From Last Time

1. Can prove that path compression alone with n Make-Set ops, f Find-Set ops, and arbitrary 
   \# of unions (up to n-1) cost \( \Theta(n + f(1 + \log_2 f n)) \)

   \# of elements

2. \( O(m \log(n)) \), where m = total \# ops, running time 
   using forest of trees with union-by-rank and path compression proven in §21.4

   — You ARE NOT RESPONSIBLE FOR THIS
Greedy Algorithms

Greedy algorithm: Overall problem solved in series of steps. Choice made at each step looks best at the moment without explicit reference to overall problem. "locally optimal"

Example:
We need to make 99¢ in change with minimum # of coins. We do this with a greedy algorithm automatically.

\[
99\text{¢} = (25\text{¢}) \times 3 + \\
72\text{¢} = 6 \times (10\text{¢}) \times 2 + \\
72\text{¢} = 6 \times (5\text{¢}) + (10\text{¢}) \times 4 \\
3 \text{ quarters} + 2 \text{ dimes} + 4 \text{ pennies}
\]

This greedy algorithm gives correct solution.
Starting with largest coin, take as many as possible without going over.

But...
1. Start with pennies: 99¢ = (1¢) × 99 ⇒ 99 coins
2. If the dime was replaced by an 11¢ piece, make 15¢:
   \[
   \text{greedy: } 15\text{¢} = (11\text{¢}) + (4\text{¢}) \times 1 + (1\text{¢}) \times 4 \Rightarrow 5 \text{ coins}
   \]
   \[
   \text{correct answer: } 15\text{¢} = (1\text{¢}) \times 0 + (5\text{¢}) \times 3 \Rightarrow 3 \text{ coins}
   \]
As greedy algorithms:
- sometimes yield correct solution (globally optimal)
- sometimes do not
  - change made is correct, but minimum number of coins not achieved.

Depends on:
- structure of algorithm (forward/reverse)
- structure of problem (coin values)

We will see that:
1. in some cases can prove greedy algorithm leads to globally optimal solution
2. in other cases greedy solutions lead to good solutions (approximate behavior) that are rapid to find and reasonable, but not guaranteed optimal.

Graphs (CLRS Appendix B.4)
- Graph is a set of vertices (points) connected by edges (lines that join two points) - directed/undirected
- "Weight" is additional information such as distance between vertices (associated with edge)
- "Connected": a path exists between any pair of vertices by traversing (multiple) edges
Minimum Spanning Tree (MST) Problem

Input: A connected, undirected graph \( G = (V, E) \) with weight function \( w: E \rightarrow \mathbb{R} \)

Output: A spanning tree, \( T \) (connecting all vertices) of minimum weight

\[
\omega(T) = \sum_{(u,v) \in E_T} \omega(u,v)
\]

Why is this problem interesting?

1. **Shortest Path Connectivity**
   - For electric circuitry, often need to wire together sets of contacts. Desirable to use minimal amount of wire. \( \rightarrow \) MST problem
   - On a larger scale, to connect multiple sites by telecommunications network, want minimal cost scheme. Weights could be length of wire or cost to install, or other

2. **One form of clustering achieved by cutting longest edges in MST**

Examples

\[
\begin{align*}
\text{not minimal} & \quad \rightarrow \quad \text{minimal} \\
\text{no cycles} & \quad \text{if all edges are possible}
\end{align*}
\]

\( \text{all edges possible} \)
Some properties:  
$|E| = \Theta(V^2)$  
- bounded from above

If graph $G$ connected, then $|E| \geq |V| - 1 \Rightarrow \lg |E| = \Theta(\lg |V|)$  
- bounded from below

Would you think a greedy algorithm would work here?  
Can growing a tree one step at a time (in a greedy manner) lead to a globally optimum smallest weight solution?  

**THE MST PROPERTY**  
Somewhat different from book

**Theorem:** Let $G = (V, E)$ be connected graph with cost function defined on edges. Let $U$ be some proper subset of $V$. If $(u, v)$ is an edge of lowest cost such that $u \in U$ and $v \in V - U$, then there is a "light edge" on MST containing $(u, v)$.

**Proof:** (Every MST satisfies above) (By "cut-and-paste")

Assume the theorem false: no MST that includes $(u, v)$.

Let $T$ be an MST.

- Adding $(u, v)$ introduces a cycle, because $T$ is an MST so already has path from $u$ to $v$.
- There must be another edge from $U$ to $V - U$, $(u', v')$ wlog.
- Adding $(u', v')$ would make there be more than one.

- Deleting edge $(u, v)$ breaks cycle, giving tree $T'$
- $T'$ has weight $\leq T$ because $(u, v)$ was lowest cost edge
- Thus, our assumption is wrong and theorem is true:  

$(u, v) \in$ MST
Now do you think greedy algorithm might work? We can use local information to exclude (and include) edges from growing MSTs.

Size of search space: All graphs - Each edge can be in or out: $2^{|E|}$
- Some will not be trees
- Some will not be connected
- Some will not be minimum cost
- Could enumerate each, evaluate whether connected (disjoint set operations) and compute weight

**Kruskal’s Algorithm**

- Initially $T = (V, \emptyset)$ (vertices but no edges)
- Examine edges of $E$ in "increasing" weight order
  - If edge connects two unconnected components, add edge to $T$
  - Else discard edge and continue (forms cycle)
- Can terminate when all edges in simple connected component

**Example:**

[Diagram showing a graph with labeled vertices and edges]
Correctness of **T**:

- **Loop invariant**

Prior to each iteration, **T** is a subset of a MST

**Initialization**: **T** has no edges, so trivially satisfied

**Maintenance**: Edges are only accepted in loop if part of MST

**Termination**: All edges are examined and added to **T** if in MST, so **T** must be MST

---

Pseudo code for implementation

**MST-Kruskal** (**G, W**) 

1. Initialize edge list to **A** ∈ **G**
2. Make a forest of trees (not only) for the vertices
3. **Sort** the edges of **E** into non-decreasing order by **E** → **O(E log E)**
4. For each edge **(u, v)** ∈ **E** in non-decreasing order
   - **If** **Find-Set**(u) ≠ **Find-Set**(v)
     - **Join** **A** ← **A** ∪ {**u, v**}
   - **Return** **A**

**Analysis**:

- **Find-Set** & **Union** operations plus **O(V)** Make-Sets
- **O(E log E)** for **E** connections
- **O(V log V)** for the initial forest

**Time Complexity**:

- **O(E log E)** for sorting
- **O(V log V)** for initial forest
- **O(E log E)** for each connection

**Overall Complexity**:

- **O(E log E)**