department of Computer Science University College Cork

CS1113 Trees

Lecturer:

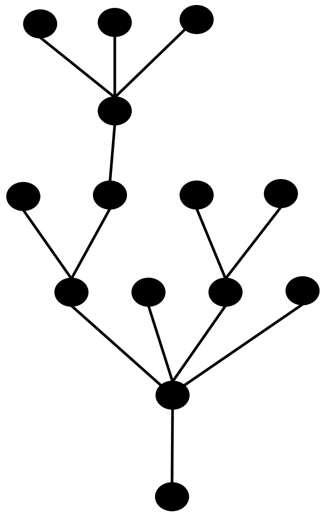
Professor Barry O'Sullivan

Office: 2.65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

http://osullivan.ucc.ie/teaching/cs1113/

Trees



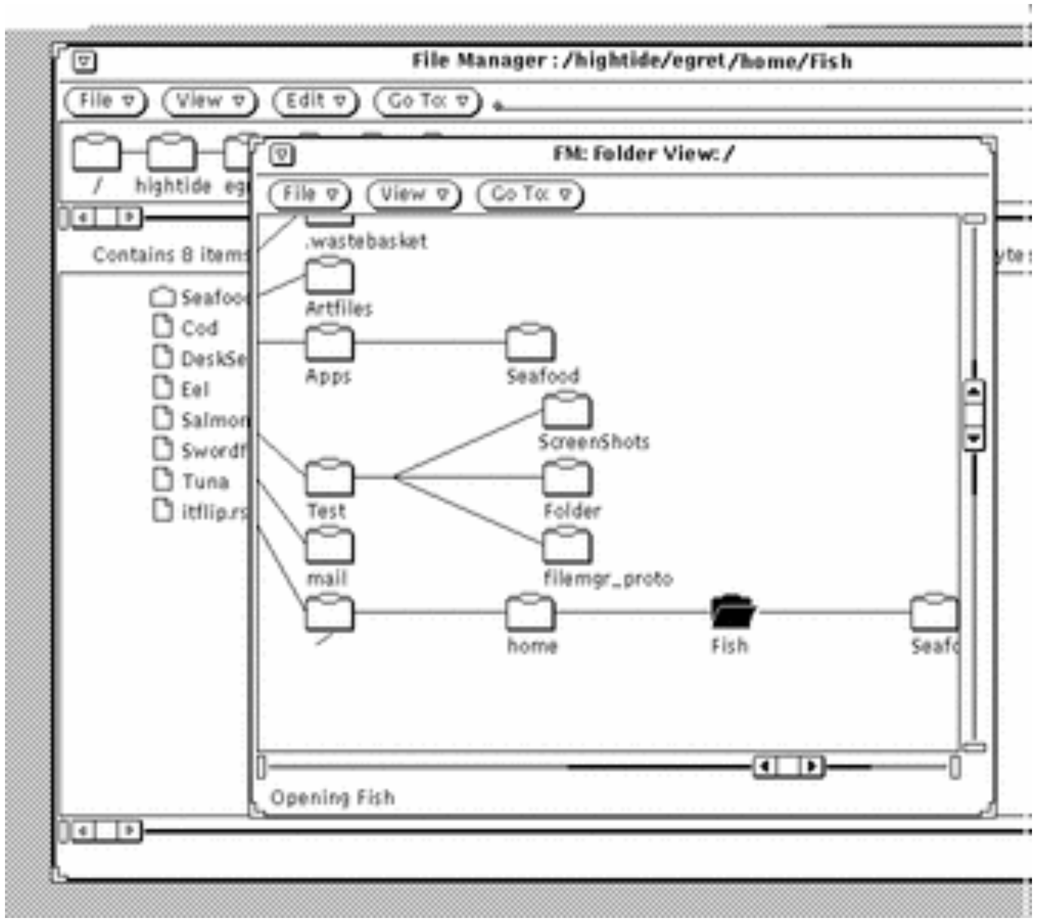
applications of trees

cycles
trees
rooted trees

spanning trees
Prim's algorithm

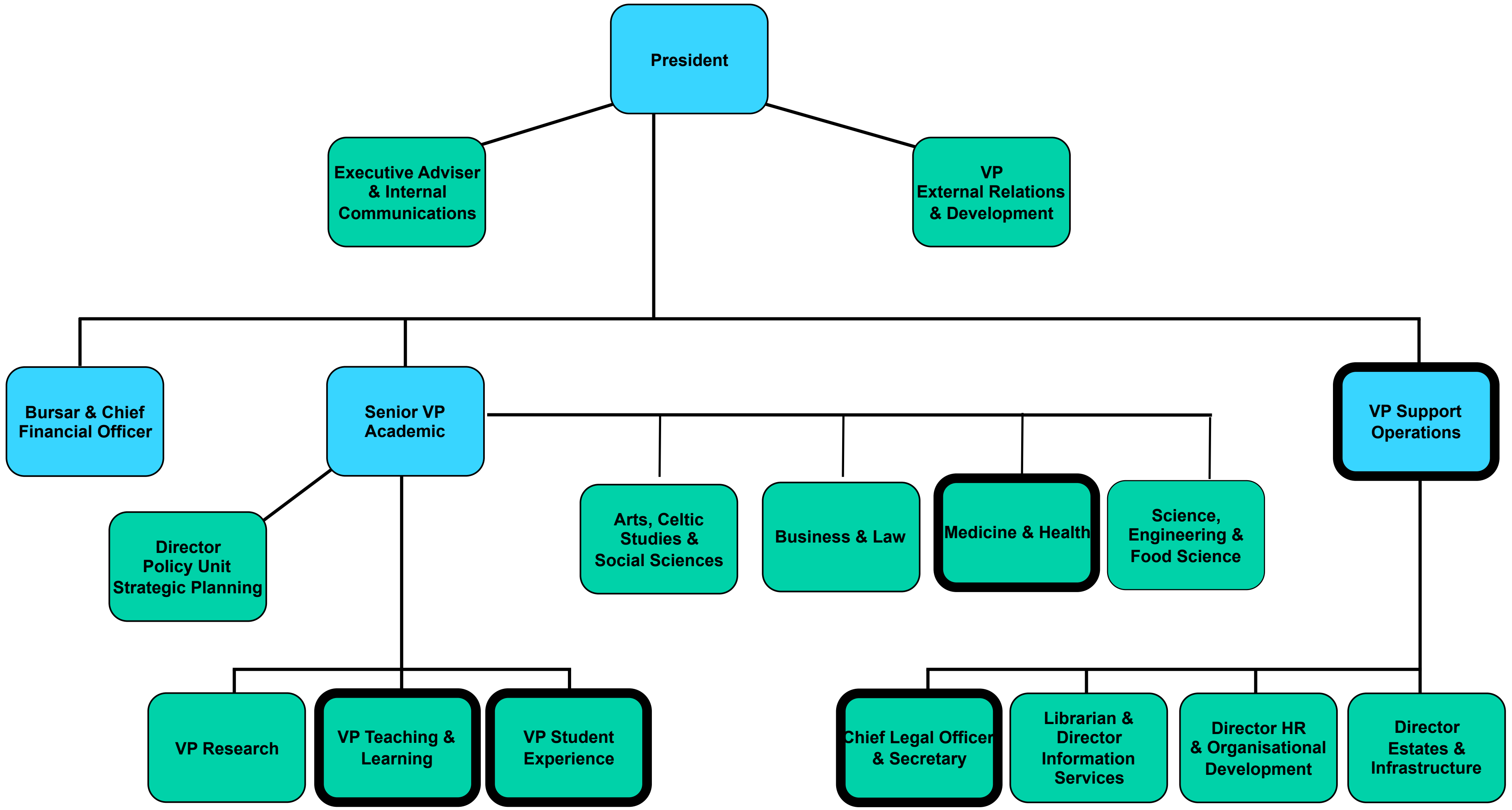
Uses of Trees

File systems:



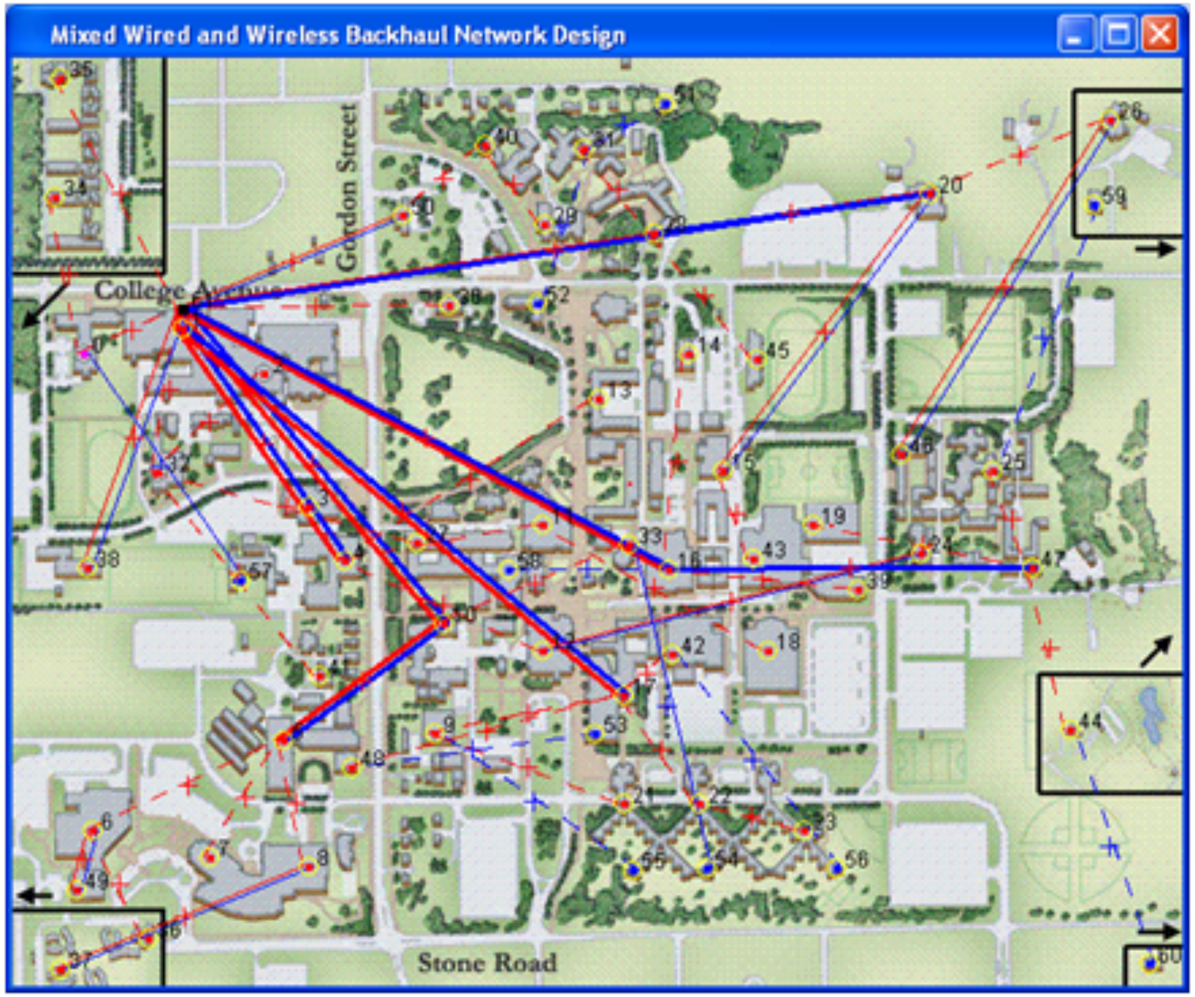
from www.sun.com

organisational structure:

