CSE373: Data Structures & Algorithms

Lecture 10: Disjoint Sets and the Union-Find ADT

Lauren Milne
Spring 2015
Announcements

• Start homework 3 soon…..
  – Priority queues and binary heaps
  
  – TA Sessions on Tuesday and Thursday
  
  – Office hours for Conrad or Catie covered by other Tas this week.
Where we are

Last lecture:
• Priority queues and binary heaps

Today:
• Disjoint sets
• The union-find ADT for disjoint sets

Next lecture:
• Basic implementation of the union-find ADT with “up trees”
• Optimizations that make the implementation much faster
Disjoint sets

• A set is a collection of elements (no-repeats)

• Two sets are said to be disjoint if they have no element in common.
  • \( S_1 \cap S_2 = \emptyset \)

• For example, \{1, 2, 3\} and \{4, 5, 6\} are disjoint sets.
• For example, \{x, y, z\} and \{t, u, x\} are not disjoint.
Partitions

A **partition** \( P \) of a set \( S \) is a set of sets \( \{S_1, S_2, \ldots, S_n\} \) such that every element of \( S \) is in **exactly one** \( S_i \)

Put another way:
- \( S_1 \cup S_2 \cup \ldots \cup S_k = S \)
- \( i \neq j \) implies \( S_i \cap S_j = \emptyset \) (sets are disjoint with each other)

Example:
- Let \( S \) be \( \{a,b,c,d,e\} \)
- One partition: \( \{a\}, \{d,e\}, \{b,c\} \)
- Another partition: \( \{a,b,c\}, \{d\}, \{e\} \)
- A third: \( \{a,b,c,d,e\} \)
- **Not a partition**: \( \{a,b,d\}, \{c,d,e\} \ldots \text{element } d \text{ appears twice} \)
- **Not a partition**: \( \{a,b\}, \{e,c\} \ldots \text{missing element } d \)
Binary relations

- A **binary relation** $R$ is defined on a set $S$ if for every pair of elements $(x,y)$ in the set, $R(x,y)$ is either true or false. If $R(x,y)$ is true, we say $x$ is related to $y$.
  - i.e. a collection of ordered pairs of elements of $S$
  - (Unary, ternary, quaternary, ... relations defined similarly)

- Examples for $S = \text{people-in-this-room}$
  - Sitting-next-to-each-other relation
  - First-sitting-right-of-second relation
  - Went-to-same-high-school relation
Properties of binary relations

- A relation $R$ over set $S$ is:
  - reflexive, if $R(a,a)$ holds for all $a$ in $S$
    - e.g. The relation “$\leq$” on the set of integers $\{1, 2, 3\}$ is $\{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}$
      It is reflexive because $(1, 1), (2, 2), (3, 3)$ are in this relation.
  - symmetric if and only if for any $a$ and $b$ in $S$, whenever $(a, b)$ is in $R$, $(b, a)$ is in $R$.
    - e.g. The relation “$=$” on the set of integers $\{1, 2, 3\}$ is $\{(1, 1), (2, 2), (3, 3)\}$ and it is symmetric.
  - transitive if $R(a,b)$ and $R(b,c)$ then $R(a,c)$ for all $a,b,c$ in $S$
    - e.g. The relation “$\leq$” on the set of integers $\{1, 2, 3\}$ is transitive, because for $(1, 2)$ and $(2, 3)$ in “$\leq$”, $(1, 3)$ is also in “$\leq$” (and similarly for the others)
Equivalence relations

• A binary relation $R$ is an equivalence relation if $R$ is reflexive, symmetric, and transitive.

• Examples
  – Same gender
  – Electrical connectivity, where connections are metal wires
  – "Has the same birthday as" on the set of all people
  – …
Punch-line

- Equivalence relations give rise to partitions.

- Every partition induces an equivalence relation.
- Every equivalence relation induces a partition.

- Suppose $P=\{S_1, S_2, \ldots, S_n\}$ is a partition
  - Define $R(x,y)$ to mean $x$ and $y$ are in the same $S_i$
    - $R$ is an equivalence relation.

- Suppose $R$ is an equivalence relation over $S$
  - Consider a set of sets $S_1, S_2, \ldots, S_n$ where
    1. $x$ and $y$ are in the same $S_i$ if and only if $R(x,y)$
    2. Every $x$ is in some $S_i$
    - This set of sets is a partition.
Example

• Let $S$ be \{a,b,c,d,e\}

• One partition: \{a,b,c\}, \{d\}, \{e\}

• The corresponding equivalence relation:
  \[(a,a), (b,b), (c,c), (a,b), (b,a), (a,c), (c,a), (b,c), (c,b), (d,d), (e,e)\]
Example

• Let \( S \) be \{a, b, c, d, e\}

• The equivalence relation: \((a,a),(a,b),(b,a),(b,b),(c,c),(d,d),(e,e)\)

• The corresponding partition?
  \{\{a,b\},\{c\},\{d\},\{e\}\}

Spring 2015
CSE373: Data Structures & Algorithms
11
The **Union-Find ADT**

- The union-find ADT (or "Disjoint Sets" or "Dynamic Equivalence Relation") keeps track of a set of elements partitioned into a number of disjoint subsets.

