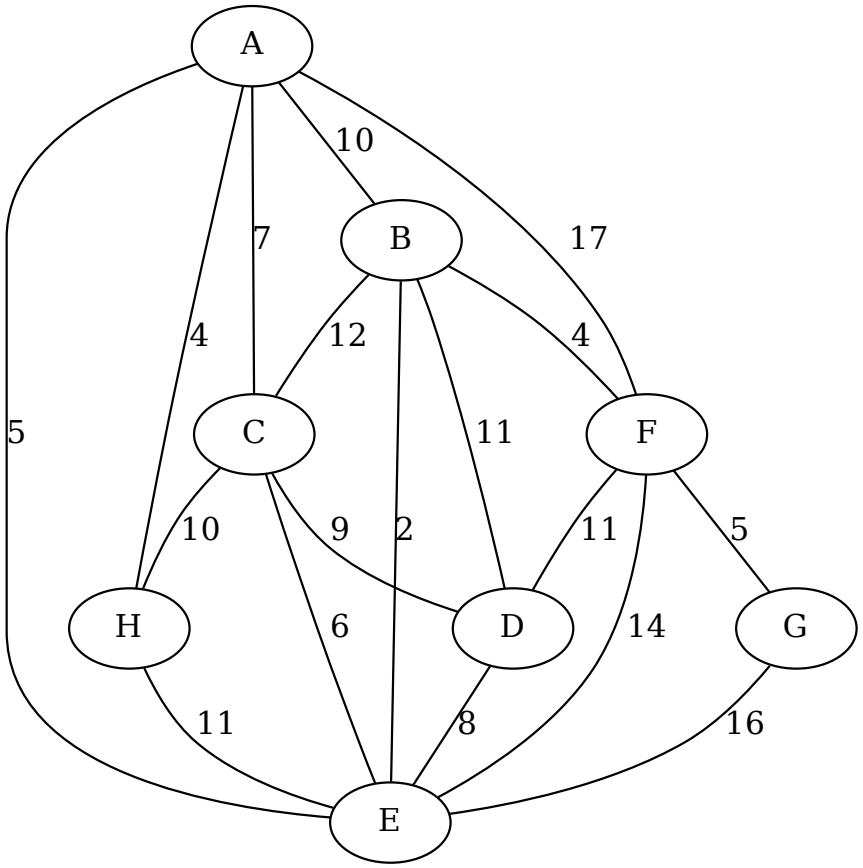
TD 8: Minimum Spanning Trees II

1 Execute Kruskal's algorithm

Execute Kruskal's algorithm and show the resulting spanning tree. If there are ties, break them alphabetically. Also keep track of the leader of each component and its rank, so that Union-Find can be implemented efficiently. When merging two components whose leaders have the same rank, make the vertex that comes first alphabetically the new leader.



2 Light Edges

Recall that an edge e is light for a set of vertices S if for all edges f with exactly one endpoint in S we have $w(f) \geq w(e)$. In other words, e is light if it is the cheapest edge connecting S to $V \setminus S$.

Let $G = (V, E)$ be a given connected graph. Prove or disprove the following:

1. Let T^* be a spanning tree of minimum weight for G. For all edges $e \in T^*$ there exists $S \subseteq V$ such that e is light for S.

2. For all $S \subseteq V$, if e is an edge that is light for S, then there exists a minimum-weight spanning tree T^* of G with $e \in T^*$.

3 Reverse Kruskal

Consider the following reversed version of Kruskal's algorithm: sort the edges in order of **decreasing** cost (that is, start with the most expensive edge first). Then, for each edge e in this order, delete e from the graph if e is not currently a bridge, that is, if the graph remains connected after removing e.

Prove that this algorithm computes a spanning tree of minimum weight (assuming the initial input graph is connected). Calculate its complexity.

4 Second-Best Spanning Tree

We are given as input a connected edge-weighted graph G where all edge weights are distinct. Recall that in this case there exists a unique minimum spanning tree T^*.

1. Show an example where there exist two distinct "second-best" spanning trees T_1, T_2. A spanning tree is second-best if it has cost less than or equal to that of all other trees except T^*.
2. Show that any second-best spanning tree can be obtained from T^* by removing some edge of T^* and adding another edge.
3. Use this to obtain an $O(nm)$-time algorithm that finds a second-best spanning tree.

5 Hard Tree Problems

Show that the following variations of the Minimum Spanning Tree problem have no polynomial-time algorithms (they are NP-complete).

1. Given an unweighted graph G, find a spanning tree of G of minimum **maximum degree**.
2. Given a weighted graph $G = (V, E)$, find a set of edges $E' \subseteq E$ of minimum weight such that $G' = (V, E')$ contains two edge-disjoint paths between any two vertices (or equivalently, $G' = (V, E')$ contains no bridges).
3. Given a weighted graph $G = (V, E)$ and a set $T \subseteq V$ of important vertices (terminals), find a set $E' \subseteq E$ of minimum weight such that $G' = (V, E')$ has all vertices of T in the same connected component. (This problem is known as the Steiner Tree problem.)

You may assume that the following problems are NP-complete and therefore have no polynomial-time algorithms[1]:

- Hamiltonian Path: Given $G = (V, E)$ decide if there exists a simple path in G that visits every vertex once.
- Hamiltonian Cycle: Given $G = (V, E)$ decide if there exists a Hamiltonian Path in G where the first and last vertices are adjacent.
- Minimum Vertex Cover: Given $G = (V, E)$, and integer k, decide if there exists a set $S \subseteq V$ of at most k vertices such that all edges have at least one endpoint in S (so, $G - S$ contains no edges).

[1] unless P=NP...