DISCRETE STRUCTURE

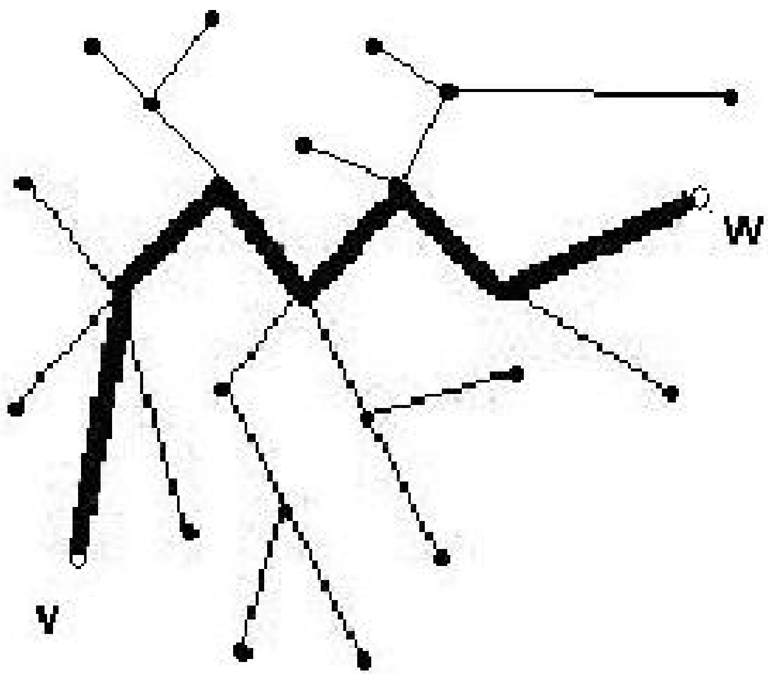
Lecture-28

Introduction to Tree & Spanning tree

Topics covered

- ❑ Introduction to tree
- ❑ Spanning tree
- ❑ Prim’s algorithm
- ❑ Kruskal’s algorithm

