Mat 3770
Week 10

Spring 2014

Homework

Due date	Tucker	Rosen
3/18	3.3	10.4, 10.5
3/20	Heapify	worksheet

The Union-Find Data Structure

Given a collection of disjoint sets $S = \{s_1, s_2, \ldots, s_k\}$, we need the operations:

- `Find(S, x)` : return the set ID of the set containing x
- $\texttt{Merge}(S, s_i, s_j)$: combine s_i and s_j into a single set

Implementation:

```
Assume set elements are {1, ..., n}
Use array S[1..n] where
          S[i] = name of the set containing i
```

First Attempt

Try 1. Let the name of the set be the smallest element in the set.

Example. Suppose we have merged several sets and currently have:

$$
\begin{array}{lll}
s_1 = \{1, 5\} & s_2 = \{2, 3, 7, 8\} & s_3 = \{\} \\
s_4 = \{4, 9, 10\} & s_5 = \{\} & s_6 = \{6\} \\
s_7 = \{\} & s_8 = \{\} & s_9 = \{\} \\
s_{10} = \{\} & &
\end{array}
$$

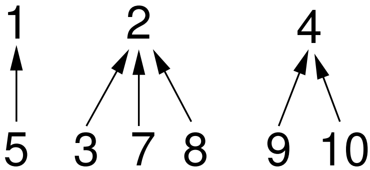
Then set `S` would contain:

`vertex number:`	1	2	3	4	5	6	7	8	9	10
`set ID:`	1	2	2	4	1	6	2	2	4	4

Set form:

`vertex number:`	1	2	3	4	5	6	7	8	9	10
`set ID:`	1	2	2	4	1	6	2	2	4	4

Represented visually:



```
Find_1 (S, x) : integer
   return S[x]
end
```

Find_1 time: O(1)

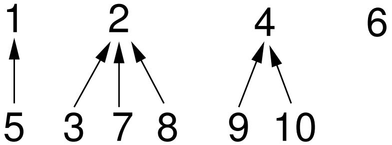
```
Merge_1 (S, a, b)
   if a > b then swap(a,b)     // now a <= b
   for i = 1 to n              // fix names
      if S[i] is b
         then S[i] = a
end
```

Merge_1 time: O(n)

What happens when sets 2 and 4 are merged?

`Merge(S, 2, 4)`

vertex number:	1	2	3	4	5	6	7	8	9	10
set ID:	1	2	2	4	1	6	2	2	4	4



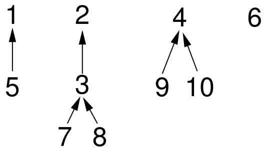
Second Attempt

Try 2. Don't always require `S[x]` to be the name of the set containing `x`. Instead:

- `S[x]` is the name of the set containing `x` if `x` is the smallest element in its set
- otherwise it's `y`, where `y < x` and `x` and `y` are in the same set.

Example

Using the same sets as before, with sets 7 and 8 merged with set 3 before it is merged with set 2:



Then set `S` would contain:

`vertex number:`	1	2	3	4	5	6	7	8	9	10
`set ID:`	1	2	2	4	1	6	3	3	4	4

```
Find_2 (S, x) : integer
   while (S[x] isn't x)
      x = S[x]
   return x
end
```

Find_2 time: Θ(height of tree containing x)

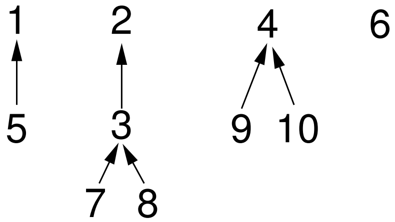
```
Merge_2 (S, a, b)
// Note:  a and b cannot be just any elements,
//    they must be set names
   if a < b then S[b] = a
            else S[a] = b
end
```

Merge_2 time: $\Theta(1)$

What happens when sets 2 and 4 are merged in this case?

`Merge(S, 2, 4)`

vertex number:	1	2	3	4	5	6	7	8	9	10
set ID:	1	2	2	4	1	6	3	3	4	4



In **worst case**, what is the height of the tree containing x?

Third Attempt

Try 3. Keep the tree height to logarithmic size.

