CSE 373: Data Structures and Algorithms

Disjoint Sets

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...
Final exam info (logistics, topics covered, and practice material) is on course website.

Homework 6 is due this Friday at noon.

Homework 7 will be posted this Friday evening.
- Fill out the team sign up form by tomorrow 5pm to get the repo in time.
- Fill out the partner pool form by tomorrow 5pm to get assigned to a partner.
Today

Trees and Forests
Kruskal’s algorithm
Disjoint Set ADT
Trees and Forests

A tree is an undirected, connected, and acyclic graph.

A graph, however, can have unconnected vertices, or multiple trees.

Collection of trees is called a forest.

A forest is any undirected and acyclic graph.
- By definition a tree is a forest
Worksheet question 1

\[ n \text{ Edges } = n - 1 \]

\[ |E| \leq \frac{1}{2} n^2 \]
## Worksheet question 2a

<table>
<thead>
<tr>
<th>Step</th>
<th>Components</th>
<th>Edge</th>
<th>Include?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${A, B, C, D}$, ${E, F}$</td>
<td>$AB$</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>${A, B, C, D}$, ${E, F}$</td>
<td>$CD$</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>${A, B, C, D}$, ${E, F}$</td>
<td>$EF$</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>${A, B, C, D}$, ${E, F}$</td>
<td>$AD$</td>
<td>No</td>
</tr>
</tbody>
</table>
Worksheet question 2b

1: function Kruskal(Graph G)
2:     initialize each vertex to be a component
3:     sort all edges by weight
4:     for each edge (u, v) in sorted order do
5:         if u and v are in different components then
6:             add edge (u,v) to the MST
7:             update u and v to be in the same component
8:         end if
9:     end for
10: end function
Kruskal’s Algorithm

1: function Kruskal(Graph G)
2:     initialize each vertex to be a component
3:     sort all edges by weight
4:     for each edge (u, v) in sorted order do
5:         if u and v are in different components then
6:             add edge (u,v) to the MST
7:             update u and v to be in the same component
8:         end if
9:     end for
10: end function
Minimum spanning forest

Given a forest, find a set of minimum spanning trees for each tree in the given forest.

Question: Which algorithm (Prims or Kruskals) would you use to find a minimum spanning forest?
Disjoint Set ADT
New ADT

Set ADT

**state**
- Set of elements
  - Elements must be unique!
  - No required order

**behavior**
- `create(x)` - creates a new set with a single member, x
- `add(x)` - adds x into set if it is unique, otherwise add is ignored
- `remove(x)` - removes x from set
- `size()` - returns current number of elements in set

Disjoint-Set ADT

**state**
- Set of Sets
  - **Disjoint**: Elements must be unique across sets
  - No required order
  - Each set has representative

**behavior**
- `makeSet(x)` – creates a new set within the disjoint set where the only member is x. Picks representative for set
- `findSet(x)` – looks up the set containing element x, returns representative of that set
- `union(x, y)` – looks up set containing x and set containing y, combines two sets into one. Picks new representative for resulting set
Example

new()
makeSet(a)
makeSet(b)
makeSet(c)
makeSet(d)
makeSet(e)
findSet(a)
findSet(d)
union(a, c)
Example

new()
makeSet(a)
makeSet(b)
makeSet(c)
makeSet(d)
makeSet(e)
findSet(a)
findSet(d)
union(a, c)
union(b, d)
Example

new()
makeSet(a)
makeSet(b)
makeSet(c)
makeSet(d)
makeSet(e)
findSet(a)
findSet(d)
union(a, c)
union(b, d)

\[
\text{findSet}(a) = \text{findSet}(c)
\]

\[
\text{findSet}(a) \neq \text{findSet}(d)
\]
Implementation

Disjoint-Set ADT

**state**
- Set of Sets
  - **Disjoint**: Elements must be unique across sets
  - No required order
  - Each set has representative

**behavior**
- `makeSet(x)` – creates a new set within the disjoint set where the only member is x. Picks representative for set
- `findSet(x)` – looks up the set containing element x, returns representative of that set
- `union(x, y)` – looks up set containing x and set containing y, combines two sets into one. Picks new representative for resulting set

**TreeDisjointSet<E>**

**state**
- `Collection<TreeSet>` `forest`
- `Dictionary<NodeValues, NodeLocations>` `nodeInventory`

**behavior**
- `makeSet(x)` – creates a new tree of size 1 and add to our forest
- `findSet(x)` – locates node with x and moves up tree to find root
- `union(x, y)` – append tree with y as a child of tree with x

**TreeSet<E>**

**state**
- `SetNode` `overallRoot`

**behavior**
- `TreeSet(x)`
- `add(x)`
- `remove(x, y)`
- `getRep()` – returns data of overallRoot

**SetNode<E>**

**state**
- `E` `data`
- `Collection/SetNode` `children`

**behavior**
- `SetNode(x)`
- `addChild(x)`
- `removeChild(x, y)`
Implement `makeSet(x)`

- `makeSet(0)`
- `makeSet(1)`
- `makeSet(2)`
- `makeSet(3)`
- `makeSet(4)`
- `makeSet(5)`

**Worst case runtime?**

\(O(1)\)
Implement union(x, y)

union(3, 5)
Implement union(x, y)

union(3, 5)
union(2, 1)
Implement union(x, y)

union(3, 5)
union(2, 1)
union(2, 5)
Implement union(x, y)

union(3, 5)
union(2, 1)
union(2, 5)
**Implement $\text{findSet}(x)$**

- $\text{findSet}(0)$
- $\text{findSet}(3)$
- $\text{findSet}(5)$

**TreeDisjointSet$\langle E \rangle$**

**state**

- Collection$\langle$TreeSet$\rangle$ forest
- Dictionary$\langle$NodeValues, NodeLocations$\rangle$ nodeInventory

**behavior**

- $\text{makeSet}(x)$ - create a new tree of size 1 and add to our forest
- $\text{findSet}(x)$ - locates node with $x$ and moves up tree to find root
- $\text{union}(x, y)$ - append tree with $y$ as a child of tree with $x$

**Worst case runtime?**

$O(n)$

**Worst case runtime of union?**

$O(n)$
Improving union

Problem: Trees can be unbalanced

Solution: Union-by-rank!
- let rank(x) be a number representing the upper bound of the height of x so rank(x) ≥ height(x)
- Keep track of rank of all trees
- When unioning make the tree with larger rank the root
- If it’s a tie, pick one randomly and increase rank by one
Worksheet question 3

Given the following disjoint-set what would be the result of the following calls on union if we add the “union-by-rank” optimization. Draw the forest at each stage with corresponding ranks for each tree.

union(2, 13)
union(4, 12)
union(2, 8)
Worksheet question 3

Given the following disjoint-set what would be the result of the following calls on union if we add the “union-by-rank” optimization. Draw the forest at each stage with corresponding ranks for each tree.

union(2, 13)
union(4, 12)
union(2, 8)

Does this improve the worst case runtimes?

findSet is more likely to be $O(\log(n))$ than $O(n)$
Improving findSet()

**Problem:** Every time we call findSet() you must traverse all the levels of the tree to find representative

**Solution: Path Compression**
- Collapse tree into fewer levels by updating parent pointer of each node you visit
- Whenever you call findSet() update each node you touch’s parent pointer to point directly to overallRoot

`findSet(5)`

`findSet(4)`

Does this improve the worst case runtimes?
`findSet` is more likely to be $O(1)$ than $O(\log(n))$