- **Cycle property:** for any cycle in graph $G$, removing any one of its longest edges does not change the length of MST for $G$.

- **Cut property:** Given a subset of edges $F$, suppose $F$ is a subset of an MST $T$. Pick any cut $(S, \bar{S})$ that does not intersect with $F$, let $e$ be (any) edge with minimum cost in $(S, \bar{S})$, then $F \cup \{e\}$ is a subset of an MST $T'$ ($T'$ may or may not be the same as $T$).

Proof of cut property:

Let $e$ be min cost edge of cut $(S, \bar{S})$.

**Case 1:** $e$ is an edge in $T$, this is trivial because $F \cup \{e\} \subseteq T$, can choose $T' = T$.

**Case 2:** $e$ is not an edge in $T$. 

1. **Black:** edges in $T$
2. **Green:** MST $T$ containing $F$
3. **Purple:** min cost edge $e$
4. **Yellow:** cycle formed by adding edge $e$
5. **Green:** an edge in $C$, $(S, \bar{S})$, $T'$
6. **Blue:** new MST $T'$
add $e$ to $T$ form a cycle, call it $C$.

every cycle that interese $S$ with $(S, S')$ must intersect an even number of times.

there must be another edge $e' \in C$

$e' \in T$, $e'$ also crosses the cut $(S, S')$

we will swap $e$ and $e'$

define $T' = (T \setminus \{e\}) \cup \{e'\}$

$\text{cost}(T') = \text{cost}(T) - w(e') + w(e)$

$\leq \text{cost}(T)$ by assumption $w(e) \leq w(e')$

if $T$ is an MST, then $T'$ is also an MST.

since $E \cup \{e'\} \subseteq T'$, this concludes the proof. \( \square \)

- Prim's algorithm

\[ \text{Graph} \]

\[ \text{Diagram} \]
- In implementation, want to find min cost edge in the cut efficiently.
  
  maintain an array \( \text{dis}(u) \)

  \( \text{dis}(u) \): minimum cost of an edge \((u,v)\) where \(v\) is already connected to \(S\).

  running time: \( O(m+n \log n) \) (use Fibonacci heap)

- Kruskal

  \[
  \begin{align*}
  \text{union } (1,2) \\
  \{1,2\} \quad \{3\} \quad \{4\} \quad \{5\} \\
  \text{find}(3) = \text{find}(4) \\
  \text{union } (3,4) \\
  \{1,2\} \quad \{3,4\} \quad \{5\} \\
  \text{find}(1) = \text{find}(3) \\
  \text{union } (1,3) \\
  \{1,2,3,4\} \quad \{5\} \\
  \text{find}(2) = \text{find}(4) \\
  \text{adding this edge creates a cycle, Kruskal will not add this edge.} \\
  \text{find}(3) = \text{find}(5) \\
  \text{union } (3,5)
  \end{align*}
  \]
- Implementing Kruskal
  - union-find data structure.
    - maintain disjoint sets of \( \{1, 2, \ldots, N \} \)
    - initially, every element is in a separate set
      \( \{1\}, \{2\}, \{3\}, \ldots, \{N\} \)
      (corresponds to the case that none of the vertices are connected)
  - two operations
    1. union: merges two sets
    2. find: for every element \( u \), \( \text{find}(u) \) identifies the
       set that \( u \) belongs to.
       - if \( u, v \) are in the same set \( \text{find}(u) = \text{find}(v) \)
       - if \( u, v \) are in different sets \( \text{find}(u) \neq \text{find}(v) \)

- one implementation of union-find
  - idea: use a tree structure
    - every tree \( \leftrightarrow \) set
    - every vertex maintains a pointer to its parent

- find: finds the root of tree

\[ \text{find}(1) = 1, \quad \text{find}(2) = 1 \]
\[ \text{find}(4) = 1 \quad \text{find}(2) = 1 \]

- **union**: first find the two roots, point one of them to the other

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \]

union 0 2

\[ 1 \]
\[ \{1, 2\} \]

union 3 4

\[ 2 \]
\[ 3 \]
\[ \{3, 4\} \]

union 0 3

\[ 0 \]
\[ \{1, 2, 3, 4\} \]

union 3 5

\[ 3 \]
\[ 4 \]
\[ \{3, 4, 5\} \]

Claim: always link shallower tree to deeper tree, depth of the tree is at most $O(\log n)$.

Runtime: find: proportional to depth $O(\log n)$

union: two find operations + linking $O(1)$

$O(\log n)$

This implementation: can check whether adding $(u, v)$ creates a cycle in $O(\log n)$ time.
\[ \Rightarrow \text{Kruskal runs in } O(m \log n). \]