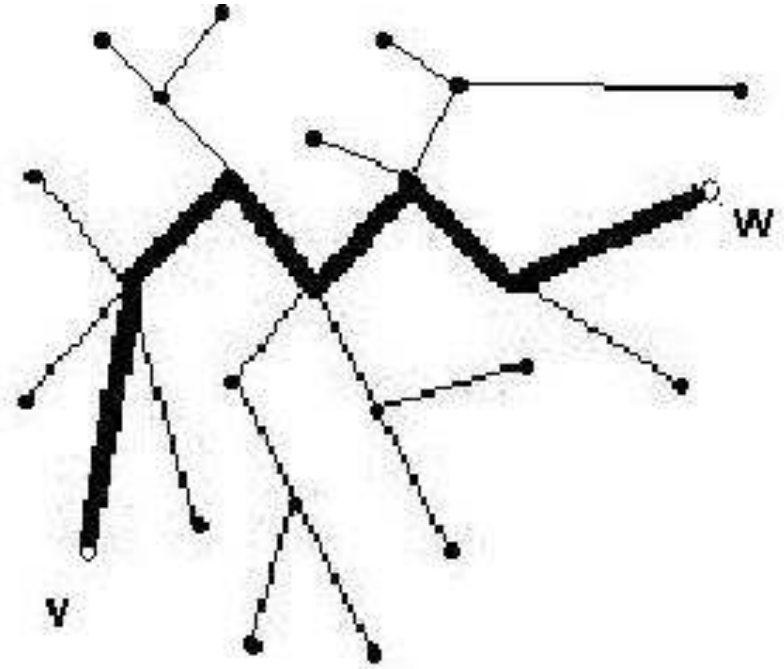
Discrete Mathematics

Trees

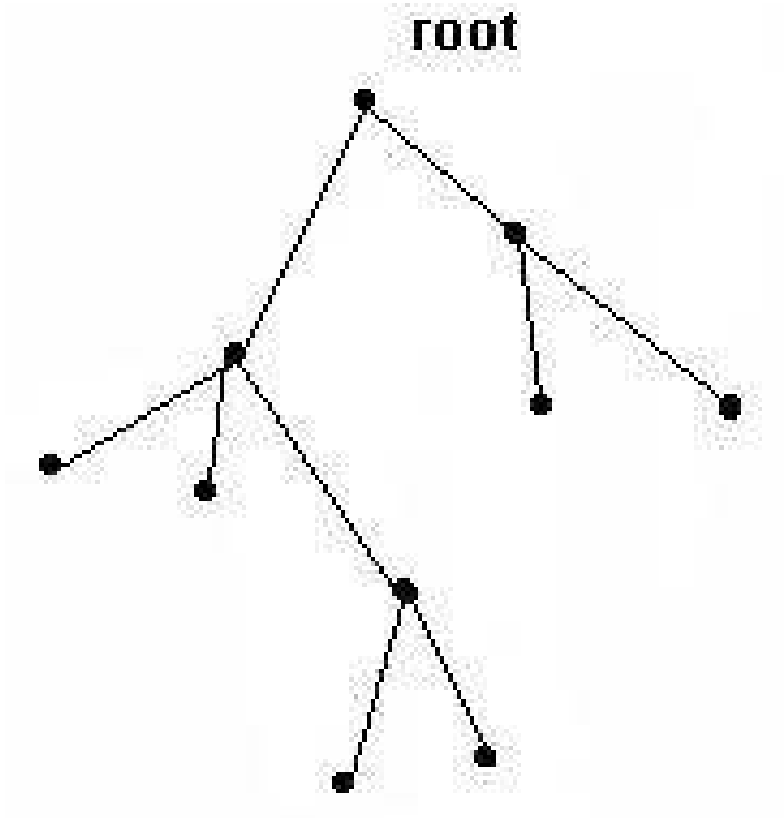
Introduction



A *(free) tree* T is

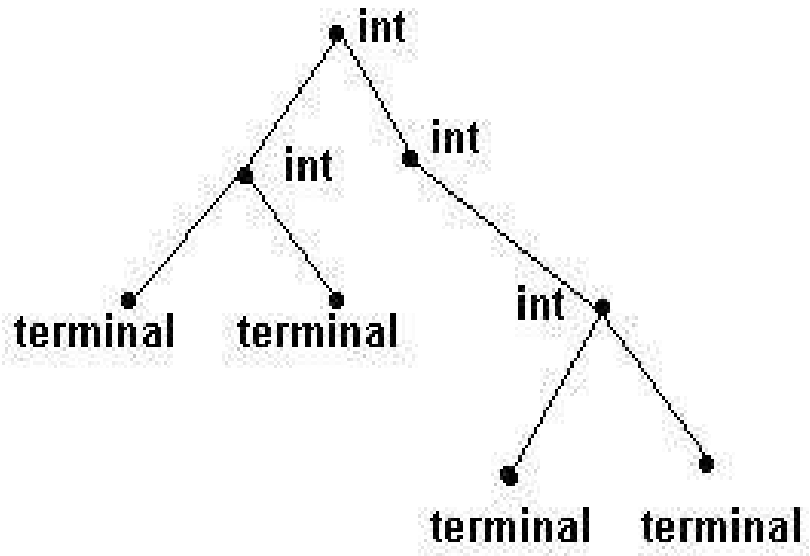
- A simple graph such that for every pair of vertices v and w
- there is a unique path from v to w

Rooted tree



A *rooted tree* is a tree where one of its vertices is designated the *root*

Internal and external vertices

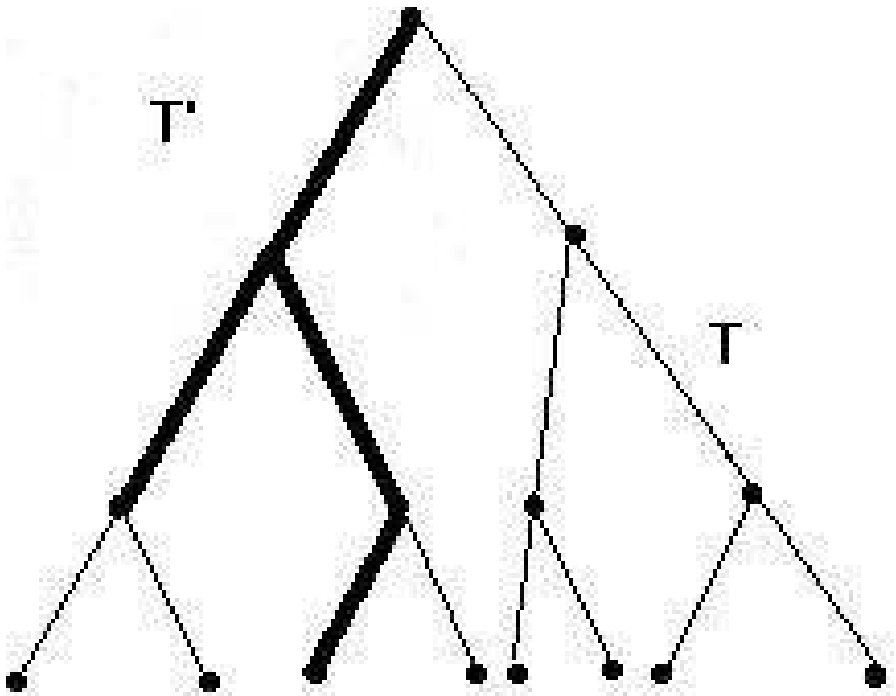


- An *internal vertex* is a vertex that has at least one child
- A *terminal* vertex is a vertex that has no children
- The tree in the example has 4 internal vertices and 4 terminal vertices

Subtrees

A subtree of a tree T is a tree T' such that

- $V(T') \subseteq V(T)$ and
- $E(T') \subseteq E(T)$



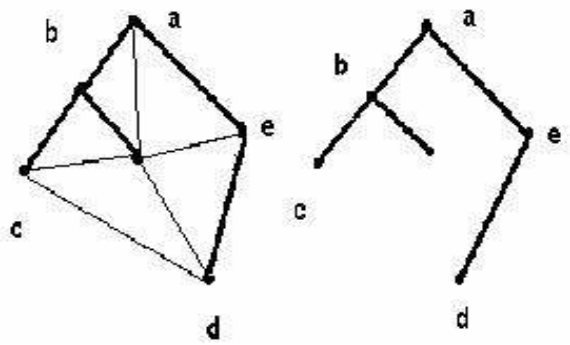
Spanning trees

Given a graph G, a tree T is a *spanning tree* of G if:

