

Lecture 24: Disjoint Sets

Agenda:

- 2nd implementation — forest of rooted trees (review)
- Improvements:
 - Union by rank `rUnion`
 - Compressed find `cFind`

Reading:

- Textbook pages 505 – 522

2nd implementation — forest of rooted trees

- Forest of rooted trees:
 - elements of a set $\longleftrightarrow$ nodes in the rooted trees
 - representative of a set $\longleftrightarrow$ root of the tree
 - each node needs only ‘parent’ $\longrightarrow$ implement via an array
 - $P(x)$ — parent of x , for $x = 1, 2, \dots, n$

- procedure MakeSet(x) ****initialize parent for x**

$$P(x) \leftarrow x$$

- procedure Find(x) ****return root of the tree containing x**

```

while  $P(x) \neq x$  do
   $x \leftarrow P(x)$ 
return  $x$ 

```

- procedure Union(x, y) ****make root of x 's tree
a child of root of y 's tree

```

 $rx \leftarrow \text{Find}(x)$ 
 $ry \leftarrow \text{Find}(y)$ 
 $P(rx) \leftarrow ry$ 

```

- Running time per operation — $\Theta(n)$

Union by rank — `rUnion`:

- Observation: running time affected by the depth of the element(s) in both `find` and `union`
- Goal: to reduce the height of the tree
- Tree height determined by how we do the `union`
- Improvement — union by rank (denoted as `rUnion`)
 - idea: when union two sets, root of shorter tree becomes a child of the root of higher tree
 - rank of an element x — height of subtree rooted at x
 - pseudocode:

```

procedure rUnion( $x, y$ )    **make smaller rank root
                           **child of the other root
     $rx \leftarrow \text{Find}(x)$ 
     $ry \leftarrow \text{Find}(y)$ 
    if rank( $rx$ ) > rank( $ry$ ) then
         $P(ry) \leftarrow rx$ 
    else
         $P(rx) \leftarrow ry$ 
        if rank( $rx$ ) = rank( $ry$ ) then
            rank( $ry$ )  $\leftarrow$  rank( $ry$ ) + 1
    
```

- Need to initialize the rank for every element:

```

procedure MakeSet( $x$ ) **initialize parent for  $x$ 
     $P(x) \leftarrow x$ 
    rank( $x$ )  $\leftarrow$  0
    
```

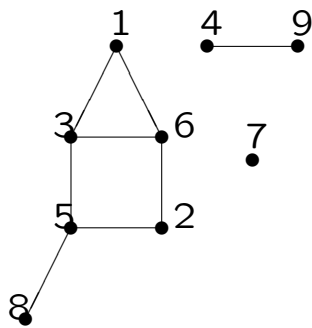
Lecture 23: Disjoint Sets

Finding connected components of a graph:

- procedure ConnectedComponents(G)

 for each vertex $v \in V(G)$ do
 MakeSet(v)
 for each edge $(x, y) \in E(G)$ do
 if not SameComponent(x, y, G) then
 rUnion(x, y) ** previously, Union(x, y)
- procedure SameComponent(x, y, G)

 return Find(x) = Find(y)
- An example:



$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$E = \{\{1, 3\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{3, 5\}, \{3, 6\}, \{4, 9\}, \{5, 8\}\}$$

Lecture 23: Disjoint Sets

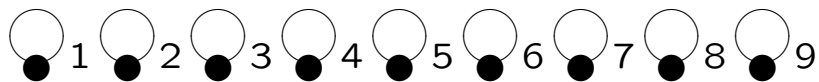
Finding connected components of a graph:

- Graph $G = (V, E)$:

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$E = \{\{1, 3\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{3, 5\}, \{3, 6\}, \{4, 9\}, \{5, 8\}\}$$

- After MakeSets:



Lecture 23: Disjoint Sets

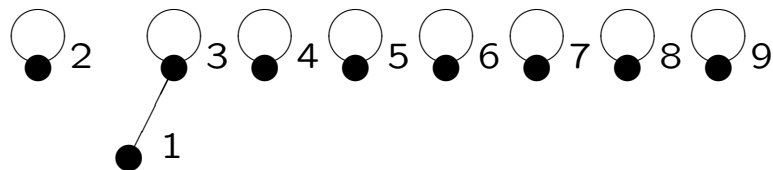
Finding connected components of a graph:

