definition: tree

A tree is a connected graph: for any pair $u, v \in V$, we have that there exists a path from $u$ to $v$ in $G$, connected graph that has no cycles.
what we want:

1. set of edges AC E that connects all nodes in the graph
2. minimize the cost of this set A

(each edge has a cost)
minimum spanning tree

looking for a set of edges that $T \subseteq E$
(a) connects all vertices
(b) has the least cost $\min \sum_{(u,v) \in T} w(u,v)$
looking for a set of edges that $T \subseteq E$

(a) connects all vertices
(b) has the least cost $\min \sum_{(u,v) \in T} w(u,v)$

- how many edges does solution have? $V - 1$
- does solution have a cycle? No cycle!!
Greedy strategy

start with an empty set of edges \( A \)
repeat for \( v-1 \) times:
    add lightest edge that does not create a cycle
example

Graph with nodes labeled a, b, c, d, e, f, g, and i, and edges labeled with numbers.
Kruskal
Kruskal
Kruskal
Kruskal
Kruskal
Kruskal
Kruskal
why does this work?

1. \( T \leftarrow \emptyset \)
2. repeat \( V - 1 \) times:
3. \( \quad \text{add to } T \text{ the lightest edge } e \in E \text{ that does not create a cycle} \)
A cut is a partition of the set $V$ into $2$ sets $(S, V-S)$. 
example of a cut

\[ v \sim S \]
definition: crossing a cut

An edge \( e = (u, v) \) crosses a cut \((S, V-S)\)

if \( u \in S \) and \( v \in V-S \).
definition: crossing a cut

an edge $e = (u, v)$ crosses a graph cut $(S, V - S)$ if $u \in S$ and $v \in V - S$.
example of a crossing
definition: respect

A set $A$ respects the cut $(S, V-S)$ if no edge $e \in A$ crosses $(S, V-S)$. 

A set $A$ respects the cut $(S, V-S)$ if no edge $e \in A$ crosses $(S, V-S)$. 
Cut theorem

Let $T$ be an MST for $(S, V)$ and let $A \subseteq T$.

Let $(S, V - S)$ be some cut that $A$ respects and let $e$ be the lightest edge that crosses $(S, V - S)$.

$\Rightarrow A \cup \{e\}$ is a subset of some MST of $G$. 
Cut theorem

Suppose the set of edges $\mathcal{A}$ is part of an m.s.t. Let $(S, V - S)$ be any cut that $\mathcal{A}$ respects. Let edge $e$ be the min-weight edge across $(S, V - S)$. Then: $\mathcal{A} \cup \{e\}$ is part of an m.s.t.
example of theorem
\[ A = \{ \{i, g\}, \{c, f\}\} \]
Theorem 2 Suppose the set of edges $A$ is part of a minimum spanning tree of $G = (V, E)$. Let $(S, V - S)$ be any cut that respects $A$ and let $e$ be the edge with the minimum weight that crosses $(S, V - S)$. Then the set $A \cup \{e\}$ is part of a minimum spanning tree.

Proof: By hypothesis $A \subseteq T$ where $T$ is an MST of $G$.

If $A \cup \{e\}$ is already $\subseteq T$, then the theorem follows.

If not, then we need to construct another $T'$ tree such that $A \cup \{e\} \subseteq T'$ and $T'$ is also an MST.

\text{How?}
Let $e = (u, v)$ be the lightest edge that crosses $(S, V-S)$. We is not part of $T$, but since $T$ is an MST, it connects all nodes in $S$. So follow the path from $u$ to $v$ and let $e'$ be the first edge to cross $(S, V-S)$. Why does $e'$ exist? By $e$ crossing $(S, V-S)$ so we use and $v \in V-S$.

Consider the tree $T' = T - \varepsilon e_3 + \varepsilon e'_3$. It has $(v-1)$ edges.

1. $w(e) \leq w(e') \Rightarrow w(T') \leq w(T)$. But $T$ was MST, so $T'$ is an MST.
Minimum Spanning Tree Algorithm

Kruskal-pseudocode

Kruskal-pseudocode($G$)

1. $A \leftarrow \emptyset$
2. repeat $V - 1$ times:
3. \hspace{1em} add to $A$ the lightest edge $e \in E$ that does not create a cycle

Theorem 2

Suppose the set of edges $A$ is part of a minimum spanning tree of $G = \langle V, E \rangle$. Let $w_S, V_S$ be any cut that respects $A$ and let $e$ be the edge with the minimum weight that crosses $w_S, V_S$. Then the set $A \cup \{e\}$ is part of a minimum spanning tree.