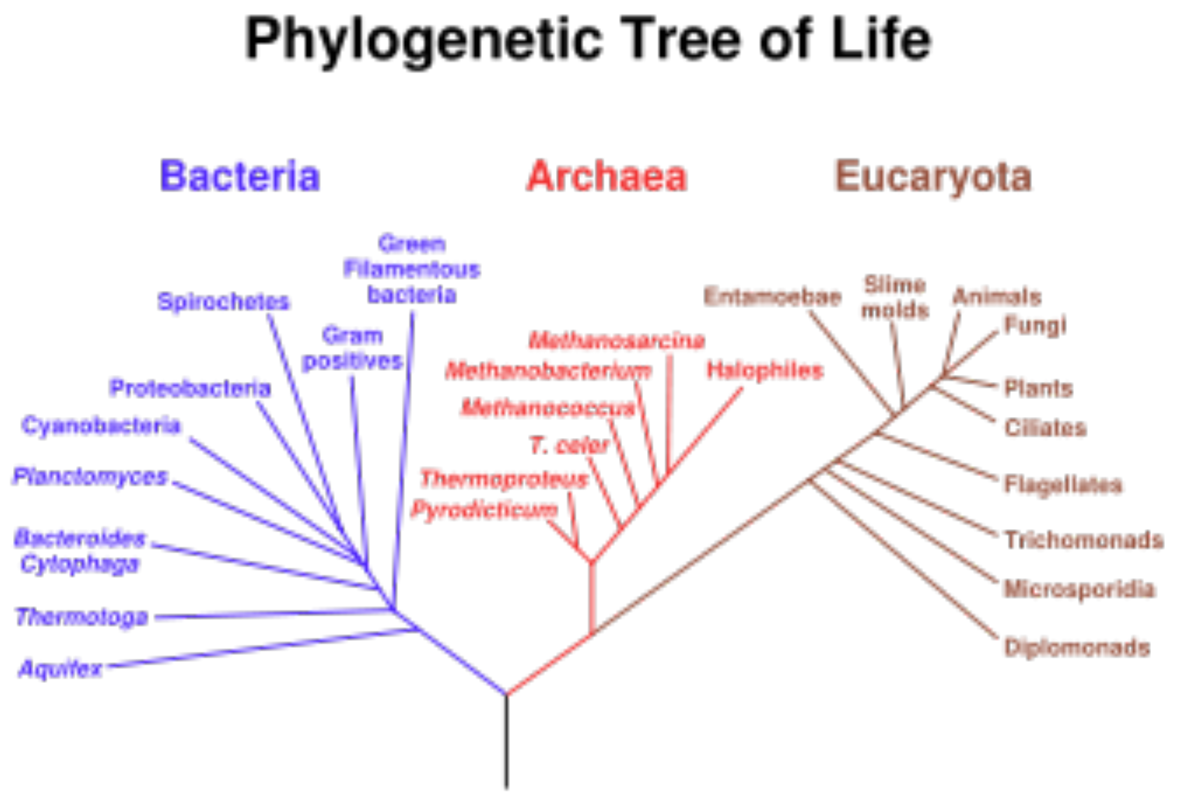

President
Executive Adviser & Internal Communications
VP External Relations & Development
Bursar & Chief Financial Officer
Senior VP Academic
VP Support Operations
Director Policy Unit Strategic Planning
Arts, Celtic Studies & Social Sciences
Business & Law
Medicine & Health
Science, Engineering & Food Science
VP Research
VP Teaching & Learning
VP Student Experience
Chief Legal Officer & Secretary
Librarian & Director Information Services
Director HR & Organisational Development
Director Estates & Infrastructure

from UCC president's address (modified)

mobile phone network backhaul:

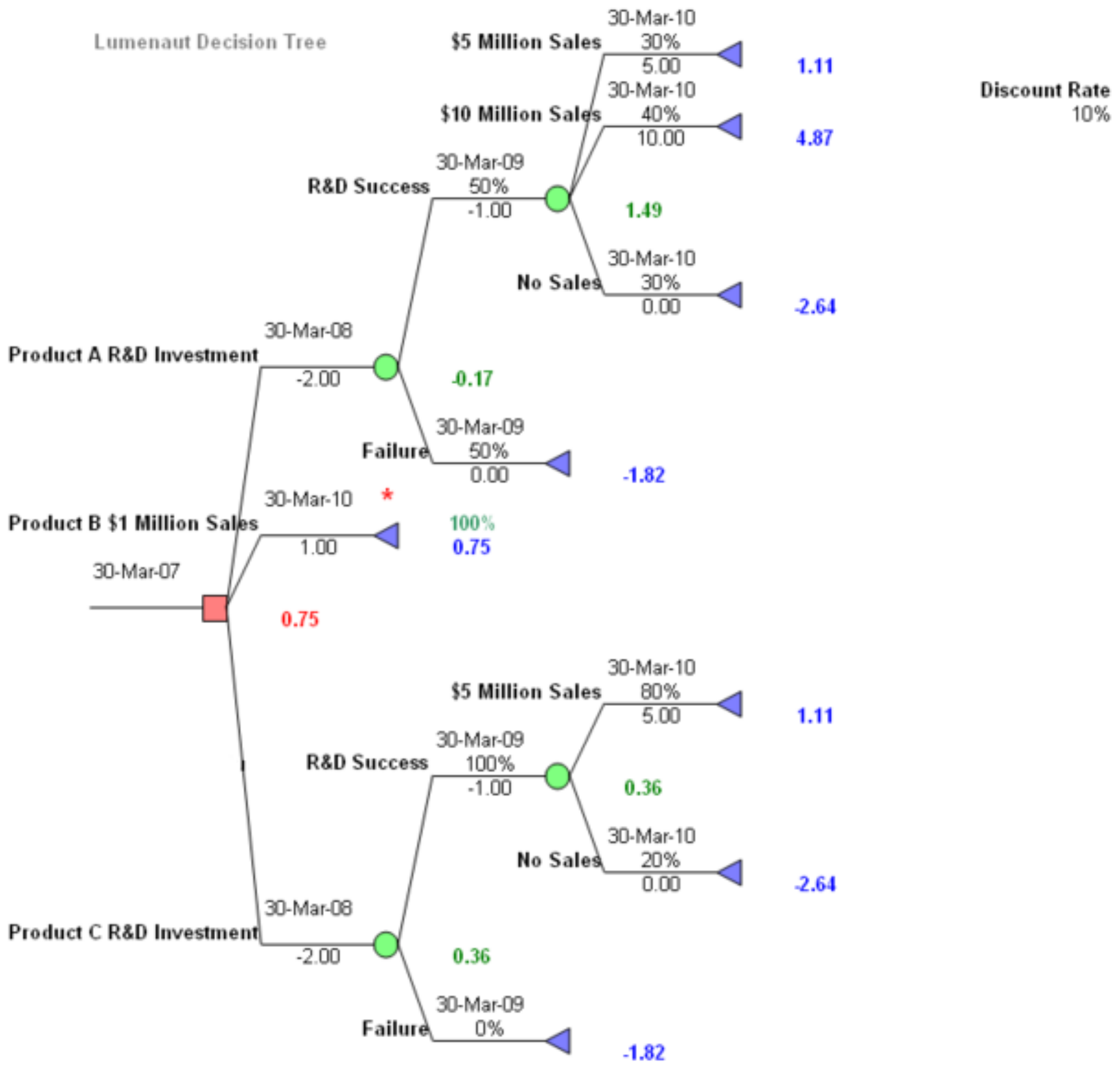


phylogenetic tree:



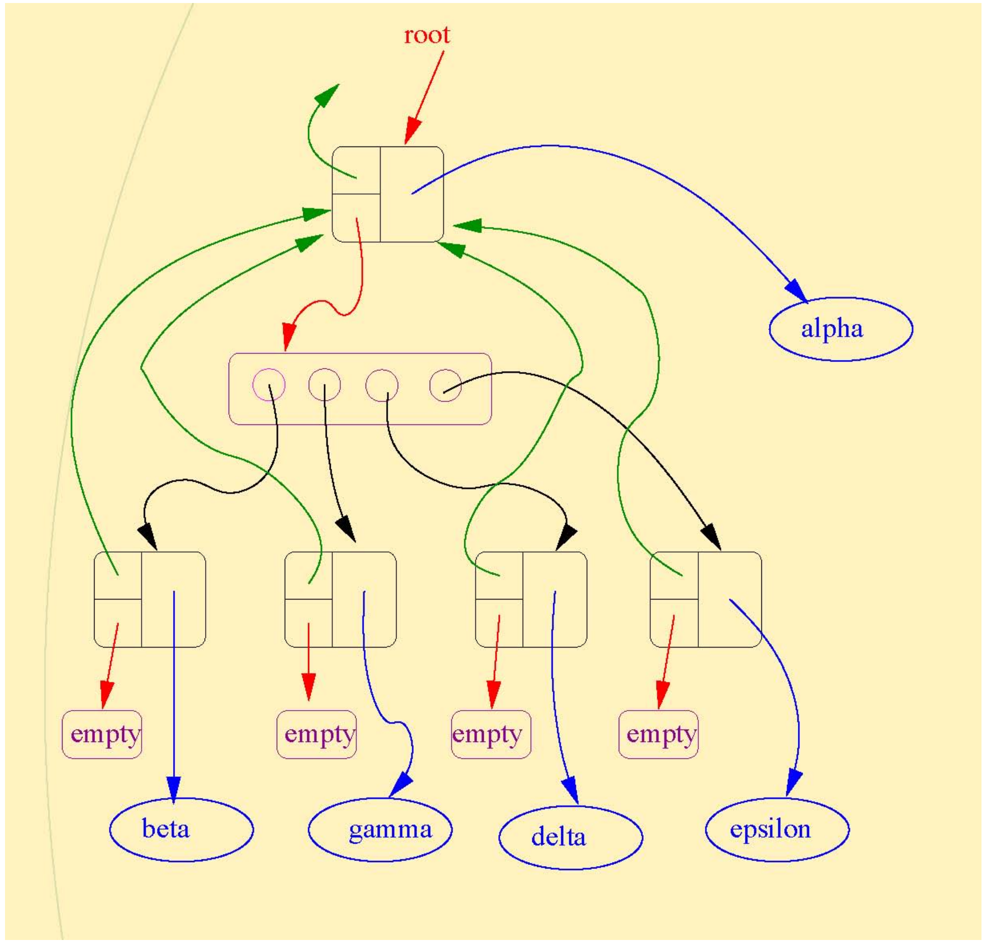
from wikipedia

Decision trees:



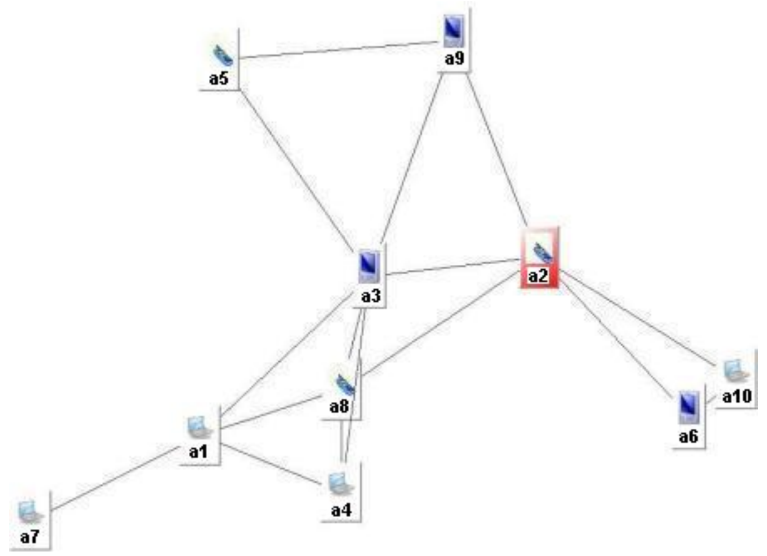
from wikipedia

Computer science algorithms and data structures:

