Outline

1. Problem Definition
2. The General Recipe
3. Kruskal’s Algorithm
4. Prim’s Algorithm
5. Borůvka’s Algorithm
Plan

1. Problem Definition
2. The General Recipe
3. Kruskal’s Algorithm
4. Prim’s Algorithm
5. Borůvka’s Algorithm
Minimum spanning tree

**Input:** An *undirected*, connected and *weighted* graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with weights $w : \mathcal{E} \to \mathbb{R}$.

**Output:** A *spanning* tree $\mathcal{T}$ with *minimum* weight.

**Tree:** connected subgraph with no cycles.

**Spanning:** includes every node in $\mathcal{G}$.

**Claim:** A spanning tree has exactly $n - 1$ edges.

**Weight:** The weight of a tree is the sum of weights of its edges.

**Quiz:** how to find a *maximum* spanning tree?
Applications

- circuit design
- communication networks
- A* search
Example (CLRS)
How many MSTs?

Incidence matrix

\[ I \in \{-1, 0, 1\}^{n \times m}, \quad I_{ue} = 1, \quad I_{ve} = -1 \iff e = (u, v) \]

Laplacian matrix

\[ L = II^\top = D - A, \quad D = \text{diag}(d_v) \]

Kirchhoff theorem: Let \( \lambda_1, \ldots, \lambda_{n-1}, \lambda_n = 0 \) be eigenvalues of \( L \). Then

\[ \#\text{MST} = \prod_{i=1}^{n-1} \lambda_i. \]
Plan

1. Problem Definition

2. The General Recipe

3. Kruskal’s Algorithm

4. Prim’s Algorithm

5. Borůvka’s Algorithm
Algorithm 1: Generic-MST($G$, $w$)

1. $A \leftarrow \emptyset$
2. while $A$ does not form a spanning tree do
   3. find a safe edge $(u, v)$ for $A$
   4. $A \leftarrow A \cup \{(u, v)\}$
5. return $(A)$

**Safe edge:** An edge $(u, v)$ is safe for $A$ if $\exists$ a spanning tree $T$ that contains both $A$ and $\{(u, v)\}$.

**Claim:** There always exists a safe edge.

**Claim:** Easy to check if $A$ forms a spanning tree.
Finding an safe edge

**Theorem:** Let \((S, V - S)\) be any cut of \(G\) that respects \(A\), and let \((u, v)\) be a light edge crossing \((S, V - S)\). Then, \((u, v)\) is safe for \(A\).

**respect:** no edge in \(A\) crosses the cut.

**light:** minimum weight crossing edge.

**proof:** cut and paste.
Plan

1. Problem Definition
2. The General Recipe
3. Kruskal’s Algorithm
4. Prim’s Algorithm
5. Borůvka’s Algorithm
Corollary: Let $C = (V_C, E_C)$ be a tree in the forest $G_A = (V, A)$. If $(u, v)$ is a light edge connecting $C$ to some other tree in $G_A$, then $(u, v)$ is safe for $A$.

Algorithm 2: Kruskal($G, w$)

1. $A \leftarrow \emptyset$
2. sort $E$ increasingly according to $w$
3. for each edge $(u, v) \in E$ do
   4. if $u$ and $v$ in different trees of $(V, A)$ then
   5. $A \leftarrow A \cup \{(u, v)\}$
6. return $(A)$

Property: $A$ is a forest.

Complexity: with suitable data structure, $O(m \log m) = O(m \log n)$. 
Think oppositely

Kruskal keeps adding light edges to "connect" a forest.

Conversely, can keep deleting heavy edges and avoiding disconnect the graph.

**Claim:** the heaviest edge in a cycle may not be in a MST.
Example (CLRS)
Example cont’

Graph III: MST

(i) Example graph 1

(j) Example graph 2

(k) Example graph 3

(l) Example graph 4

(m) Example graph 5

(n) Example graph 6
Plan

1. Problem Definition
2. The General Recipe
3. Kruskal’s Algorithm
4. Prim’s Algorithm
5. Borůvka’s Algorithm
Prim’s idea

Maintain a tree $A$ that is included in some MST.

Each node not in $A$ maintain a light edge to some node in $A$.

Each time add a safe edge to a node not in $A$.

Update the light edges.

With suitable data structure, can run in $O(m + n \log n)$.
Algorithm 3: Prim\((G, w)\)

1. \(A \leftarrow \emptyset\)
2. \(V_A \leftarrow \{u\}\) \hspace{1cm} // arbitrary \(u\)
3. \textbf{forall} \(v \in V \setminus \{u\} \text{ do} \)
   4. \hspace{1cm} \(\omega[v] \leftarrow w(u, v)\)
   5. \hspace{1cm} \(\pi[v] \leftarrow u\)
4. \textbf{while} \(|A| < n - 1\) \textbf{do} \)
   5. \hspace{1cm} choose \(v \in V \setminus V_A\) with \(\omega[v]\) minimized
   6. \hspace{1cm} \(V_A \leftarrow V_A \cup \{v\}\)
   7. \hspace{1cm} \(u \leftarrow \pi[v]\)
   8. \hspace{1cm} \(A \leftarrow A \cup \{uv\}\)
   9. \hspace{1cm} \textbf{forall} \(z \in V \setminus V_A\) \textbf{do} \)
     10. \hspace{1cm} \hspace{1cm} \textbf{if} \(w(v, z) < \omega[z]\) \textbf{then} \)
     11. \hspace{1cm} \hspace{1cm} \hspace{1cm} \(\omega[z] \leftarrow w(v, z)\)
     12. \hspace{1cm} \hspace{1cm} \hspace{1cm} \(\pi(z) \leftarrow v\)
13. \textbf{return} \((A)\)
Example (CLRS)
Plan

1. Problem Definition
2. The General Recipe
3. Kruskal’s Algorithm
4. Prim’s Algorithm
5. Borůvka’s Algorithm
Algorithm 4: Prim($G, w$)

1. $T \leftarrow \{\{u\} : u \in \mathcal{V}\}$
2. while $T$ is disconnected do
3.   forall component $C$ in $T$ do
4.     find light edge $e$ in $(C, \mathcal{V} - C)$
5.     add $e$ to $T$
6.   combine components in $T$
7. return $(A)$

With suitable data structure, can run in $O(m \log n)$. 
Example

courtesy to wikipedia