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 $w : E \to \mathbb{R}$,
find spanning tree $T \subseteq E$ of minimum weight:
$$w(T) = \min_{T \subseteq E} w(T)$$

Example:

Naïve algorithm: check all spanning trees
- exponential time

Greedy properties: problems amenable to greedy algorithms usually satisfy:

1. optimal substructure: optimal solution to problem contains optimal solution(s) to subproblem(s)
   - also common for 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:

\[
G \quad \xrightarrow{\text{contract}} \quad G/e
\]

- if \( T' \) is an MST of \( G' = G/e \),
then \( T = T' \cup e \) is an MST of \( G \).
remap edges to decontract \( e \):

\[
G \quad \xrightarrow{\text{contract}} \quad G/e \quad \xrightarrow{\text{decontract}} \quad G
\]

**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 \( G' \)
  \( \Rightarrow w(T') \leq w(T^*/e) \)
  \( \Rightarrow w(T) = w(T') + w(e) \leq w(T^*/e) + w(e) = w(T^*). \blacksquare \)
Dynamic program attempt:
- guess an edge to put in MST
- contract to get new subproblem
- recurse
- decontract & add e

but # subproblems is exponential :(
Greedy-choice property for MST:

For any cut \((S, V \setminus S)\) in graph \(G = (V, E, w)\), any least-weight crossing edge \(e = \{u, v\}^3\) 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 \((S, V \setminus S)\).

\[ w(e) \leq w(e') \]
\[ w(T') \leq w(T) \]
\[ \Rightarrow T' \text{ is a MST too.} \]
Prim's algorithm: start with $|S|=1$ & grow from there
- maintain priority queue $Q$ on $V \setminus S$,
  where $v$.key = min \{w(u,v) | u \in S\}$
- initially $Q$ stores $V$ ($S=\emptyset$)
- $s$.key = $\emptyset$ (for arbitrary start vertex $s \in V$)
- for $v \in V \setminus \{s\}$: $v$.key = $\infty$
- until $Q$ empty:
  - $u = \text{Extract-Min}(Q)$ (add $u$ to $S$)
  - for $v \in \text{Adjacent}[u]$:
    - if $v \in Q (v \notin S) \& w(u,v) < v$.key:
      - $v$.key = $w(u,v)$ $\leftarrow \text{Decrease-Key}$
      - $v$.parent = $u$
  - return $\{ v \setminus v$.parent $\mid v \in V \setminus \{s\} \}$

Correctness:
- invariant: $v \notin S \Rightarrow v$.key = min \{w(u,v) | u \in S\}
- invariant: tree $T_S$ within $S \subseteq$ MST of $G$
  - if $c = \min \{a, b, c\} \leq \min \{a, b, c\} \& d$
    then by greedy-choice property, $e$ is in an MST
... even when $S$ is contracted to one vertex
- find MST containing $e$ in $G/T_S$
  & then decontract $T_S$ via optimal substructure
$\Rightarrow T_S \cup \exists e \subseteq$ an MST of $G$ (by ind.) $\square$
Example:
\[ \text{Time: } \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}} \]

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

<table>
<thead>
<tr>
<th>data structure</th>
<th>( T_{\text{Extract-Min}} )</th>
<th>( T_{\text{Decr.-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(lg V) )</td>
<td>( O(lg V) )</td>
<td>( O(E + V lg V) )</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>( O(lg V) )</td>
<td>( O(1) )</td>
<td>( O(E + V lg V) )</td>
</tr>
</tbody>
</table>

(CLRS ch. 19)
Kruskal’s algorithm: take globally lowest-weight edge & contract
- maintain connected components in MST-so-for T in Union-Find structure [Recitation 3]
- \( T = \emptyset \)
- for \( v \in V \): Make-Set(\( v \)) \( \leftarrow \) initially, 1 vertex/comp.
- sort \( E \) by \( w \)
- for \( e = (u, v) \in E \) (in increasing weight order):
  - if Find-Set(u) \( \neq \) Find-Set(v):
    - add \( e \) to MST
    - Union(u,v)

Correctness: imagine components as contracted + greedy-choice + optimal substructure

Time:
\[
\text{Time} = \underbrace{T_{\text{sort}}(V)}_{O(n \log n)} + O(V) \cdot T_{\text{make-set}} + O(E) \cdot (T_{\text{find}} + T_{\text{union}}) \]
\[
O(\alpha(V)) \text{ am. tiny}
\]
\[
O(n) \text{ e.g. if weights are integers } \in [O, n^0(1)] \quad \text{then can beat Prim}
\]

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