Proof Techniques

1. Invariants for "tree growing" algorithms.

- Prim's: edges of $S$ are a subset of a MST.
- Dijkstra's: edges in $S$ for shortest paths from $s$ to each vertex in $S$

$S$ = vertices already in the tree

updates: priority is updated for neighbors of $v$ when $v$ is added, based on
- Prim's: $w(v,t)$
- Dijkstra's: $d(v) + w(v,t)$

where $t$ = the neighbor

2. grey $\rightarrow$ grey

$\overrightarrow{abc}$

(a) cross
(b) forward/tree
(c) back

DFS Properties

can't happen because a vertex is grey only between its parentheses
Recursion/induction

\[ K_1 \leq K_2 \leq \ldots \leq K_{m+1} \]

Suppose \( K_1 \leq K_2 \leq K_3 \leq K_4 \) and smallest is in \( K_2 \).

Solve off-line min for \( K_1 \leq K_2 \cup K_3 - \{\text{smallest} \} \leq K_4 \) and then "insert" the extraction of smallest into the solution.

\[
\begin{array}{c|c|c|c}
\times & y & z \\
\hline
\times & \text{(smallest)} & x \\
\end{array}
\]
4. Static properties

Graph G is semi-connected iff

The dag of SCC's has a path that includes all the vertices (the vertices of the SCC dag, that is, each one of which represents an SCC of the original graph G), semi-connected \( \Rightarrow \) there exists a path that includes all vertices in the SCC dag.

Suppose there is no such path and let \( P \) be the path of the SCC dag having the most vertices. Let \( X \) be a vertex not on \( P \) and let \( X \epsilon \) SCC represented by \( X \). Let \( Y \) be the last vertex on \( P \) that contains only predecessors of \( X \) and \( Z \) be the first vertex the contains only successors. Then \( YX \) and \( XZ \) must be edges of the SCC dag, contradicting our choice of \( P \) as longest. Note: we know that \( Y \) and \( Z \) exist because \( G \) is semi-connected (every vertex must either be a predecessor or successor of \( X \)) and because the SCC graph is acyclic (if any vertex contained both predecessors and successors, there would be a cycle involving \( X \) and that vertex).

Existence of path \( \Rightarrow \) G semi-connected

This part is easy to do directly: Choose any pair of vertices \( u \) and \( v \) and let \( C_u \) and \( C_v \) be their SCC's. If \( C_u = C_v \), we are done. Otherwise, if \( C_u \) precedes \( C_v \) on the path then there's obviously a path from \( u \) to \( v \) in \( G \) and vice versa if \( C_v \) precedes \( C_u \). \( \Box \)