- Many uses!
  - Road/network/graph connectivity (will see this again)
    - keep track of “connected components” e.g., in social network
  - Partition an image by connected-pixels-of-similar-color

- Not as common as dictionaries, queues, and stacks, but valuable because implementations are very fast, so when applicable can provide big improvements
Union-Find Operations

- Given an unchanging set $S$, create an initial partition of a set
  - Typically each item in its own subset: $\{a\}$, $\{b\}$, $\{c\}$, …
  - Give each subset a “name” by choosing a representative element

- Operation find takes an element of $S$ and returns the representative element of the subset it is in

- Operation union takes two subsets and (permanently) makes one larger subset
  - A different partition with one fewer set
  - Affects result of subsequent find operations
  - Choice of representative element up to implementation
Example

• Let $S = \{1,2,3,4,5,6,7,8,9\}$
• Let initial partition be (will highlight representative elements red)
  \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}
• union(2,5):
  \{1\}, \{2, 5\}, \{3\}, \{4\}, \{6\}, \{7\}, \{8\}, \{9\}
• find(4) = 4, find(2) = 2, find(5) = 2
• union(4,6), union(2,7)
  \{1\}, \{2, 5, 7\}, \{3\}, \{4, 6\}, \{8\}, \{9\}
• find(4) = 6, find(2) = 2, find(5) = 2
• union(2,6)
  \{1\}, \{2, 4, 5, 6, 7\}, \{3\}, \{8\}, \{9\}
No other operations

• All that can “happen” is sets get unioned
  – No “un-union” or “create new set” or …

• As always: trade-offs
  – Implementations will exploit this small ADT

• Surprisingly useful ADT
  – But not as common as dictionaries or priority queues
Example application: maze-building

• Build a random maze by erasing edges

  Possible to get from anywhere to anywhere
  • Including “start” to “finish”
  • No loops possible without backtracking
    • After a “bad turn” have to “undo”
Maze building

Pick start edge and end edge

Start

End
Repeatedly pick random edges to delete

One approach: just keep deleting random edges until you can get from start to finish

Start

End
Problems with this approach

1. How can you tell when there is a path from start to finish?
   - We do not really have an algorithm yet

2. We could have cycles, which a “good” maze avoids
   - Want one solution and no cycles
Revised approach

- Consider edges in random order (i.e. pick an edge)
- Only delete an edge if it introduces no cycles (how? TBD)
- When done, we will have a way to get from any place to any other place (including from start to end points)
Cells and edges

- Let’s number each cell
  - 36 total for 6 x 6
- An (internal) edge \((x,y)\) is the line between cells \(x\) and \(y\)
  - 60 total for 6x6: \((1,2), (2,3), \ldots, (1,7), (2,8), \ldots\)
The trick

• Partition the cells into disjoint sets
  – Two cells in same set if they are “connected”
  – Initially every cell is in its own subset
• If removing an edge would connect two different subsets:
  – then remove the edge and union the subsets
  – else leave the edge because removing it makes a cycle
The algorithm

- **P** = disjoint sets of connected cells
  - initially each cell in its own 1-element set
- **E** = set of edges not yet processed, initially all (internal) edges
- **M** = set of edges kept in maze (initially empty)

while P has more than one set {
  - Pick a random edge \((x,y)\) to remove from E
  - \(u = \text{find}(x)\)
  - \(v = \text{find}(y)\)
  - if \(u==v\)
    - add \((x,y)\) to M // same subset, leave edge in maze, do not create cycle
  else
    - \(\text{union}(u,v)\) // connect subsets, remove edge from maze
}
Add remaining members of E to M, then output M as the maze
Example

Pick edge (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

P

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

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

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

\[ P \{1,2,7,8,9,13,19,14,20,26,27\} \{3\} \{4\} \{5\} \{6\} \{10\} \{11,17\} \{12\} \{15,16,21\} \{18\} \{25\} \{28\} \{31\} \{22,23,24,29,30,32,33,34,35,36\} \]
Example: Add edge to $M$ step

Pick edge $(19,20)$
Find $(19) = 7$
Find $(20) = 7$
Add $(19,20)$ to $M$
At the end of while loop

- Stop when P has one set (i.e. all cells connected)
- Suppose green edges are already in M and black edges were not yet picked
  - Add all black edges to M

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

P
\{1,2,3,4,5,6,7,\ldots 36\}

Done! 😊
A data structure for the union-find ADT

• Start with an initial partition of $n$ subsets
  – Often 1-element sets, e.g., \{1\}, \{2\}, \{3\}, …, \{n\}

• May have any number of \texttt{find} operations
• May have up to $n$-1 \texttt{union} operations in any order
  – After $n$-1 \texttt{union} operations, every \texttt{find} returns same 1 set
Teaser: the up-tree data structure

- Tree structure with:
  - No limit on branching factor
  - References from children to parent

- Start with forest of 1-node trees

- Possible forest after several unions:
  - Will use roots for set names