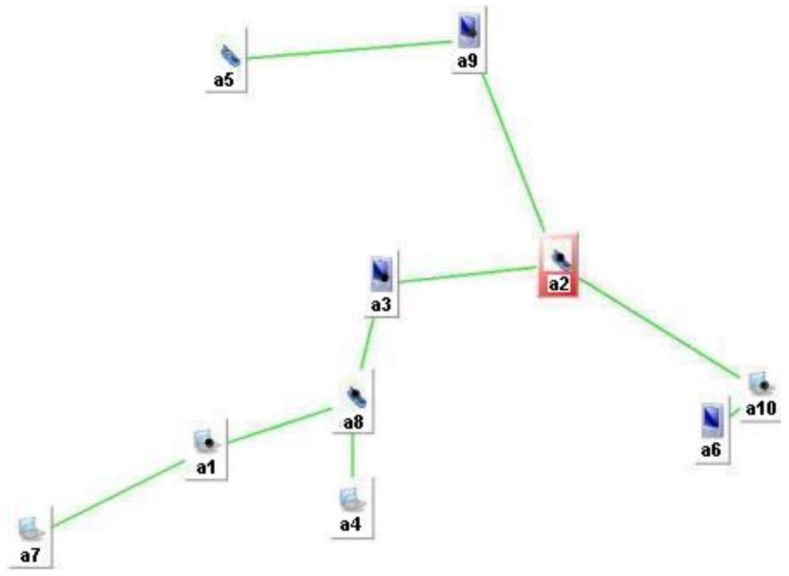


from CS2201 Data Structures, by Kieran Herley

mobile ad hoc networks broadcast trees:



mobile radios, links show ability to communicate directly

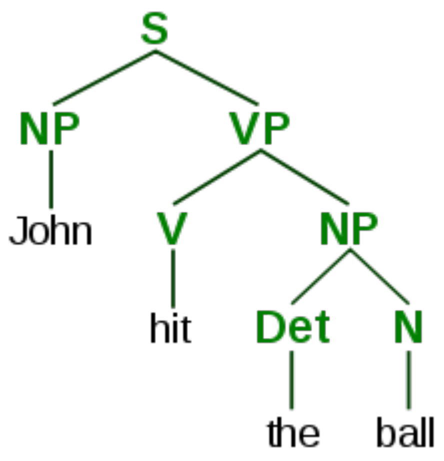


rooted tree of active links, with minimum total broadcast power requirements

propositional logic syntax trees

$(p \wedge (q \vee r)) \to \neg p$

$$\begin{tikzcd}
 & & & \to \arrow[dll] \arrow[dr] & \\
 & \wedge \arrow[dl] \arrow[dr] & & & \neg \arrow[d] \\
p & & \vee \arrow[dl] \arrow[dr] & & p \\
 & q & & r &
\end{tikzcd}$$

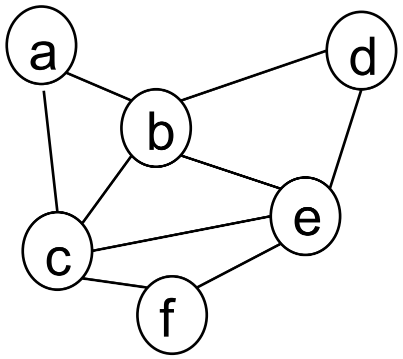


analysing structure of natural language sentences

Cycles

A cycle is a circuit with the following properties

- it contains at least one edge
- no edge is visited twice
- no vertex is visited twice except the start/finish vertex

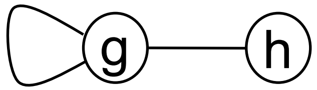


<*(b,e),(e,f),(f,c),(c,b)*> is a cycle

<*(a,c),(c,f),(f,e),(e,d),(d,b),(b,a)*> is a cycle

<(*b,c*),(*c,e*),(*e,f*),(*f,c*),(*c,a*),(*a,b*)> is not a cycle (*c* twice)

<*(d,e),(e,d)* > is not a cycle (edge d--e twice)

