CS2351
Data Structures

Tutorial 3:
Data Structures for Disjoint Sets
About this lecture

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

• Various Methods:
  1. Union by Size
  2. Union by Rank
  3. 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:
  Make-Set($x$): create a set containing $x$
  Find($x$): return which set $x$ belongs
  Union($x$, $y$): merge the sets containing $x$ and containing $y$ into one
Example Application: Finding Connected Components

Step 0: Begin with the input graph

\[
\begin{align*}
\text{Step 0: Begin with the input graph} \\
\text{Example Application: Finding Connected Components} \\
\text{Step 0: Begin with the input graph}
\end{align*}
\]
Example Application: Finding Connected Components

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

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

Step 2: Visit each edge \((u,v)\), perform \(\text{Union}(u,v)\)

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

\[
\text{current } \Sigma: \{ \{a,b\}, \{c,d\}, \{e,f\}, \{g\}, \{h\} \}
\]

Step 2: Visit (b,c)

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

\[
\text{current } \Sigma: \{ \{a,b,c,d\}, \{e,f,g\}, \{h\} \}
\]

Step 2: Visit (b, d)

\[
\text{current } \Sigma: \{ \{a,b,c,d\}, \{e,f,g\}, \{h\} \}
\]
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
  \[ \Rightarrow \text{Find}(x) \text{ returns the representative element of the set where } x \text{ belongs} \]

• To check if \( x \) and \( y \) belong to the same set, we can just check if
  \[ \text{Find}(x) == \text{Find}(y) \]
Disjoint-Set Forest

• One popular method to maintain disjoint sets is by a forest
  • Each set $\Leftrightarrow$ a separate rooted tree
  • Representative $\Leftrightarrow$ root of tree
Example

Current dynamic sets: \{ \{a,b,c,d\}, \{e,f,g\}, \{h\} \}
Disjoint-Set Forest

- To perform $\text{Union}(x,y)$, we join the trees containing $x$ and containing $y$, by linking their roots.
- E.g., $\text{Union}(f,h)$ in previous example gives:

![Disjoint-Set Forest Diagram]

```
   g
  /|
 / |
/  |
f   h
  \
   e
```
Disjoint-Set Forest

- Let $H_{\text{max}} = \text{max height of all trees}$
- In the worst-case:
  
  \begin{align*}
  \text{Make-Set} & : \ \Theta(1) \text{ time} \\
  \text{Find or Union} & : \ \Omega(H_{\text{max}}) \text{ time}
  \end{align*}

  \Rightarrow \quad m \text{ operations on } n \text{ elements :} \\
  \text{worst-case } \Theta(mn) \text{ time}
Union By Size

- Let us apply a union-by-size heuristic:

To perform Union, we link root of the smaller tree to root of the larger tree

- \( H_{\text{max}} = \mathcal{O}(\log n) \) (how to prove??)
- \( m \) operations: \( \Theta(m \log n) \) time
Union By Rank

- A similar heuristic is called union-by-rank
- Each node keeps track of its rank – an upper bound on the height of the node
- In a single-node tree (created by Make-Set)
  
  rank of root = 0

To perform Union, we link root with smaller rank to root with larger rank
Union By Rank

• Rank needs not be very accurate
  • as long as it always gives an upper bound of height is enough

• When Union is performed, only the rank of the roots may change:
  • If both roots have same rank
    ➔ rank of new root increases by 1
  • Else, no change
Example of Union by Rank

Before Union

After Union(c,f)

? = rank
Union By Rank

• Let $H_{\text{max}} = \text{max height of all trees}$
  
  $\Rightarrow H_{\text{max}} = O(\log n)$ (how to prove??)

  $\Rightarrow m$ operations : $\Theta(m \log n)$ time

• So, union by rank is no better than union by size, but ...
Path Compression

• The closer a node to its root, the faster the Find or Union operation

• When we perform Find(x), we will need to find the root of the tree containing x
  ➔ will access every ancestor of x

• why don’t we make all these ancestors of x closer to the root now?
  (Because no increase in asymptotic performance !!!)
Example of Path Compression

Before Find(x)

After Find(x)
Union by Rank + Path Compression

• If Union(x,y) is always performed by first Find(x), Find(y), and then linking the roots, then by combining union-by-rank (at Union) and path compression (at Find and Union):

\[ m \text{ operations: } \Theta(m \alpha(n)) \text{ time} \]

Inverse Ackermann
(in practice, at most 4)
Finding Connected Components

• Recall: To find connected components of a graph $G$ with $n$ vertices and $m$ edges
  • there are $n$ Make-Set and $m$ Find or Union operations

• Which scheme for dynamic disjoint sets gives the best running time (theoretically)?)
  Ans. Depends on $m$ (why?)