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

Recall: [Lecture 1]

Greedy algorithm: repeatedly make locally best choice/decision, ignoring effect on future
- saw greedy algorithm for scheduling problem
- today: greedy algorithm for graph problem

Recall: [6.006]

Adjacency-list representation of graph $G=(V,E)$:
- array `Adj` of $|V|$ linked lists, one per vertex
- `Adj[u] = neighbors of vertex u`
  $= \{ v \in V \mid (u,v) \in E \}$

$\text{Adj}[c] = \{ a, d \}$
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:

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:

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

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

but # subproblems is exponential 😞
**Greedy-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 \(u \in S \& v \not\in S\)
is in some MST of \(G\).

**Proof:** cut & paste argument
- Consider an MST \(T\) of \(G\)
- \(T\) has a path from \(u\) to \(v\)
- \(u \in S \& v \not\in S\), so the path has some edge \(e'=\{u', v'\}\) with \(u' \in S \& v' \not\in S\)
- 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. \(\Box\)
Prim's algorithm: 
start with $|S| = 1$ & grow from there
- maintain priority queue $Q$ on $V - S$,
  where $v.key = \min_{u \in S} \text{weight of edge } (u, v)$
- initially $Q$ stores $V$ (S=∅)
- $s.key = ∅$ (for arbitrary start vertex $s \in V$)
- for $v \in V - S$: $v.key = \infty$
- until $Q$ empty:
  - $u = \text{Extract-Min}(Q)$ (add $u$ to $S$
  - for $v \in \text{Adj}[u]$
    - if $v \in Q (v \notin S) \& w(u, v) < v.key$
      - $v.key = w(u, v) \leftarrow \text{Decrease-Key}$
      - $v.parent = u$
  - return $\forall v \in V \exists v.parent \exists 1 v \in V - S$

Correctness:
- invariant: $v \notin S \Rightarrow v.key = \min_{u \in S} \text{weight of edge } (u, v)$
- invariant: tree $T_S$ within $S \subseteq \text{MST of } G$
  - if $c = \min_{e \in T_S} c \leq \min_{e \in \{a, b, c\} \& d}$
    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
$\Rightarrow T_S U \exists e \exists \subseteq \text{an MST of } G$
Example:
\[
\text{Time: } \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}}
\]

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

<table>
<thead>
<tr>
<th>Priority Queue</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 \cdot lg V) )</td>
</tr>
<tr>
<td>Fibonacci Heap</td>
<td>( O(lg V) )</td>
<td>( O(1) )</td>
<td>( O(E + V \cdot lg V) )</td>
</tr>
</tbody>
</table>

(\text{CLRS ch. 19})
Union-find problem: [CLRS ch. 21]

- Maintain a collection of disjoint sets
- Subject to: Make-Set(x) - create \( \exists x \in \mathbb{Z} \)
- Find-Set(x) - return set \( \exists x \in \mathbb{Z} \)
- Union(x, y) - union sets containing \( x \) & \( y \), resp., destroying old sets

- E.g. 2-3 trees (& most balanced search trees)
  - Find-Set = walk up to root \( \mathcal{O}(\log n) \)
  - Union = concatenate (no order) \( \mathcal{O}(\log n) \)

- Best (possible) DS: \( \mathcal{O}(\alpha(n)) \) amortized/op.

- Ackermann function:
  - \( A_0(n) = n + 1 \)
  - \( A_m(0) = A_{m-1}(1) \)
  - \( A_m(n) = A_{m-1}(A_m(n-1)) \) \( m > 0 \)
  - \( A_m(n) = A_{m-1}(1) \) \( m, n > 0 \)

\[ A_1(n) = 2n \]
\[ A_2(n) = 2^n \]
\[ A_3(n) = 2 \uparrow \uparrow n = 2^{2^{\cdots^n}} \]
\[ A_4(n) = 2 \uparrow \uparrow \cdots \uparrow 2 \]

- \( \alpha = A^{-1} : \alpha(n) = \text{smallest } x \text{ s.t. } A(x) \geq n \)
Kruskal's algorithm: take globally lowest-weight edge & contract
- maintain connected components in MST-so-far \( T \)
- \( T = \emptyset \)
- 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 \( \text{Find-Set}(u) \neq \text{Find-Set}(v) \):
    - add \( e \) to MST
    - Union\((u,v)\)

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

Time: \( T_{\text{sort}}(V) + \Theta(V) \cdot T_{\text{make-set}} + \Theta(E) \cdot (T_{\text{find}} + T_{\text{union}}) \)
\[ O(n \log n) \]
\[ O(n) \text{ e.g. if weights are integers } \in [O_n, n^{O(1)}] \text{ then can beat Prim} \]

Best MST algorithm: \( O(V+E) \) expected time (randomized)