About this lecture

- Data Structure for Disjoint Sets
  - Support Union and Find operations

- Various Methods:
  1. Linked List (this lecture)
  2. Union by Size
  3. Union by Rank
  4. Union by Rank + Path Compression
Maintaining Disjoint Set

- In some applications, especially in algorithms relating to graphs, we often have a set of elements, and want to maintain a dynamic partition of them.
  - I.e., the partition changes over time.

- Our target corresponds to maintaining dynamic disjoint sets of the elements.
Maintaining Disjoint Set

• Let $\Sigma = \{ S_1, S_2, \ldots, S_k \}$ be a collection of dynamic disjoint sets of the elements
• Let $x$ and $y$ be any two elements
• We want to support:
  \begin{align*}
  \text{Make-Set}(x) & : \text{create a set containing } x \\
  \text{Find}(x) & : \text{return which set } x \text{ belongs} \\
  \text{Union}(x,y) & : \text{merge the sets containing } x \\
  & \quad \text{and containing } y \text{ into one}
  \end{align*}
Example Application: Finding Connected Components

Step 0: Begin with the input graph
Example Application: Finding Connected Components

Step 1: Make-Set(v) for each vertex v

current \Sigma: \{ \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\} \}
Step 2: Visit each edge \((u,v)\), perform \(\text{Union}(u,v)\)

Example Application: Finding Connected Components

\[\text{current } \Sigma: \{ \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\} \} \]
Step 2: Visit \((a,b)\)

\[
\begin{align*}
\text{current } \Sigma & : \{\{a,b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}\} \\
\end{align*}
\]

Step 2: Visit \((c,d)\)

\[
\begin{align*}
\text{current } \Sigma & : \{\{a,b\}, \{c,d\}, \{e\}, \{f\}, \{g\}, \{h\}\} \\
\end{align*}
\]
Step 2: Visit (e,f)

current $\Sigma$: \{ {a,b}, {c,d}, {e,f}, {g}, {h} \}

Step 2: Visit (b,c)

current $\Sigma$: \{ {a,b,c,d}, {e,f}, {g}, {h} \}
Step 2: Visit \((f,g)\)

Step 2: Visit \((b,d)\)

current \(\Sigma\): \(\{\{a,b,c,d\}, \{e,f,g\}, \{h\}\}\)
Example Application: Finding Connected Components

After Step 2 (when all edges visited):
Each Disjoint Set ⇔ Connected Component

current $\Sigma$: $\{ \{a,b,c,d\}, \{e,f,g\}, \{h\} \}$
Remarks

• To facilitate $\text{Find}(x)$, each set usually chooses one of its element as a representative
  $\Rightarrow \text{Find}(x)$ returns the representative element of the set where $x$ belongs

• To check if $x$ and $y$ belong to the same set, we can just check if
  $\text{Find}(x) == \text{Find}(y)$
Disjoint Set with Linked List

- A simple way to maintain disjoint sets is by using linked lists:
  - Each set ↔ a separate linked list
  - Representative ↔ head element of list

- To facilitate Find and Union:
  - each element in the list has an extra pointer that points at head element
  - each list has a pointer to the tail
E.g., disjoint sets \( \{ \{a,b,c,d\}, \{e,f,g\}, \{h\} \} \) is stored by:
Disjoint Set with Linked List

• To perform Union(x,y), we join the lists containing x and containing y, one list after the other, and update the pointers of the latter list.

• E.g. Union(g,h) in previous example gives:
Disjoint Set with Linked List

• In the worst-case:
  
  Make-Set or Find: $\Theta(1)$ time
  Union: $\Theta(n)$ time

$\Rightarrow$ $m$ operations on $n$ elements:
  $\Theta(m + n^2)$ time
Disjoint Set with Linked List

- Let us apply a weighted-union heuristic:
  To perform Union, we merge lists with longer one first, followed by shorter list

- No change in worst-case time, but ...

- $m$ operations: $\Theta(m + n \log n)$ time

Reason: The time to perform Union is from changing head pointer of each element in the latter list.
With the heuristic, each element changes head pointer at most $\log n$ times (why??)