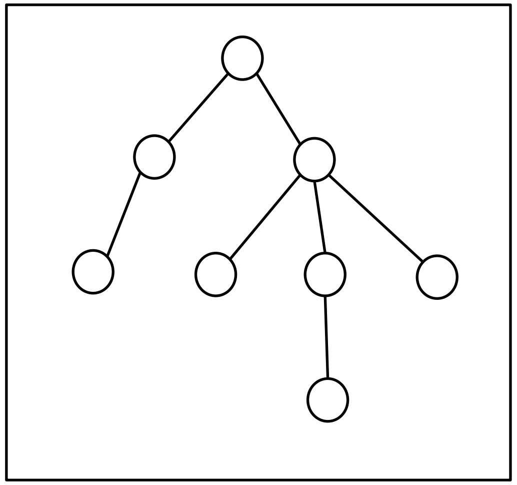


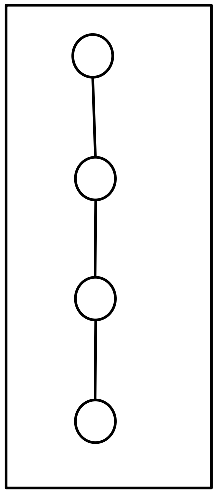
<(*g,g*)> is a cycle

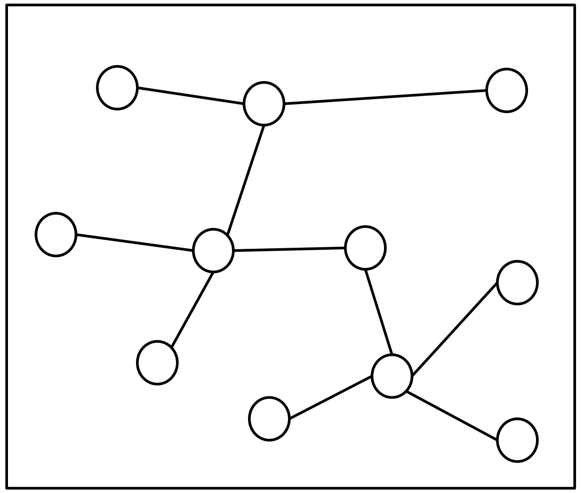
Trees

A tree is a connected undirected simple graph with no cycles

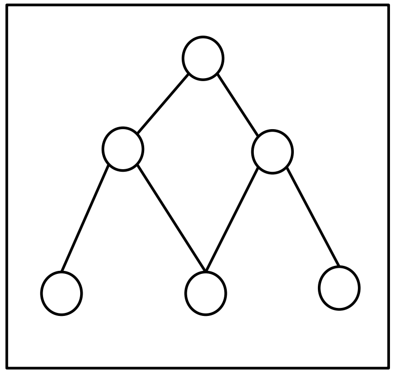
trees:

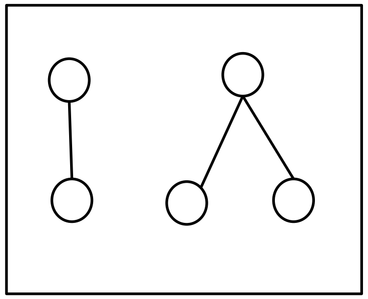


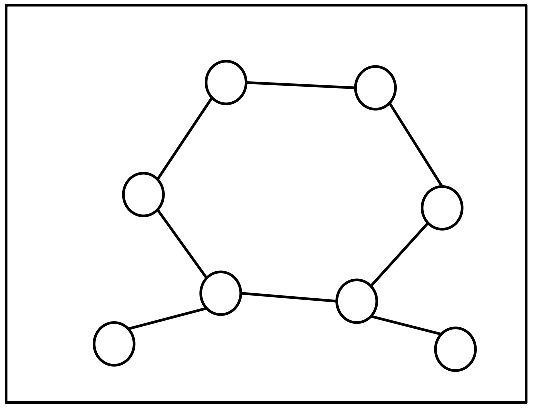




not trees:



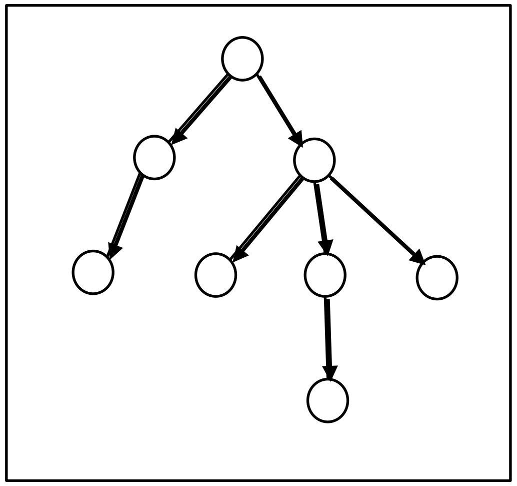


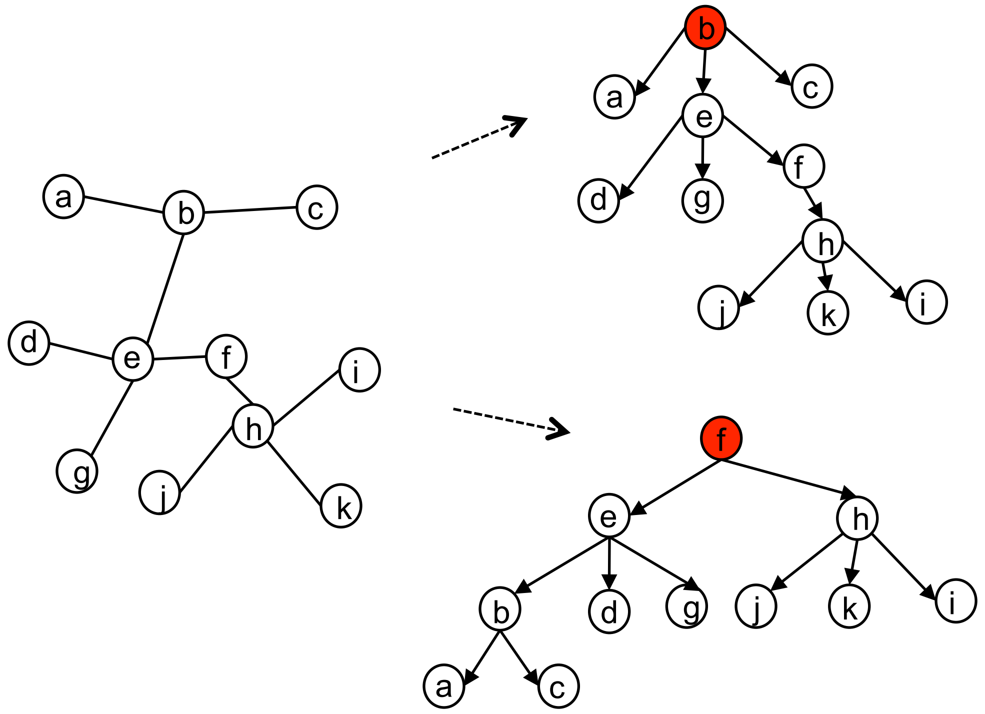


Rooted Trees

Usually, when we think of trees, we assume there is a root.

A rooted tree is a tree in which one of the vertices is designated the root, and all edges are then directed away from that root.





