CSE373: Data Structures & Algorithms
Lecture 9: Disjoint Sets & Union-Find

Dan Grossman
Fall 2013
The plan

• What are disjoint sets
  – And how are they “the same thing” as equivalence relations

• The union-find ADT for disjoint sets

• Applications of union-find

Next lecture:

• Basic implementation of the 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 disjoint if they have no elements in common
  \[ S_1 \cap S_2 = \emptyset \]

- Example: \{a, e, c\} and \{d, b\} are disjoint

- 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\}, $\emptyset$, \{d\}, \{e\}
- A third: \{a, b, c, d, e\}
- Not a partition: \{a, b, d\}, \{c, d, e\}
- Not a partition of $S$: \{a, b\}, \{e, c\}
Binary relations

- \( S \times S \) is the set of all pairs of elements of \( S \)
  - Example: If \( S = \{a,b,c\} \)
    then \( S \times S = \{(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c,a),(c,b),(c,c)\} \)

- A binary relation \( R \) on a set \( S \) is any subset of \( S \times S \)
  - Write \( R(x,y) \) to mean \((x,y)\) is “in the relation”
  - (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
  - Same-gender-relation
  - First-is-younger-than-second relation
Properties of binary relations

- A binary relation $R$ over set $S$ is **reflexive** means
  \[ R(a,a) \] for all $a$ in $S$

- A binary relation $R$ over set $S$ is **symmetric** means
  \[ R(a,b) \text{ if and only if } R(b,a) \] for all $a,b$ in $S$

- A binary relation $R$ over set $S$ is **transitive** means
  \[ \text{If } R(a,b) \text{ and } R(b,c) \text{ then } R(a,c) \] for all $a,b,c$ in $S$

- 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
  - Same-gender-relation
  - First-is-younger-than-second relation
Equivalence relations

• A binary relation \( R \) is an equivalence relation if \( R \) is reflexive, symmetric, and transitive

• Examples
  – Same gender
  – Connected roads in the world
  – Graduated from same high school?
  – …
Punch-line

• Every partition induces an equivalence relation
• Every equivalence relation induces a partition

• Suppose \( P=\{S_1, S_2, \ldots, S_n\} \) be 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

Fall 2013

CSE373: Data Structures & Algorithms
Example

• Let $S$ be \{a,b,c,d,e\}

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

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

• What are disjoint sets
  – And how are they “the same thing” as equivalence relations

• The union-find ADT for disjoint sets

• Applications of union-find

Next lecture:

• Basic implementation of the ADT with “up trees”

• Optimizations that make the implementation much faster
The operations

• Given an unchanging set \( S \), \textbf{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 \textit{representative element}

• Operation \textbf{find} takes an element of \( S \) and returns the representative element of the subset it is in

• Operation \textbf{union} takes two subsets and (permanently) makes one larger subset
  – A different partition with one fewer set
  – Affects result of subsequent \textbf{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: list of applications after one example surprising one
  – 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
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 have cycles, which a “good” maze avoids
   - Want one solution and no cycles
Revised approach

• Consider edges in random order
• But only delete them if they introduce no cycles (how? TBD)
• When done, will have one way to get from any place to any other place (assuming no backtracking)

• Notice the funny-looking tree in red
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**: “are they connected”
  – Initially every cell is in its own subset
• If 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

- \( P = \text{disjoint sets} \) of connected cells, initially each cell in its own 1-element set
- \( E = \text{set} \) of edges not yet processed, initially all (internal) edges
- \( M = \text{set} \) 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 \( u == v \)
  then add \((x,y)\) to \( M \) // same subset, do not create cycle
else \( \text{union}(u,v) \) // do not put edge in \( M \), connect subsets
}
Add remaining members of \( E \) to \( M \), then output \( M \) as 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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 step

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

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

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

Pick (19,20)

<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>

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\}
At the end

• Stop when P has one set
• Suppose green edges are already in M and black edges were not yet picked
  – Add all black edges to M

\[
P = \{1,2,3,4,5,6,\ldots, 36\}
\]
Other applications

• Maze-building is:
  – Cute
  – Homework 4 😊
  – A surprising use of the union-find ADT

• Many other uses (which is why an ADT taught in CSE373):
  – Road/network/graph connectivity (will see this again)
    • “connected components” e.g., in social network
  – Partition an image by connected-pixels-of-similar-color
  – Type inference in programming languages

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