Idea: Balancing

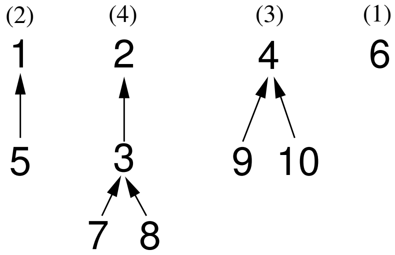
- keep a list of tree **sizes**
- merge the **smaller** tree into the **bigger** tree

Note: Size information only needs to be maintained at the root of each tree.

Example

Same as Second attempt, but with addition of size information:

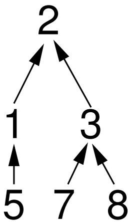
vertex number:	1	2	3	4	5	6	7	8	9	10
set ID:	1	2	2	4	1	6	3	3	4	4



Merge_3 (S, 1, 2)

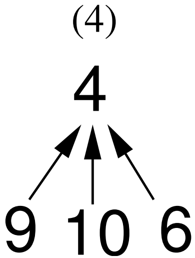
vertex number:	1	2	3	4	5	6	7	8	9	10
set ID:	2^*	2	2	4	1	6	3	3	4	4

(6)



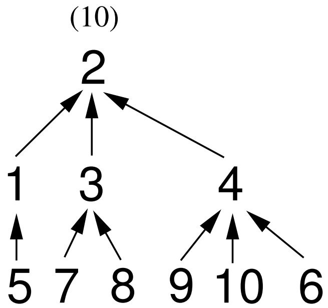
Merge_3 (S, 4, 6)

vertex number:	1	2	3	4	5	6	7	8	9	10
set ID:	2	2	2	4	1	4*	3	3	4	4



Merge_3 (S, 2, 4)

`vertex number:`	1	2	3	4	5	6	7	8	9	10
`set ID:`	2	2	2	2^*	1	4	3	3	4	4



Theorem. Each tree has height h $\leq \log_2(\mathit{size})$,
i.e., $2^h \leq \mathit{size}$.

(Proof is by induction on the number of unions)

```
Find_3 (S, x) : integer     // same as Find_2
   while (S[x] isn't x)
      x = S[x]
   return x
end
```

Find_3 time: Θ(height of tree containing x) $= \Theta(\log n)$

```
Merge_3 (S, a, b)
   if size[a] <= size[b]    // merge smaller
      S[a] = b              //   into larger
      size[b] += size[a]
   else
      S[b] = a
      size[a] += size[b]
end
```

Merge_3 time: $\Theta(1)$

Thus, any collection of k union–find operations takes at most $O(k \log n)$ time.

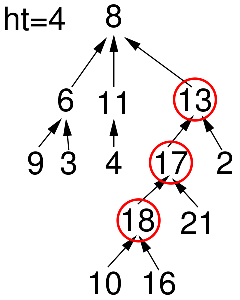
But, **wait**! That's not all!

“*Ginzu*” Path Compression!

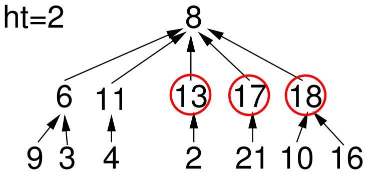
Suppose that whenever we do a **find** operation, we point every visited node toward the root (i.e., the set name element)?

Example: `Find(S, 18)`:

Before:



After:



Theorem. If path compression and balancing (merge by size) are used, then the total number of steps needed for *any* sequence of k operations is $O(k \log^* n)$, where $\log^* n$ is the **iterated logarithmic** function defined as follows:

- $\log^* 1 = \log^* 2 = 1$
- $\log^* n = 1 + \log^* \lceil \log_2 n \rceil$

Consider...

Note: $2^{16} = 65,536$

$$
\begin{aligned}
\log^* 2^{65,536} &= 1 + \log^* \lceil \log_2 2^{65,536} \rceil \\
&= 1 + \log^* \lceil \log_2 2^{2^{16}} \rceil \\
&= 1 + \log^* 2^{16} \\
&= 1 + (1 + \log^* \lceil \log_2 2^{16} \rceil) \\
&= 2 + \log^* 16 \\
&= 2 + \log^* 2^4 \\
&= 2 + (1 + \log^* \lceil \log_2 2^4 \rceil) \\
&= 3 + \log^* 4 \\
&= 3 + (1 + \log^* \lceil \log_2 2^2 \rceil) \\
&= 4 + \log^* 2 \\
&= 4 + 1 = 5
\end{aligned}
$$

A **very** slow growing function!