Describing vertices in rooted trees

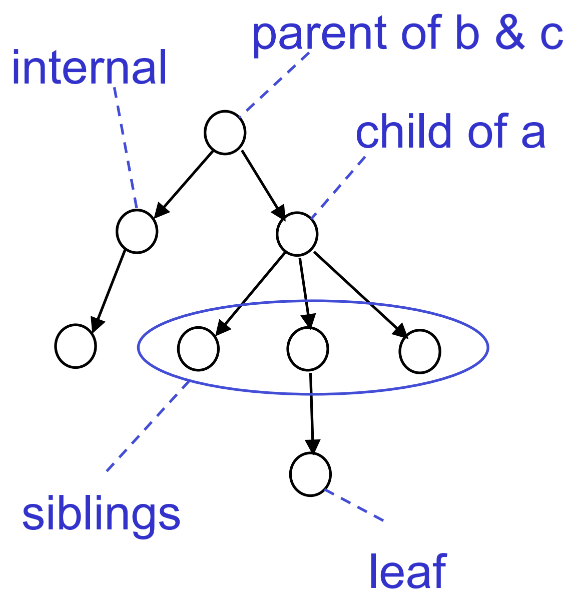
If there is a directed edge from *x* to *y*, then *x* is the parent of *y*, and *y* is a child of *x*.

If two vertices *y* and *w* have the same parent, then they are siblings of each other.

A vertex with no children is a leaf. A vertex with children is an internal vertex.

The ancestors of a vertex *v* are all the vertices in the path from *v* to the root (except for *v* itself).

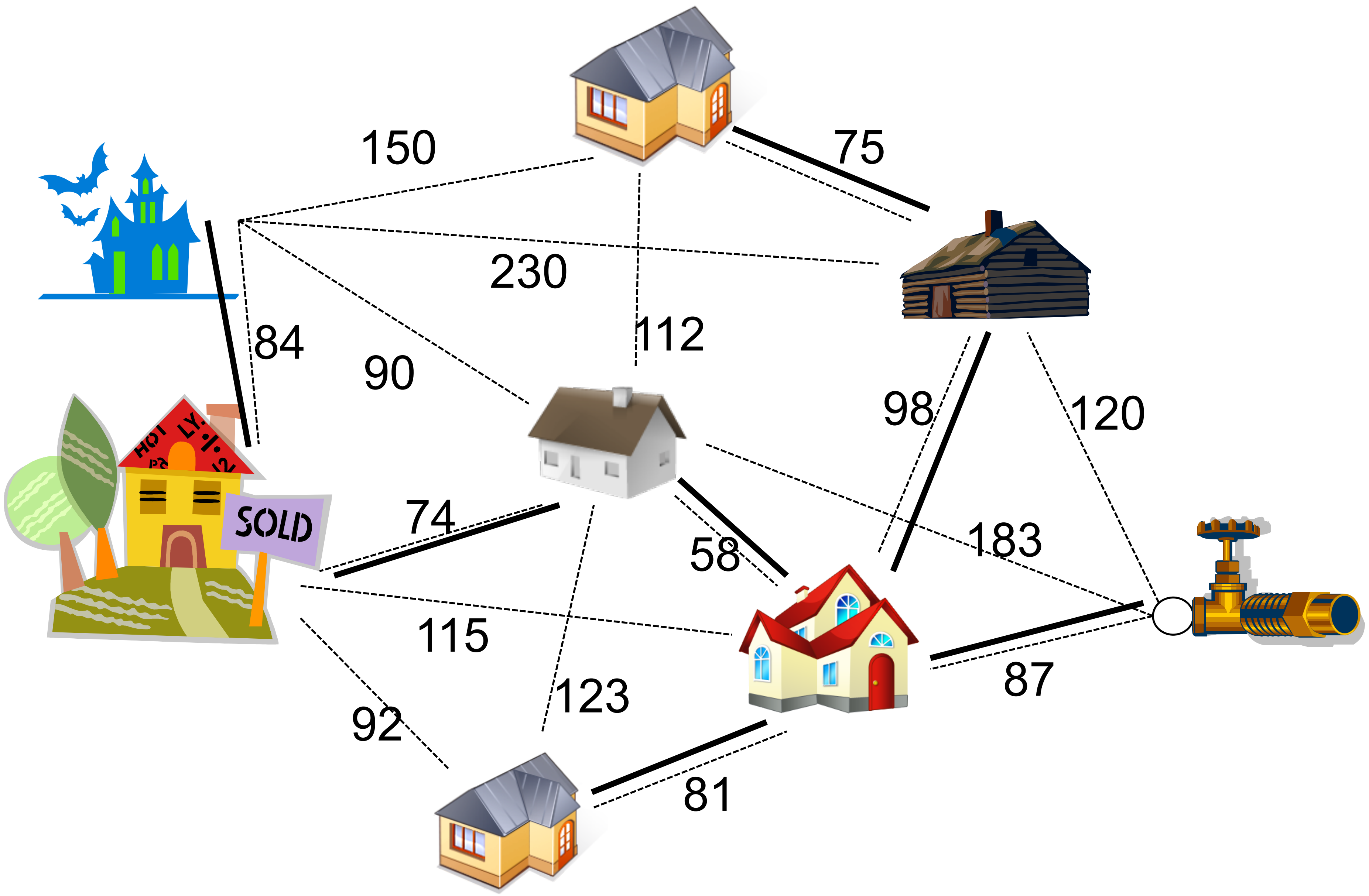
The descendants of a vertex *v* are all the vertices that have *v* as an ancestor.