- T is a subgraph of G

  and

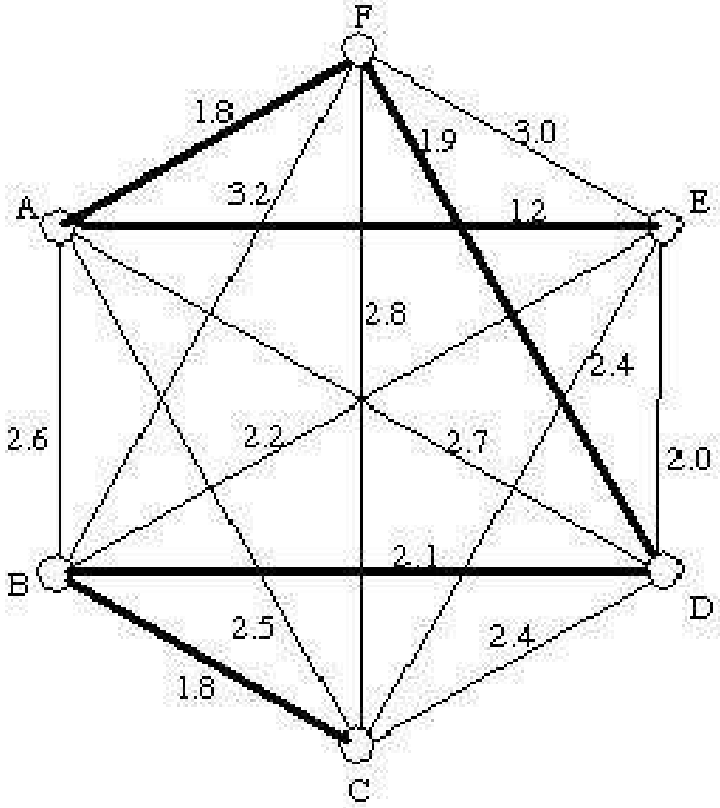
- T contains all the vertices of G



Minimal spanning trees

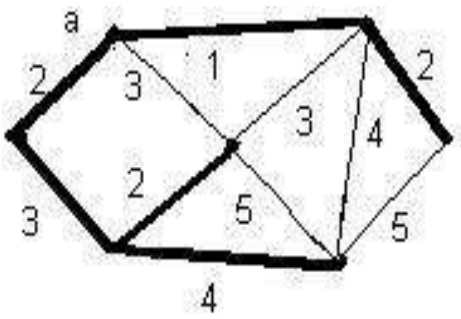
Given a weighted graph G, a *minimum spanning tree* is

- a spanning tree of G
- that has minimum “weight”



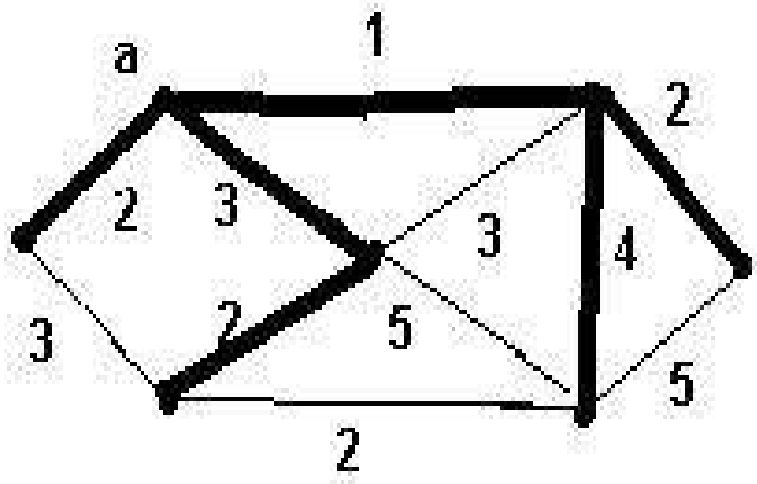
1. Prim's algorithm

- Step 0: Pick any vertex as a starting vertex (call it *a*). T = {a}.
- Step 1: Find the edge with smallest weight incident to *a*. Add it to T Also include in T the next vertex and call it *b*.
- Step 2: Find the edge of smallest weight incident to either *a* or *b.* Include in T that edge and the next incident vertex. Call that vertex *c*.
- Step 3: Repeat Step 2, choosing the edge of smallest weight that does not form a cycle until all vertices are in T. The resulting subgraph T is a minimum spanning tree.