Proof:

By induction: $A$ is part of some MST $T$ of $G$, at line 1.

Suppose $A$ is part of an MST after $k$ iterations of the main loop.

Show in the next iteration that $A$ remains part of an MST.

$\Rightarrow e$ was the lightest edge that didn't create a cycle.

$e = (u, v)$. 
correctness

Kruskal-pseudocode(G)

1 \( A \leftarrow \emptyset \)
2 repeat \( V - 1 \) times:
3 add to \( A \) the lightest edge \( e \in E \) that does not create a cycle

Proof: by induction. In step 1, \( A \) is part of some MST. Suppose that after \( k \) steps, \( A \) is part of some MST (line 2). In line 3, we add an edge \( e=(u,v) \).
3 cases for edge e.
Case 1: e=(u,v) and both u,v are in A.
3 cases for edge $e$.
Case 2: $e=(u,v)$ and only $u$ is in $A$. 
3 cases for edge $e$.
Case 3: $e=(u,v)$ and neither $u$ nor $v$ are in $A$. 
Kruskal-pseudocode($G$)

1  $A \leftarrow \emptyset$
2  repeat $V - 1$ times:
3    add to $A$ the lightest edge $e \in E$ that does not create a cycle
**General-MST-Strategy**\((G = (V, E))\)

1. \(A \leftarrow \emptyset\)
2. **repeat** \(V - 1\) times:
3. Pick a cut \((S, V - S)\) that respects \(A\),
4. Let \(e\) be min-weight edge over cut \((S, V - S)\)
5. \(A \leftarrow A \cup \{e\}\)
Prim's algorithm

**General-MST-Strategy**$(G = (V,E))$

1. $A \leftarrow \emptyset$
2. repeat $V-1$ times:
   3. Pick a cut $(S, V-S)$ that respects $A$
   4. Let $e$ be min-weight edge over cut $(S, V-S)$
   5. $A \leftarrow A \cup \{e\}$

A is a subtree.
edge $e$ is lightest edge that grows the subtree
prim

diagram of a graph with nodes labeled a, b, c, d, e, f, g, h, i and edges labeled with numbers.
prim
prim
prim
prim
idea: At each step, we need to identify the "lightest edge" which augments our tree -

- use priority queue
implementation
new data structure

Priority queue

- make \((q, \cdots, q)\) & create a queue with \(n\) elements

- \texttt{extractmin} - produces smallest element in queue

- \texttt{decreasekey} - reduces the key value for some item.
binary heap

full tree, key value $\leq$ to key of children
binary heap

full tree, key value $\leq$ to key of children
binary heap

full tree, key value $\leq$ to key of children

$\text{insert}(8)$
binary heap

full tree, key value \leq to key of children
binary heap
full tree, key value \leq\text{ to key of children}

how to extract min?
binary heap
full tree, key value $\leq$ to key of children

how to extract min?
binary heap
binary heap
full tree, key value <= to key of children