- Graph $G = (V, E)$:

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$E = \{\{1, 3\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{3, 5\}, \{3, 6\}, \{4, 9\}, \{5, 8\}\}$$

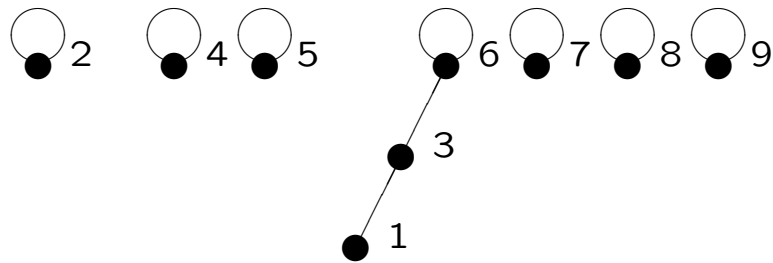
- After considering edge $(1, 3)$:



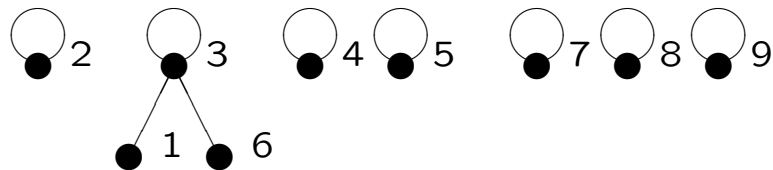
Lecture 23: Disjoint Sets

Finding connected components of a graph:

- Graph $G = (V, E)$:
 $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 $E = \{\{1, 3\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{3, 5\}, \{3, 6\}, \{4, 9\}, \{5, 8\}\}$
- After considering edge $(1, 6)$:
by original `union`:



by `rUnion`:



Lecture 23: Disjoint Sets

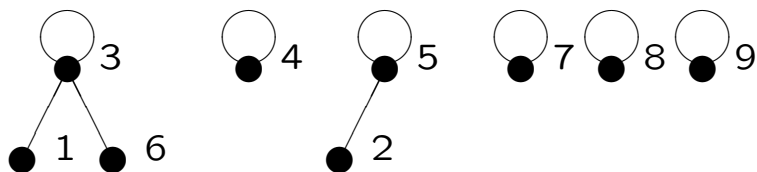
Finding connected components of a graph:

- Graph $G = (V, E)$:

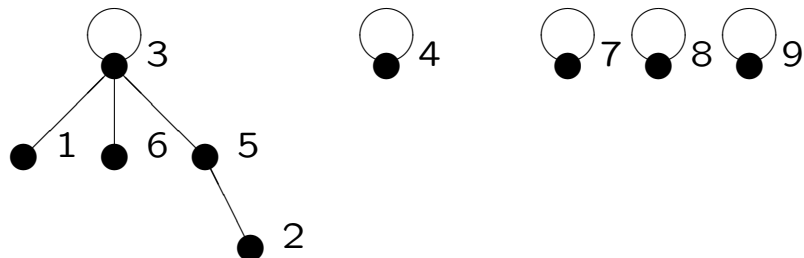
$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$E = \{\{1, 3\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{3, 5\}, \{3, 6\}, \{4, 9\}, \{5, 8\}\}$$

- After considering edge $(2, 5)$:



- After considering edge $(2, 6)$:



Lecture 23: Disjoint Sets

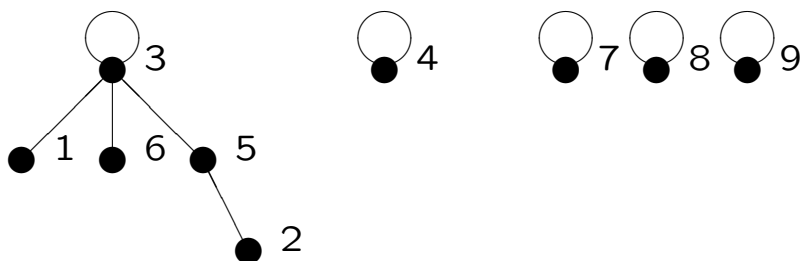
Finding connected components of a graph:

- Graph $G = (V, E)$:

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

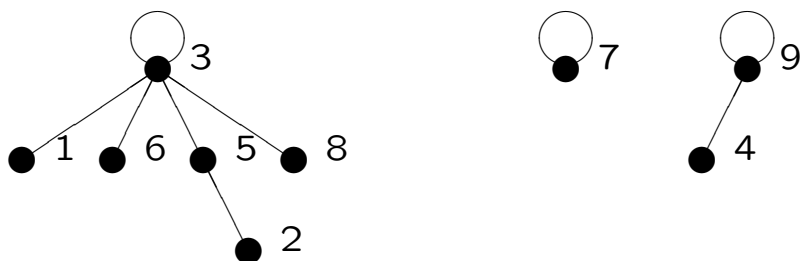
$$E = \{\{1, 3\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{3, 5\}, \{3, 6\}, \{4, 9\}, \{5, 8\}\}$$

- After considering edges $(3, 5)$ and $(3, 6)$



(no change since 3, 5, 6 are already in a same component):

- After considering edge $(4, 9)$ and $(5, 8)$:



- Therefore, there are 3 connected components

Lecture 23: Disjoint Sets

Analysis of (rUnion + Find):

- n MakeSet and $(m - n)$ Union/Find
- Each MakeSet — $\Theta(1)$ time
- Each Find — $\Theta(\text{depth}(x))$ time
 - T_x — the tree containing x
 - $\# \text{elements in } T_x \geq 2^{\text{height}(T_x)}$ — proof? by induction
 - so, $\text{depth}(x) \leq \text{height}(T_x) \leq \lg(\# \text{elements in } T_x) \leq \lg n$
- Each rUnion — $\Theta(\text{depth}(x) + \text{depth}(y))$ time
- Worst case:
 $\Theta(n + (m - n) \lg n) = \Theta(m \lg n)$ time (assuming $m \gg n$)
- On average, $\Theta(\lg n)$ per operation

Compressed find — cFind:

- Another observation — to make the trees as short as possible
- Idea:
 - during the time we Find the root of the tree containing x
 - we pass all the elements on the x -to-root path
 - re-examine them and make their parents the root
 - Find **TWICE** to make the tree shorter
- pseudocode (non-recursive):

```

procedure cFind( $x$ )

     $t \leftarrow x$ 
    while  $P(t) \neq t$  do           **find the root
         $t \leftarrow P(t)$ 
     $root \leftarrow t$ 
     $t \leftarrow x$ 
    while  $P(t) \neq t$  do         **change the parent to root
         $x \leftarrow t$ 
         $t \leftarrow P(t)$ 
         $P(x) \leftarrow root$ 
    return  $root$ 

```

- pseudocode (recursive):

```

procedure cFind( $x$ )

    if  $P(x) \neq x$  do             ** $x$  isn't the root
         $P(x) \leftarrow \text{cFind}(P(x))$ 
    return  $P(x)$ 

```

Lecture 23: Disjoint Sets

Analysis of (rUnion + cFind):

- n MakeSet and $(m - n)$ Union/Find
- Each MakeSet — $\Theta(1)$ time
- Each cFind — $\Theta(\text{depth}(x))$ time
 - $\lg^* n$ — the smallest t such that $2^{2^{\dots^{2^2}}} \geq n$ where there are t 2's

n	2 ...	4 ...	16 ...	65536 ...	2^{65536} ...
$\lg^* n$	1	2	3	4	5

- $\text{depth}(x)$ grows more slowly than $\lg^* n$

proof not required ... just memorize it ...

- Each rUnion — $\Theta(\text{depth}(x) + \text{depth}(y))$ time
- Worst case:
 $O(n + (m - n) \lg^* n) = O(m \lg^* n)$ time (assuming $m \gg n$)
- On average, $O(\lg^* n)$ per operation — almost constant

Lecture 23: Disjoint Sets

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	disjoint sets?
☐	☐	☐	3 operations
☐	☐	☐	forest of rooted trees
☐	☐	☐	finding connected components
☐	☐	☐	union by rank
☐	☐	☐	running time analysis
☐	☐	☐	compressed find
☐	☐	☐	running time