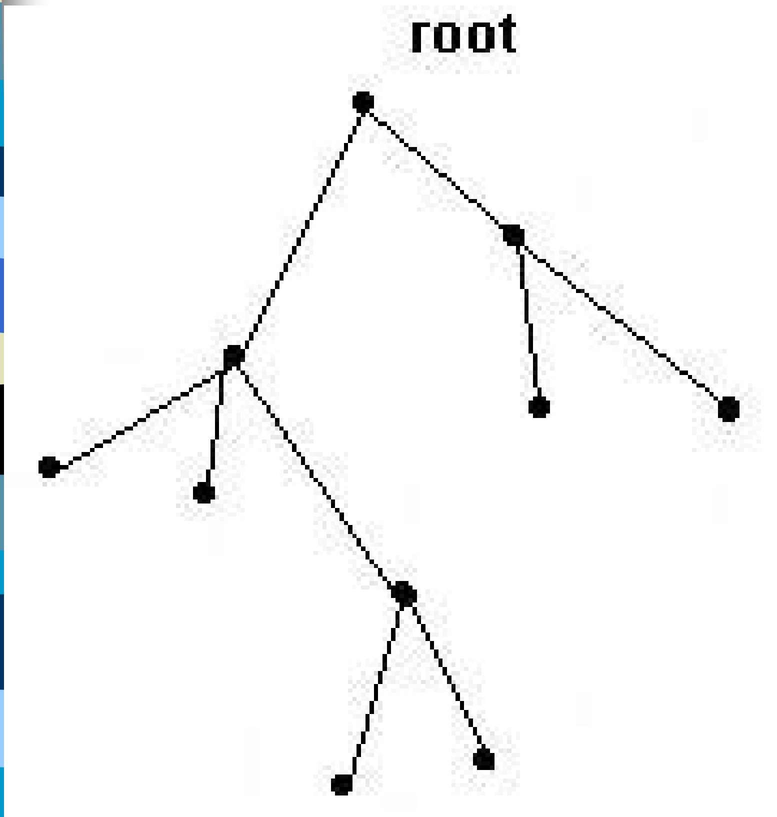
Introduction



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*

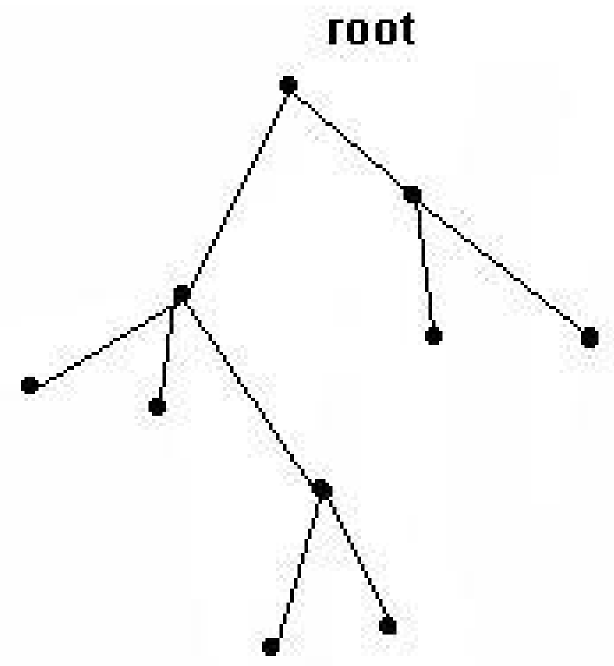
Level of a vertex and tree height

Let T be a rooted tree:

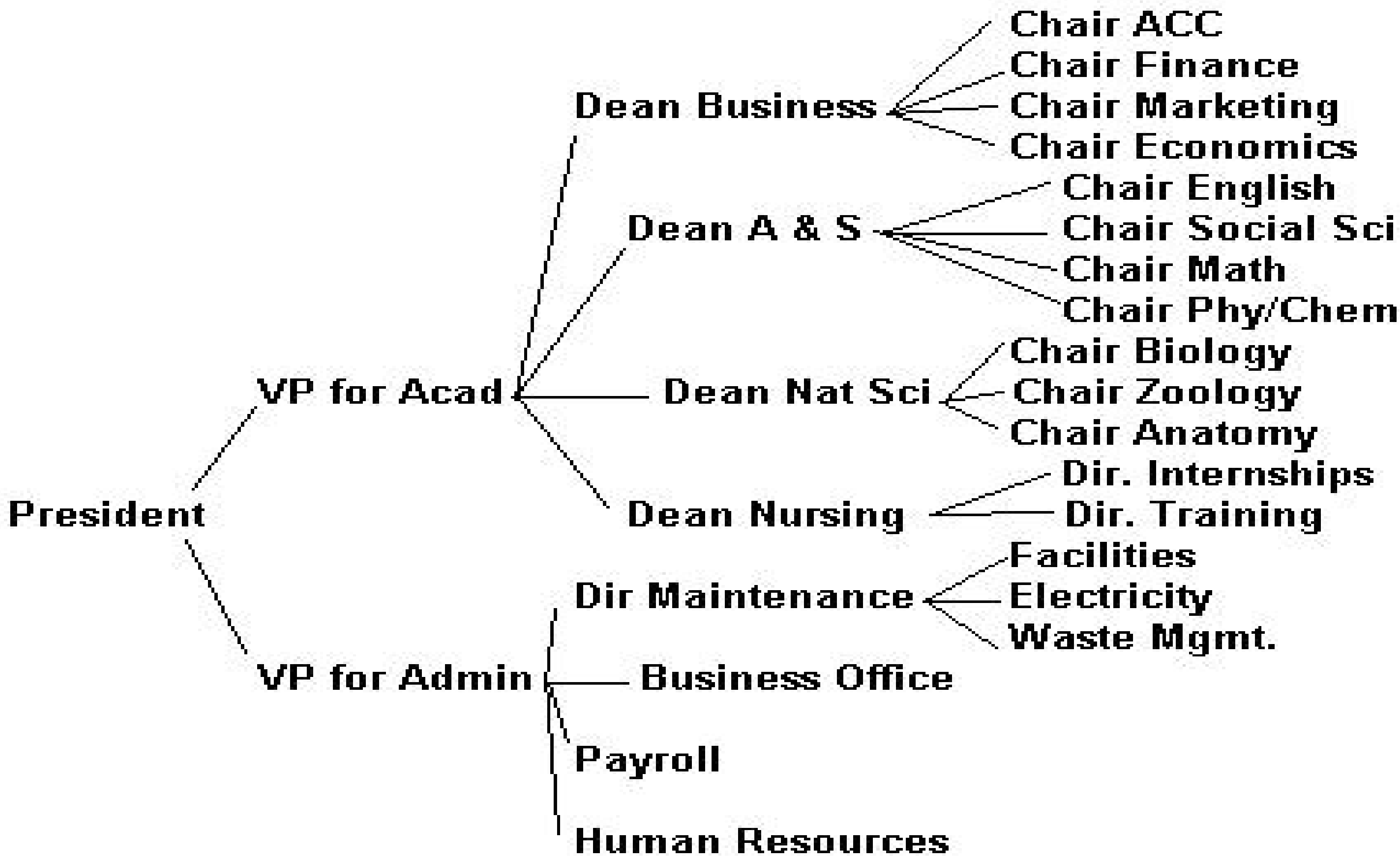
- The *level* $l(v)$ of a vertex **v** is the length of the simple path from **v** to the root of the tree
- The *height* h of a rooted tree T is the maximum of all level numbers of its vertices:

$$h = \max_{v \in V(T)} \{ l(v) \}$$

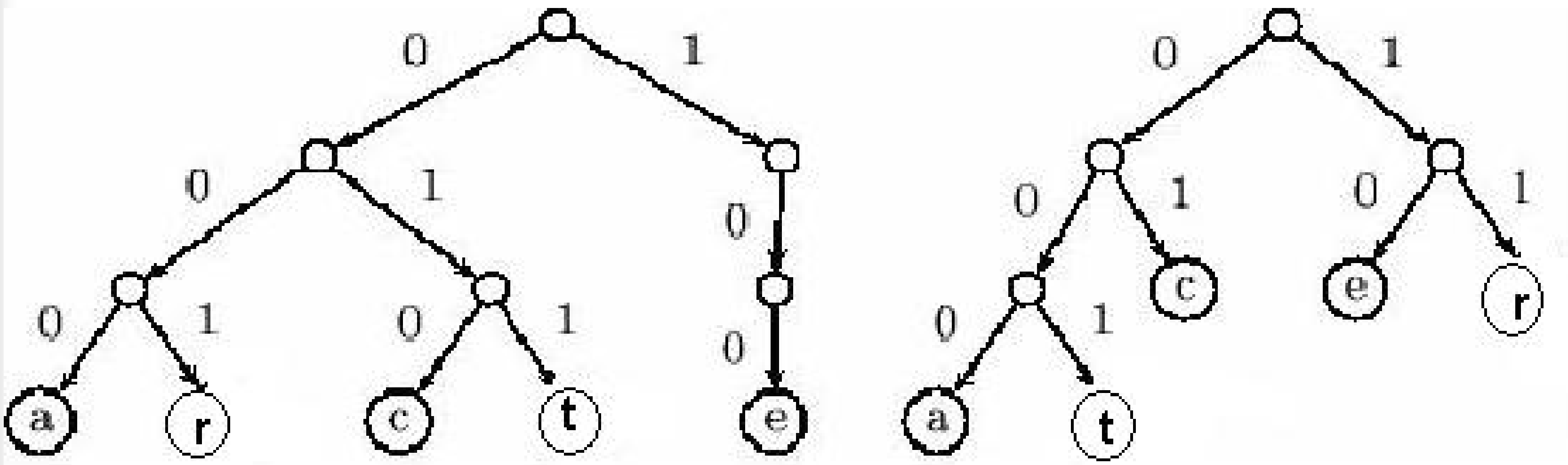
- Example:
  – the tree on the right has height 3