how to extractmin? → $\Theta(\log n)$
how to decreasekey?
binary heap

full tree, key value <= to key of children

how to extract min?
how to decrease key?
implementation

use a priority queue to keep track of light edges

insert: \( O(Cn) \)
makequeue: \( O(Cn) \)
extractmin: \( O(\log n) \)
decreasekey: \( O(\log n) \)
Prim’s algorithm
PRIM($G = (V, E)$)
1 $Q \leftarrow \emptyset$  \hspace{1em} ▷ $Q$ is a Priority Queue
2 Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3 Pick a starting node $r$ and set $k_r \leftarrow 0$
4 Insert all nodes into $Q$ with key $k_v$.
5 \hspace{1em} \textbf{while} $Q \neq \emptyset$
6 \hspace{2em} do $u \leftarrow \text{EXTRACT-MIN}(Q)$
7 \hspace{3em} \textbf{for} each $v \in \text{Adj}(u)$
8 \hspace{4em} do if $v \in Q$ and $w(u, v) < k_v$
9 \hspace{5em} then $\pi_v \leftarrow u$
10 \hspace{4em} $\text{DECREASE-KEY}(Q, v, w(u, v))$  \hspace{1em} ▷ Sets $k_v \leftarrow w(u, v)$
PRIM($G = (V, E)$)
1. $Q \leftarrow \emptyset$  \quad \triangleright Q is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. \textbf{while} $Q \neq \emptyset$
6. \quad \textbf{do} $u \leftarrow \text{EXTRACT-MIN}(Q)$
7. \quad \textbf{for each} $v \in \text{Adj}(u)$
8. \quad \quad \textbf{do if} $v \in Q$ and $w(u, v) < k_v$
9. \quad \quad \quad \textbf{then} $\pi_v \leftarrow u$
10. \quad \quad \quad \textbf{DECREASE-KEY}(Q, v, w(u, v))  \quad \triangleright \text{Sets } k_v \leftarrow w(u, v)$
PRIM($G = (V, E)$)

1. $Q \leftarrow \emptyset$  \quad $\triangleright$  $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. \hspace{1em} while $Q \neq \emptyset$
6. \hspace{2em} do $u \leftarrow \text{EXTRACT-MIN}(Q)$
7. \hspace{3em} for each $v \in \text{Adj}(u)$
8. \hspace{4em} do if $v \in Q$ and $w(u, v) < k_v$
9. \hspace{5em} then $\pi_v \leftarrow u$
10. \hspace{4em} \hspace{1em} $\text{DECREASE-KEY}(Q, v, w(u, v))$  \quad $\triangleright$  Sets $k_v \leftarrow w(u, v)$
PRIM$(G = (V, E))$

1. $Q \leftarrow \emptyset$  \hspace{1em} $\triangleright$  \hspace{1em} $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. While $Q \neq \emptyset$
6. \hspace{1em} Do $u \leftarrow \text{EXTRACT-MIN}(Q)$
7. \hspace{2em} For each $v \in \text{Adj}(u)$
8. \hspace{3em} Do if $v \in Q$ and $w(u, v) < k_v$
9. \hspace{4em} Then $\pi_v \leftarrow u$
10. \hspace{3em} DECREASE-KEY$(Q, v, w(u, v))$  \hspace{1em} $\triangleright$  \hspace{1em} Sets $k_v \leftarrow w(u, v)$
PRIM($G = (V, E)$)
1. $Q \leftarrow \emptyset$  \(\triangleright\) $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. while $Q \neq \emptyset$
6.  \hspace{0.5cm} do $u \leftarrow$ EXTRACT-MIN($Q$)
7.  \hspace{1cm} for each $v \in Adj(u)$
8.  \hspace{1.5cm} do if $v \in Q$ and $w(u, v) < k_v$
9.  \hspace{2cm} then $\pi_v \leftarrow u$
10. \hspace{1.5cm} DECREASE-KEY($Q, v, w(u, v)$) \(\triangleright\) Sets $k_v \leftarrow w(u, v)$
PRIM($G = (V, E)$)

1. $Q \leftarrow \emptyset$  \quad \triangleright \quad Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty, \pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. while $Q \neq \emptyset$
   
   do $u \leftarrow \text{EXTRACT-MIN}(Q)$
   
   for each $v \in \text{Adj}(u)$
   
   do if $v \in Q$ and $w(u, v) < k_v$
   
   then $\pi_v \leftarrow u$

10. $\text{DECREASE-KEY}(Q, v, w(u, v))$  \quad \triangleright \quad Sets \( k_v \leftarrow w(u, v) \)
PRIM($G = (V, E)$)
1 $Q \leftarrow \emptyset$  \hspace{1em} ▷ $Q$ is a Priority Queue
2 Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3 Pick a starting node $r$ and set $k_r \leftarrow 0$
4 Insert all nodes into $Q$ with key $k_v$.
5 while $Q \neq \emptyset$
  6 do $u \leftarrow \text{EXTRACT-MIN}(Q)$
  7 for each $v \in Adj(u)$
  8 do if $v \in Q$ and $w(u, v) < k_v$
  9 then $\pi_v \leftarrow u$
 10 $\text{DECREASE-KEY}(Q, v, w(u, v))$  \hspace{1em} ▷ Sets $k_v \leftarrow w(u, v)$
running time

\[ \text{PRIM}(G = (V, E)) \]

