CS 225

Data Structures

Nov. 10 – Disjoint Sets Implementation
Wade Fagen-Ulmschneider
Disjoint Sets ADT

• Maintain a collection $S = \{s_0, s_1, \ldots, s_k\}$

• Each set has a representative member.

• API:
  void makeSet(const T & t);
  void union(const T & k1, const T & k2);
  T & find(const T & k);
Implementation #1

Find(k):

Union(k1, k2):
Implementation #2

• We will continue to use an array where the index is the key

• The value of the array is:
  • -1, if we have found the representative element
  • The index of the parent, if we haven’t found the rep. element

• We will call these UpTrees:
# UpTrees

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

```
0 1 2 3
-1 -1 -1 -1
```
Disjoint Sets

2 5 9

7

0 1 4 8

3 6

0 1 2 3 4 5 6 7 8 9

4 8 5 6 -1 -1 -1 -1 4 5
Disjoint Sets Find

Running time?

What is the ideal UpTree?

```cpp
int DisjointSets::find() {
    if ( s[i] < 0 ) { return i; }
    else { return _find( s[i] ); }
}
```
Disjoint Sets Union

```cpp
void DisjointSets::union(int r1, int r2) {
}
```
Disjoint Sets – Union

```
0  1  2  3  4  5  6  7  8  9 10 11
6  6  6  8  -1 10  7  -1  7  7  4  5
```
Disjoint Sets – Smart Union

**Idea:** Keep the height of the tree as small as possible.

**Idea:** Minimize the number of nodes that increase in height.

Both guarantee the height of the tree is: _____________.

**Union by height**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Union by size**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Disjoint Sets Find

```cpp
int DisjointSets::find(int i) {
    if ( s[i] < 0 ) { return i; }
    else { return _find( s[i] ); }
}
```

```cpp
void DisjointSets::unionBySize(int root1, int root2) {
    int newSize = arr_[root1] + arr_[root2];
    // If arr_[root1] is less than (more negative), it is the larger set;
    // we union the smaller set, root2, with root1.
    if ( arr_[root1] < arr_[root2] ) {
        arr_[root2] = root1;
        arr_[root1] = newSize;
    }
    // Otherwise, do the opposite:
    else {
        arr_[root1] = root2;
        arr_[root2] = newSize;
    }
}
```
Path Compression
Disjoint Sets Analysis

The **iterated log** function:

*The number of times you can take a log of a number.*

\[
\log^*(n) =
\begin{align*}
0 & , \ n \leq 1 \\
1 + \log^*(\log(n)) & , \ n > 1
\end{align*}
\]

What is \(\log^*(2^{65536})\)?
Disjoint Sets Analysis

In an Disjoint Sets implemented with smart unions and path compression on find:

Any sequence of $m$ union and find operations result in the worse case running time of $O(\_\_\_\_\_\_\_\_)$, where $n$ is the number of items in the Disjoint Sets.
Exam 9 (theory, trees) is ongoing!
More Info: https://courses.engr.illinois.edu/cs225/fa2017/exams/

MP6: One week MP*
Due Monday, Nov. 17 at 11:59pm

Lab: lab released today
Due Sunday, Nov. 12 at 11:59pm

POTD
Every Monday-Friday – Worth +1 Extra Credit /problem (up to +40 total)