Organizational charts



Huffman codes



- On the left tree the word **rate** is encoded

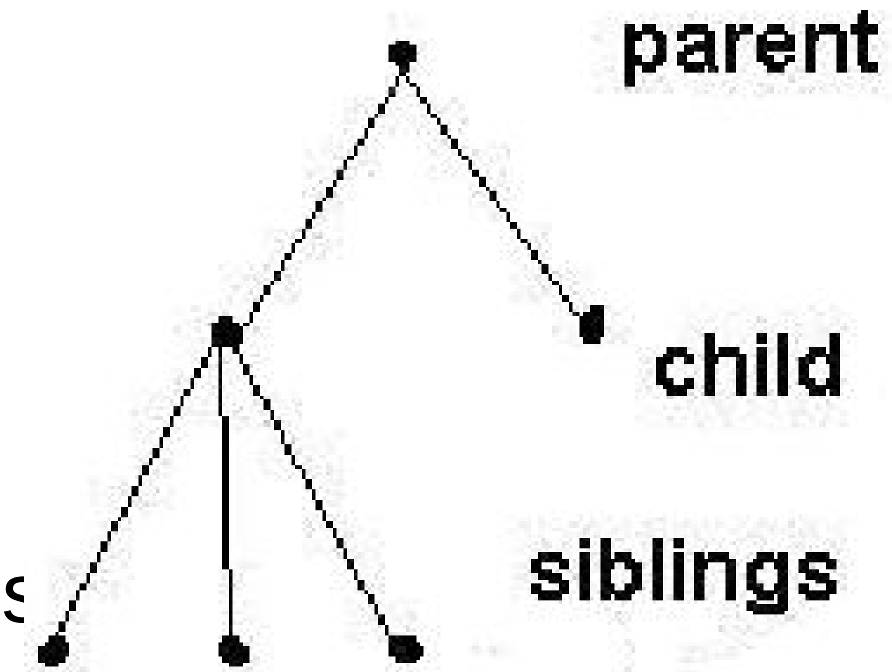
  001 000 011 100

- On the right tree, the same word **rate** is encoded

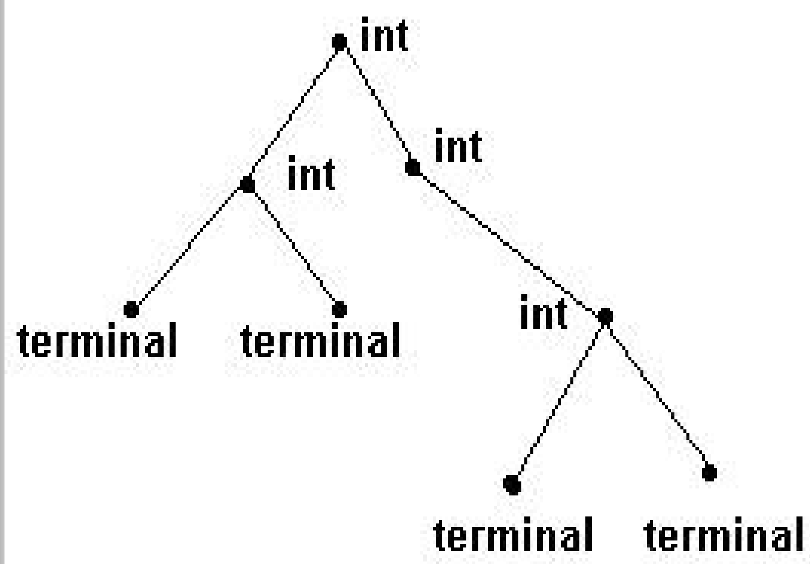
  11 000 001 10

Tree Terminology

- Parent
- Ancestor
- Child
- Descendant
- Siblings
- Terminal vertices
- Internal vertices
- Subtrees