1. \( Q \leftarrow \emptyset \quad \triangleright \text{Q is a Priority Queue} \)
2. Initialize each \( v \in V \) with key \( k_v \leftarrow \infty \), \( \pi_v \leftarrow \text{NIL} \)
3. Pick a starting node \( r \) and set \( k_r \leftarrow 0 \)
4. Insert all nodes into \( Q \) with key \( k_v \).
5. \( \text{while } Q \neq \emptyset \) \( \rightarrow \Theta(V \log V) \text{ time} \)
6. \( \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \)
7. \( \text{for each } v \in \text{Adj}(u) \)
8. \( \text{do if } v \in Q \text{ and } w(u, v) < k_v \)
9. \( \text{then } \pi_v \leftarrow u \)
10. \( \text{DECREASE-KEY}(Q, v, w(u, v)) \quad \triangleright \text{Sets } k_v \leftarrow w(u, v) \)

\[ \Theta(E \log V) \text{ time} \]

\[ O(E \log V + V \log V) = O(E \log V) \]
implementation

PRIM\((G = (V, E))\)
1 \(Q \leftarrow \emptyset\) \(\triangleright\) \(Q\) is a Priority Queue
2 Initialize each \(v \in V\) with key \(k_v \leftarrow \infty, \pi_v \leftarrow\) NIL
3 Pick a starting node \(r\) and set \(k_r \leftarrow 0\)
4 Insert all nodes into \(Q\) with key \(k_v\).
5 while \(Q \neq \emptyset\)
6 do \(u \leftarrow\) EXTRACT-MIN\((Q)\)
7 for each \(v \in Adj(u)\)
8 do if \(v \in Q\) and \(w(u, v) < k_v\)
9 then \(\pi_v \leftarrow u\)
10 \(\text{DECREASE-KEY}(Q, v, w(u, v))\) \(\triangleright\) Sets \(k_v \leftarrow w(u, v)\)

\(O(V \log V + E \log V) = O(E \log V)\)
use a priority queue to keep track of light edges

<table>
<thead>
<tr>
<th>Operation</th>
<th>Priority Queue</th>
<th>Fibonacci Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>$O(\log n)$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>makequeue</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>extractmin</td>
<td>$O(\log n)$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>decreasekey</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
A $k$th order binomial tree, which I’ll abbreviate $B_k$, is defined recursively. $B_0$ is a single node. For all $k > 0$, $B_k$ consists of two copies of $B_{k-1}$ that have been linked together, meaning that the root of one $B_{k-1}$ has become a new child of the other root.

Binomial trees have several useful properties, which are easy to prove by induction when $k = 0$. These properties include:

- The root of $B_k$ has degree $k$.
- The children of the root of $B_k$ are the roots of $B_0, B_1, ..., B_k$.
- $B_k$ has height $k$.
- $B_k$ has $2^k$ nodes.
- $B_k$ can be obtained from $B_{k-1}$ by adding a new child to every node.
- $B_k$ has $\binom{k}{d} \cdot d^2$ nodes at depth $d$, for all $0 \leq d < k$.
- $B_k$ has $2^k h_1$ nodes with height $h$, for all $0 < h < k$, and one node with the root with height $k$.

Although we normally don’t care in this class about the low-level details of data structures, we need to be specific about how Fibonacci heaps are actually implemented, so that we can be sure that certain operations can be performed quickly. Every node in a Fibonacci heap points to four other nodes: its parent, its ‘next’ sibling, its ‘previous’ sibling, and one of its children. The sibling pointers are used to join the roots together into a circular doubly-linked root list. In each binomial tree, the children of each node are also joined into a circular doubly-linked list using the sibling pointers.