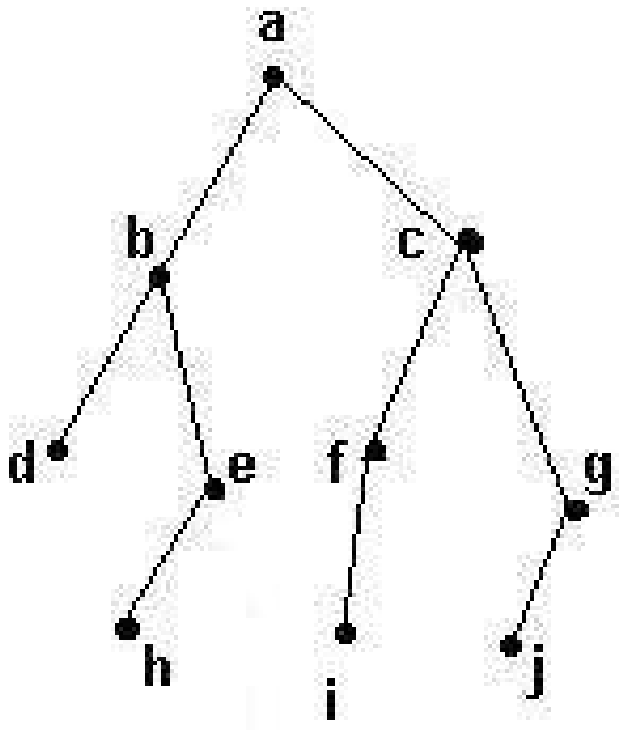
2. Kruskal's algorithm

- Step 1: Find the edge in the graph with smallest weight (if there is more than one, pick one at random). Mark it with any given color, say red.
- Step 2: Find the next edge in the graph with smallest weight that doesn't close a cycle. Color that edge and the next incident vertex.
- Step 3: Repeat Step 2 until you reach out to every vertex of the graph. The chosen edges form the desired minimum spanning tree.



Binary trees

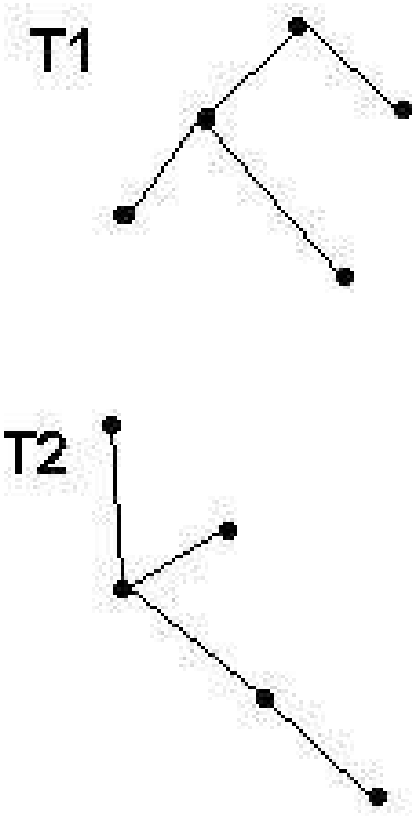
A *binary tree* is a tree where each vertex has zero, one or two children



Isomorphism of trees

Given two trees T_1 and T_2

- T_1 is *isomorphic* to T_2
- if we can find a one-to-one and onto function $f : T_1 \rightarrow T_2$
- that preserves the adjacency relation
  - i.e. if $v, w \in V(T_1)$ and $e = (v, w)$ is an edge in T_1, then $e' = (f(v), f(w))$ is an edge in T_2.




thank you