary wall (x,y) and remove from W
  u = Find(x);
  v = Find(y);
  if u $\neq$ v then
    Union(u,v)
  else
    Add wall (x,y) to Maze

Add remaining members of W to Maze
Example Step

S
\{1,2,7,8,9,13,19\}
\{3\}
\{4\}
\{5\}
\{6\}
\{10\}
\{11,17\}
\{12\}
\{14,20,26,27\}
\{15,16,21\}
\{22,23,24,29,30,32,33,34,35,36\}

---

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

Pick (8,14)
Example

\[
\begin{align*}
S & \{1,2,7,8,9,13,19\} \\
& \{3\} \\
& \{4\} \\
& \{5\} \\
& \{6\} \\
& \{10\} \\
& \{11,17\} \\
& \{12\} \\
& \{14,20,26,27\} \\
& \{15,16,21\} \\
& \ldots \\
& \{22,23,24,29,30,32,33,34,35,36\}
\end{align*}
\]

Find(8) = 7
Find(14) = 20
Union(7,20)

\[
\begin{align*}
S & \{1,2,7,8,9,13,14,19,20,26,27\} \\
& \{3\} \\
& \{4\} \\
& \{5\} \\
& \{6\} \\
& \{10\} \\
& \{11,17\} \\
& \{12\} \\
& \{14,20,26,27\} \\
& \{15,16,21\} \\
& \ldots \\
& \{22,23,24,29,30,32,33,34,35,36\}
\end{align*}
\]
### 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>

**S**

{1,2,7,8,9,13,14,19,20,26,27}

{3}

{4}

{5}

{6}

{10}

{11,17}

{12}

{15,16,21}

{22,23,24,29,30,32,33,34,35,36}
### Example at the End

Consider a set $S = \{1,2,3,4,5,6,7,\ldots, 36\}$.

#### Remaining walls in $W$

- Remaining walls in $W$ are represented by vertical lines.
- The walls in $W$ are checked and added to the maze.

#### Exterior

- Exterior walls are represented by horizontal lines.

#### Maze Construction

- The maze is constructed by connecting the start and end points.
- The walls are not shown in the diagram, but they are considered in the construction process.

#### Maze Diagram

The diagram shows a 6x6 grid with walls and paths labeled from 1 to 36.

**Start**

<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>End</td>
</tr>
</tbody>
</table>

**End**
Implementing the DS ADT

- $n$ elements,
  Total Cost of: $m$ finds, at most $n-1$ unions

- Target complexity: $O(m+n)$
  \( \text{i.e. } O(1) \text{ amortized} \)

- $O(1)$ worst-case for find as well as union would be great, but...

Known result: find and union cannot both be done in worst-case $O(1)$ time
Data Structure for the DS ADT

- So, how are we going to do this?

- Does it help if I tell you the data structure is called an up-tree?
Data Structure for the DS ADT

• Observation: *trees* let us find many elements given one root…

• Idea: if we *reverse* the pointers (make them point up from child to parent), we can find a single root from many elements…

• Idea: Use one tree for each equivalence class. The name of the class is the tree root.
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.
Implementation

• How might we implement this?

• What if I said you just need an array?
Simple Implementation

• Array of indices

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

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

Constant Time!
Recursive

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

Iterative

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

    return x;
}
A Bad Case

Find(1): $n$ steps!!

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