Disjoint Set Class

Overview

- Equivalence Relations
- The Dynamic Equivalence Problem
  - Simple algorithms for solving:
    - Find\Union
  - Smart Union Algorithms
  - Path Compression
  - Example
Equivalence Relations (~)

- Relation which satisfies the three properties:
  - Reflexive - \( a \, R \, a \), for all \( a \in S \)
  - Symmetric - \( a \, R \, b \) if and only if \( b \, R \, a \)
  - Transitive – \( a \, R \, b \) and \( b \, R \, c \) implies that \( a \, R \, c \)
- Examples:
  - Electrical Connectivity
  - Same country: Two cities are related if they belong to the same country

The Dynamic Equivalence Problem

- Given an equivalence relation ~, decide if \( a \sim b \) for any \( a \) and \( b \)
  - Solving via a 2-d array of booleans
  - What if relation is implicit?
- Equivalence class of an element \( a \in S \) is a subset of \( S \) which contains all the elements that are related to \( a \)
  - Check if \( a \) and \( b \) belong to the same equivalence class
- Disjoint sets
Disjoint Set Union/Find Algorithm

- **find**: returns the equivalence class for a given element
- **union**: adds relations
  - Merges two equivalence classes
- **Dynamic**: the sets change during the course of the algorithm
- **Observations**:
  - Values don’t matter, location matters
  - find(a) == find(b) matters

Data structure for solving the problem

- **find** in constant worst-time?
- **union** in constant worst-time?
- **Consider**:
  - Each set as a tree
  - Initially – all elements are disjoint
Union operation

- union: make parent link of root of one tree to the root of the other tree
Smart Union Algorithms

- union-by-size
  - Always make smaller tree subtree of the larger
  - No extra space: instead of -1 set as -(size of tree) for root

![Diagram of union-by-size algorithm]

Smart Union Algorithms

- union-by-height: Make a shallow tree a subtree of a deep tree
  - Update height only if two equally deep trees are joined
  - Instead of -1, -(height) for root

![Diagram of union-by-height algorithm]
Smart find algorithm – Path compression

- Independent of union algorithm
- Make every node on the path from x to root have its parent changed to root.
- Note that: union-by-height => union-by-rank
Example: Maze generator

Maze Generator: Algorithm

1. Start with walls everywhere, except for the entrance and exit
2. Continually, choose a wall randomly, and knock it down if the cells that the wall separates are not already connected to each other.
3. Repeat this process until the starting and ending cells are connected
4. We have a maze.
Maze Generator: Algorithm

• Start with walls everywhere, except for the entrance and exit.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

- Use the union/find data structure
- Initially, each cell is in its own equivalence class

Initial State: All walls up, all cells in their own set.

\{0\} \{1\} \{2\} \{3\} \{4\} \{5\} \{6\} \{7\} \{8\} \{9\} \{10\} \{11\} \{12\} \{13\} \{14\} \{15\} \{16\} \{17\} \{18\} \{19\} \{20\} \{21\} \{22\} \{23\} \{24\}

Maze Generator: Algorithm

2. Continually, choose a wall randomly, and knock it down if the cells that the wall separates are not already connected to each other.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

At some point in the algorithm: several wall down, sets have merged. If at this point the wall between squares 8 and 13 is randomly selected, this wall is not knocked down, because 8 and 13 are already connected.

\{0,1\} \{2\} \{3\} \{4,6,7,8,9,13,14\} \{5\} \{10,11,15\} \{12\} \{16,17,18,22\} \{19\} \{20\} \{21\} \{23\} \{24\}
Maze Generator: Algorithm

3. Repeat this process until the starting and ending cells are connected

![Diagram of maze generation process]

- Performing two find operations → 18 and 13 are in different sets.
- Therefore, performing union to combine them.

Wall between squares 18 and 13 is randomly selected. The wall is knocked down, because 18 and 13 are not already connected. Their sets are merged.

(0, 1) {2} {3} {4, 6, 7, 8, 9, 13, 14, 16, 17, 18, 22, 5} {10, 11, 15} {19} {20} {21} {23} {24}

---

Maze Generator: Algorithm

4. We have a maze

![Diagram of completed maze]

- Running Time: \(O(N \log N)\)

Eventually, 24 walls are knocked down. All elements are in the same set.

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24)