With this representation, we can add or remove nodes from the root list, merge two root lists together, link one binomial tree to another, or merge a node’s list of children with the root list, in constant time, and we can visit every node in the root list in constant time per node. Having established that these primitive operations can be performed quickly, we never again need to think about the low-level representation details.
faster implementation

PRIM($G = (V, E)$)

1. $Q \leftarrow \emptyset$  \hspace{0.5cm} ➔ $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow $ NIL
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. while $Q \neq \emptyset$
6. \hspace{0.5cm} do $u \leftarrow$ EXTRACT-MIN($Q$)
7. \hspace{1cm} for each $v \in Adj(u)$
8. \hspace{1.5cm} do if $v \in Q$ and $w(u, v) < k_v$
9. \hspace{2cm} then $\pi_v \leftarrow u$
10. \hspace{1.5cm} DECREASE-KEY($Q, v, w(u, v)$)  \hspace{0.5cm} ➔ Sets $k_v \leftarrow w(u, v)$

$O(E + V \log V)$
Research in mst

FREDMAN-TARJAN 84:
GABOW-GALIL-SPENCER-TARJAN 86:
CHAZELLE 97
CHAZELLE 00
PETTIE-RAMACHANDRAN 02:
KARGER-KLEIN-TARJAN 95:
    (randomized)

Euclidean mst:

\[ E + V \log V \]
\[ E \log(\log^* V) \]
\[ E \alpha(V) \log \alpha(V) \]
\[ E \alpha(V) \]
\[ (optimal) \]

\[ V \log V \]
Ackerman function

\[
A(m, n) = \begin{cases} 
  n + 1 & m = 0 \\
  A(m - 1, 1) & m > 0, n = 0 \\
  A(m - 1, A(m, n - 1)) & m, n > 0
\end{cases}
\]

\[A(4, 2) = \]
inverse ackerman

\[ \alpha(n) = \]
application of mst
application of mst
application of mst
simple graph questions

what is the length of the path from a to e?
shortest path property

definition:

$$\delta(s, v)$$
shortest paths
algorithm
Dijkstra\((G = (V, E), s)\)

1. for all \(v \in V\) do 
   2. \(d_u \leftarrow \infty\)
   3. \(\pi_u \leftarrow \text{NIL}\)
4. \(d_s \leftarrow 0\)
5. \(Q \leftarrow \text{MAKEQUEUE}(V)\)  \(\triangleright\) use \(d_u\) as key
6. while \(Q \neq \emptyset\) do
7.     \(u \leftarrow \text{EXTRACTMIN}(Q)\)
8.     for each \(v \in \text{Adj}(u)\) do
9.         if \(d_v > d_u + w(u, v)\) then
10.            \(d_v \leftarrow d_u + w(u, v)\)
11.            \(\pi_v \leftarrow u\)
12.            \(\text{DECREASEKEY}(Q, v)\)

Theorem 4
Given any weighted, directed graph \(G = (V, E)\) with non-negative weights and source \(s\), \(\text{dijkstra}(G, s)\) terminates with \(d_u = (s, v)\) for all \(v \in V\).
**Dijkstra**\((G = (V, E), s)\)

1. **for** all \(v \in V\)
2. \(d_u \leftarrow \infty\)
3. \(\pi_u \leftarrow \text{NIL}\)
4. \(d_s \leftarrow 0\)
5. \(Q \leftarrow \text{MAKEQUEUE}(V)\) \(\triangleright\) use \(d_u\) as key
6. **while** \(Q \neq \emptyset\)
7. \(u \leftarrow \text{EXTRACTMIN}(Q)\)
8. **for** each \(v \in \text{Adj}(u)\)
9. **do if** \(d_v > d_u + w(u, v)\)
10. then \(d_v \leftarrow d_u + w(u, v)\)
11. \(\pi_v \leftarrow u\)
12. \(\text{DECREASEKEY}(Q, v)\)

**PRIM\((G = (V, E))\)**

1. \(Q \leftarrow \emptyset\) \(\triangleright\) \(Q\) is a Priority Queue
2. **Initialize** each \(v \in V\) with key \(k_v \leftarrow \infty\), \(\pi_v \leftarrow \text{NIL}\)
3. **Pick** a starting node \(r\) and set \(k_r \leftarrow 0\)
4. **Insert** all nodes into \(Q\) with key \(k_v\).
5. **while** \(Q \neq \emptyset\)
6. \(u \leftarrow \text{EXTRACT-MIN}(Q)\)
7. **for** each \(v \in \text{Adj}(u)\)
8. **do if** \(v \in Q\) and \(w(u, v) < k_v\)
9. then \(\pi_v \leftarrow u\)
10. \(\text{DECREASE-KEY}(Q, w(u, v))\) \(\triangleright\) Sets \(k_v \leftarrow w(:\)