Internal and external vertices

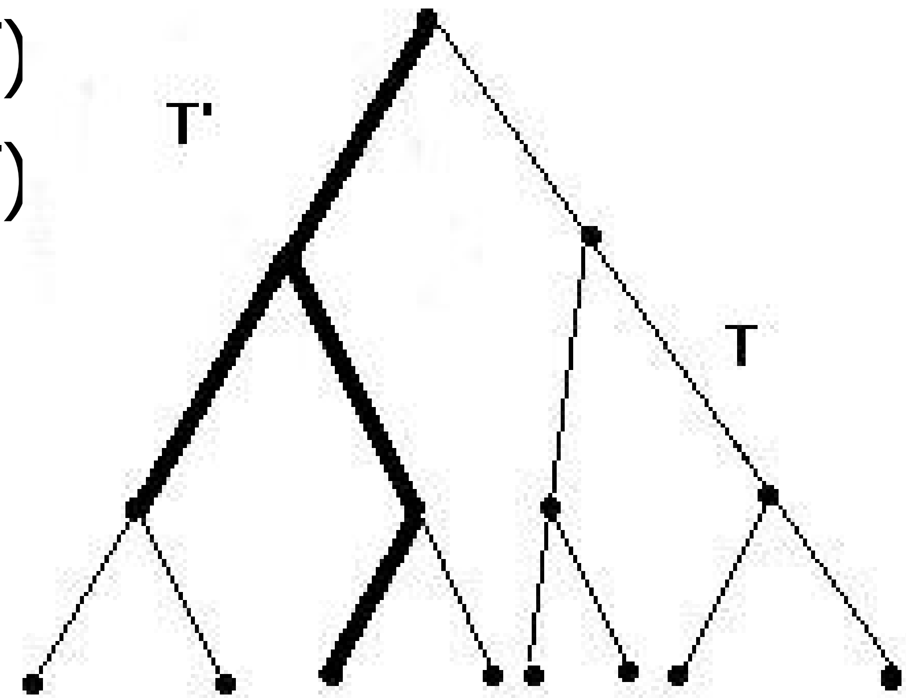


- An *internal vertex* is a vertex that has at least one child
- A *terminal* vertex is a vertex that has no children
- The tree in the example has 4 internal vertices and 4 terminal vertices

Subtrees

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

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



Characterization of trees

Theorem

If T is a graph with n vertices, the following are equivalent:

a) T is a tree

b) T is connected and acyclic

– ("acyclic" = having no cycles)

