Today: Greedy algorithms & Minimum Spanning Tree (MST)
- MST problem
- optimal substructure
- greedy-choice property
- Prim’s algorithm
- Kruskal’s algorithm

Recall: [Lecture 1]

**Greedy algorithm**: repeatedly make locally best choice/decision, ignoring effect on future
- saw greedy algorithm for scheduling problem
- Dijkstra’s algorithm also ≈ greedy (cf. Bellman-Ford: incremental improvement)
- today: greedy algorithm for graph problem

Tree = connected graph with no cycles
Spanning tree of graph = subset of graph’s edges that form a tree spanning (containing) all vertices
Minimum spanning tree (MST) problem:
given a graph $G=(V,E)$ & edge weights $w: E \rightarrow \mathbb{R}$,
find spanning tree $T \subseteq E$ of minimum weight:
$w(T) = \sum_{e \in T} w(e)$

Example:

Naive algorithm: check all spanning trees
- exponential time

Greedy properties: problems amenable to greedy algorithms usually satisfy:

1. Optimal substructure: optimal solution to problem incorporate optimal solution(s) to subproblem(s)
   - essentially dynamic programming

2. Greedy-choice property: locally optimal choices lead to globally optimal solution
**Optimal substructure for MST:**

If \( e = (u, v) \) is an edge of some MST of \( G = (V, E, w) \):
- contract edge \( e \): merge vertices \( u \) & \( v \)
- if we get multiple copies of an edge, just keep lowest weight:

- if \( T' \) is an MST of \( G' = G/e \) then \( T = T' \cup \{ e \} \) is an MST of \( G \)
- remap edges to decontract \( \xi uv \rightarrow uv \)

**Proof:**
- let \( T^* \) be an MST of \( G \) containing edge \( e \)
  \( \Rightarrow T^*/e \) is a spanning tree of \( G' \)
- \( T' \) is an MST of \( G' \)
  \( \Rightarrow w(T') \leq w(T^*/e) \)
  \( \Rightarrow w(T) = w(T') + w(e) \leq w(T^*/e) + w(e) = w(T^*) \). \( \square \)
Dynamic program attempt:
- guess an edge to put in MST
- contract to get new subproblem
- recurse
- decontract & add e

but # subproblems is exponential ::

greedy technique will make this polynomial! ::
Greedy-choice property for MST:
for any cut \((S, V \setminus S)\)
in graph \(G = (V, E, w)\),
any least-weight crossing edge
\(e = \{u, v\}\) with \(u \in S \& v \notin S\)
is in some MST of \(G\)

Proof: cut & paste argument
- consider an MST \(T\) of \(G\)
  - \(T\) has a path from \(u\) to \(v\)
  - \(u \in S \& v \notin S\), so the path has some edge \(e' = \{u', v'\}\) with \(u' \in S \& v' \notin S\)
  - then \(T' = T \setminus \{e'\} \cup \{e\}\)
is a spanning tree of \(G\)
\& \(w(T') = w(T) - w(e') + w(e)\)
- but \(e\) is a least-weight edge crossing \((S, V \setminus S)\)
  \(\Rightarrow w(e) \leq w(e')\)
  \(\Rightarrow w(T') \leq w(T)\)
  \(\Rightarrow T'\) is a MST too.

* modification only touches edge(s) crossing \((S, V \setminus S)\)

Two algorithms based on different choices of cut \((S, V \setminus S)\).
Prim's algorithm: start with $|S| = 1$ & grow from there

- maintain priority queue $Q$ on $V \setminus S$, where $v.\text{key} = \min \{ w(u,v) \mid u \in S \}$
- initially $Q$ stores $V$ ($S = \emptyset$)
- $s.\text{key} = \emptyset$ for arbitrary start vertex $s \in V$
- for $v \in V \setminus \{s\}$: $v.\text{key} = \infty$
- until $Q$ empty:
  - $u = \text{Extract-Min}(Q)$ (add $u$ to $S$)
  - for $v \in \text{Adj}[u]$:
    - if $v \in Q$ ($v \notin S$) & $w(u,v) < v.\text{key}$:
      - $v.\text{key} = w(u,v)$ ← Decrease-Key
      - $v.\text{parent} = u$
  - return $\{ \{ v \mid v.\text{parent} \neq \emptyset \mid v \in V \setminus \{s\} \}$

Correctness:

- invariant: $v \notin S \Rightarrow v.\text{key} = \min \{ w(u,v) \mid u \in S \}$
- invariant: tree $T_S$ within $S \subseteq \text{MST of } G$
  - assume by induction: $\text{MST } T^* \supseteq T_S$
  - $S \rightarrow S^* = S \cup \{e\}$
    - where $e$ is a least-weight edge crossing cut $(S, V \setminus S)$
  - greedy cut & paste ⇒ can modify $T^*$ to include $e$ without removing $T_S$
    - new $T^* \supseteq T_S' = T_S \cup \{e\}$
Example:
\[ \text{Time: } \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}} \]

\[ \frac{1}{V} |\text{Adj}[v]| = \frac{1}{V} \deg(v) = 2 \cdot \frac{1}{V} \cdot E \] (Handshaking Lemma)

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Extract-Min</th>
<th>Decrease-Key</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array (nothing)</td>
<td>( O(V) )</td>
<td>( O(1) )</td>
<td>( O(V^2) )</td>
</tr>
<tr>
<td>binary heap</td>
<td>( O(\log V) )</td>
<td>( O(\log V) )</td>
<td>( O(E \log V) )</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>( O(\log V) )</td>
<td>( O(1) )</td>
<td>( O(E + V \log V) )</td>
</tr>
</tbody>
</table>
Kruskal’s algorithm: take globally lowest-weight edge & contract
- maintain connected components in MST-so-far T
  in Union-Find structure [Recitation 3]
- T=Ø ← will become MST
- for v∈V: Make-Set(v) ← initially, 1 vertex/comp.
- sort E by w
- for e=(u,v)∈E (in increasing weight order):
  - if Find-Set(u) ≠ Find-Set(v):
    - add e to T
    - Union(u,v)

Correctness: invariant: tree T so far ≤ MST T*
- assume by induction T≤T*
- when adding e between components C₁ & C₂: use
  greedy-choice property on cut (C₁, V \ C₂)

Time: T<sub>sort</sub>(E) + O(V).T<sub>make</sub> + O(E).O(find + Union)
  O(E lg E) tiny
  O(E) e.g. if weights are integers ∈ [O,E<sup>0(1)</sup>] ~ then can beat Prim

Best MST algorithm: [Karger, Klein, Tarjan 1993]
O(V+E) expected time (randomized)