c) T is connected and has n-1 edges

d) T is acyclic and has n-1 edges

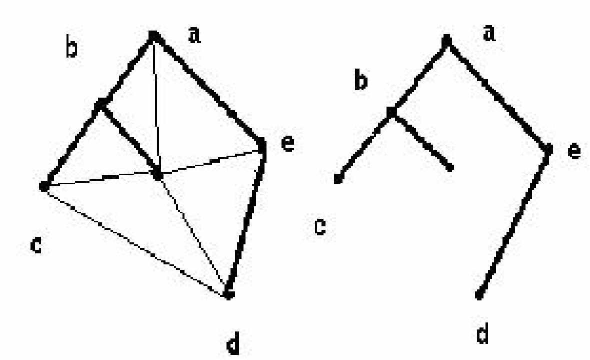
Spanning trees

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

- T is a subgraph of G

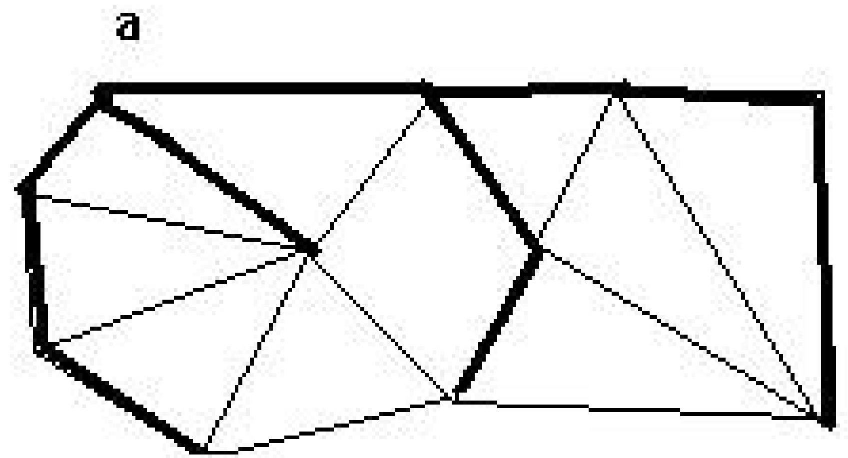
  and

- T contains all the vertices of G

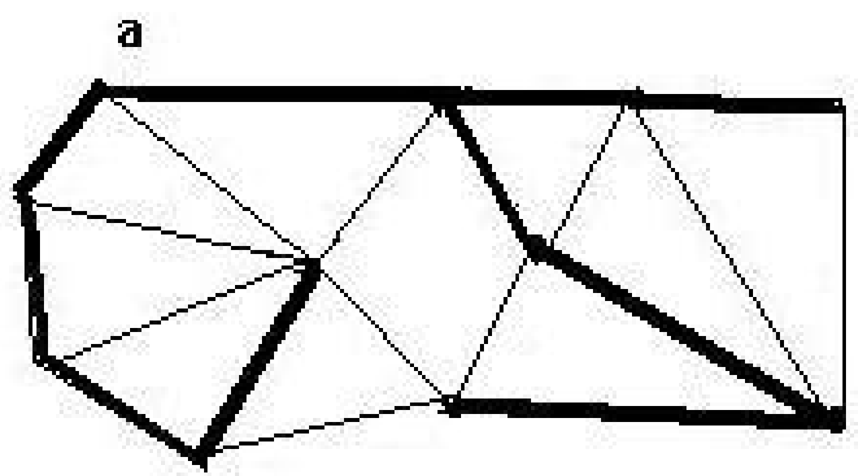


Spanning tree search

- Breadth-first search method



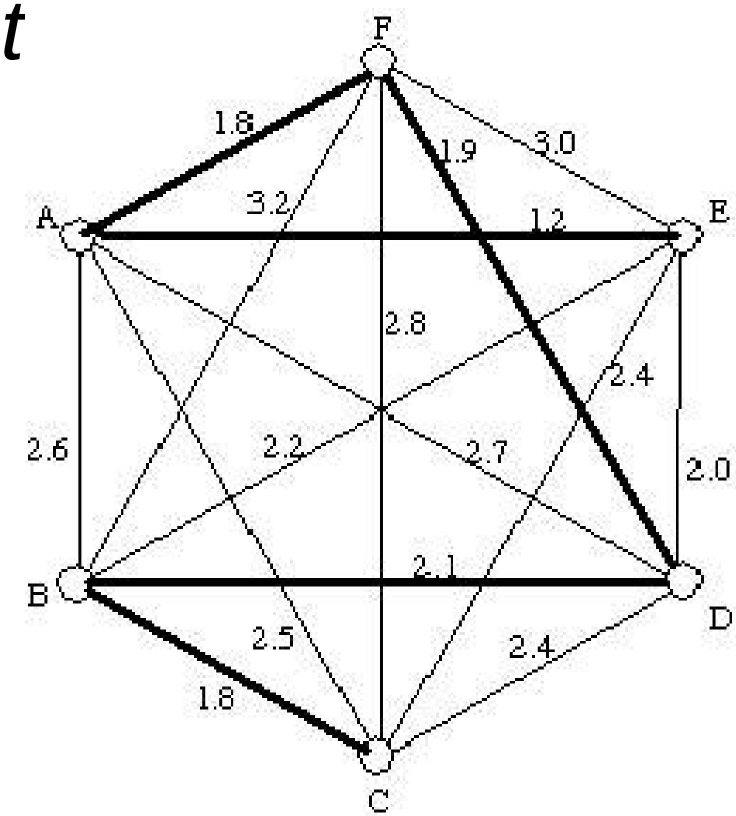
- Depth-first search method (backtracking)



Minimal spanning trees

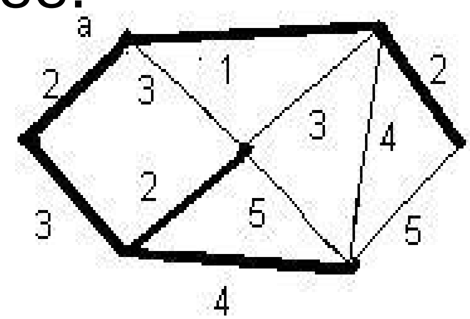
Given a weighted graph G, a *minimum spanning t* is

- a spanning tree of G
- that has minimum "weight"



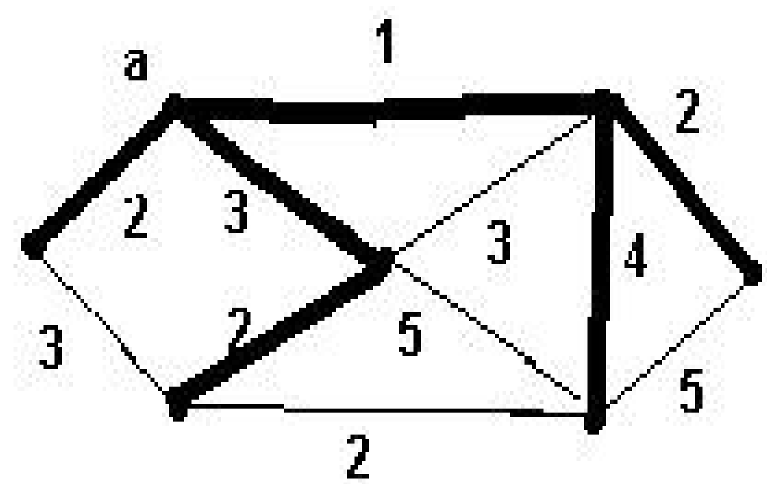
1. Prim's algorithm

- 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.



Application & Scope of research

- Application
  1. Representing hierarchical data
  2. Representing sorted list of data of data
  3. As a workflow for composing digital images for visual effects
  4. Routing algorithm algorithms
  5. Game
- Scope of research: Network