Properties of trees

- between any pair of vertices, there is exactly one simple path
- adding a new edge to a tree creates a cycle
- removing any edge from a tree makes it a disconnected graph
- every rooted tree has at least one leaf
- in any tree with 2 or more vertices, there is at least one vertex with degree=1
- any tree with *n* vertices has exactly (*n-1*) edges

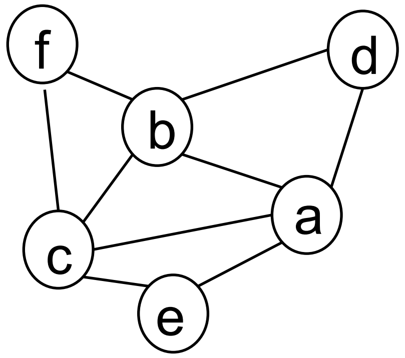
All of these statements can be proved easily ...

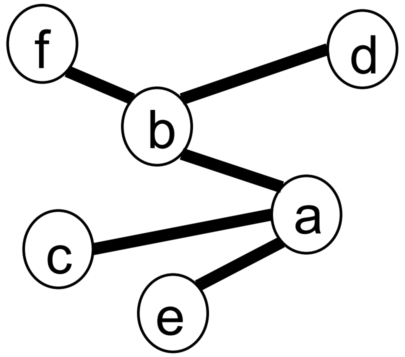

150
75
230
84
90
112
98
120
74
58
183
115
87
92
123
81
SOLD

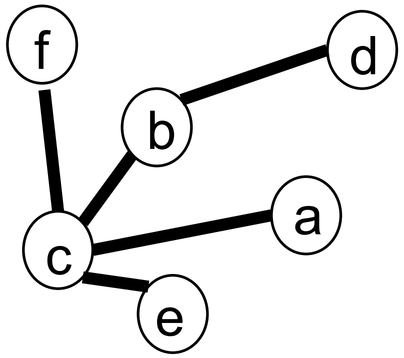
Spanning trees

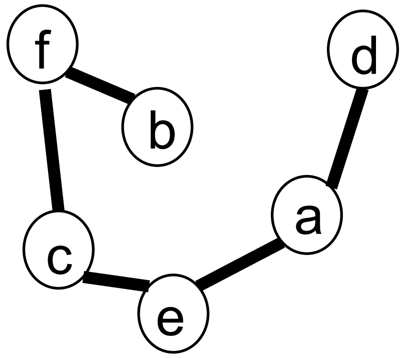
Let G=(V,E) be a simple graph. A spanning tree of G is a sub-graph of G that contains every vertex in V.

G:









Find 3 different spanning trees of G

A simple graph is connected if and only if it has a spanning tree

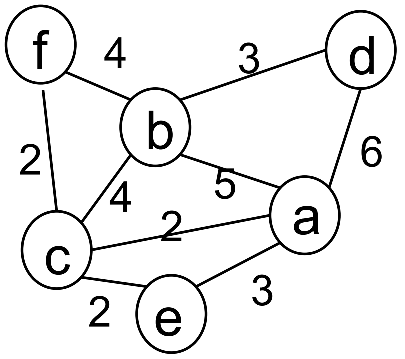
Proof

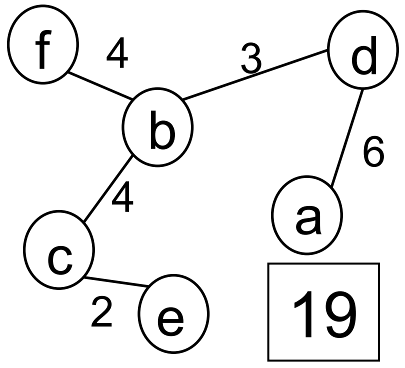
First, suppose G=(V,E) is simple graph which has a spanning tree. The tree includes all vertices in V, and by definition, all the vertices are connected, so G must be connected.

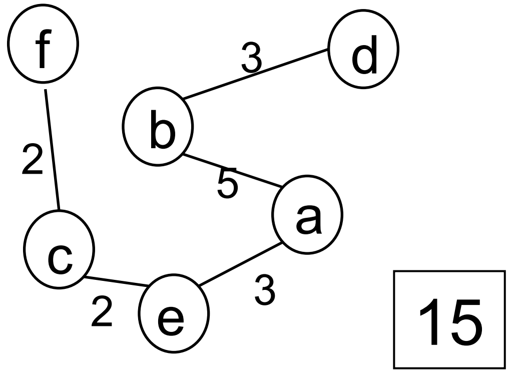
Second, suppose G=(V,E) is a connected simple graph. If G is already a tree, then G is its own spanning tree. Suppose now G is not a tree, and so it must have a cycle. We can remove any one of the edges in the cycle, and the resulting graph must still be connected and still contains all vertices. We keep repeating this process until the resulting graph has no cycles. At this point, we have a simple connected graph with no cycles, containing all vertices of G – i.e. a spanning tree of G.

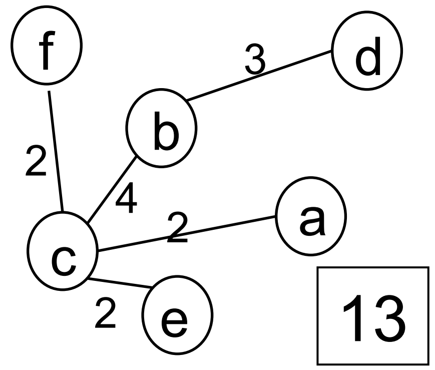
Minimal spanning trees

Consider a simple connected graph with weights on its edges. Different spanning trees will use different edges, and so the costs will be different.









How do we find the spanning tree with the minimum sum of weights?

Informal algorithm

Pick any vertex. Pick the cheapest edge that links to a vertex we haven't picked yet. Now pick the cheapest edge that connects one of our chosen vertices to a vertex we haven't picked yet. Keep doing the same step until we have selected every vertex. The vertices and the edges that we have chosen give us a spanning tree.

At each stage, we are picking the cheapest way to connect a new vertex to the tree we are creating. This style of algorithm is called greedy – at each step, we greedily pick what seems to be the best option.

Note: we are NOT growing the tree out in a path from the root to the leaves – at each stage we add a new vertex and connect it anywhere in the previous tree that is cheapest

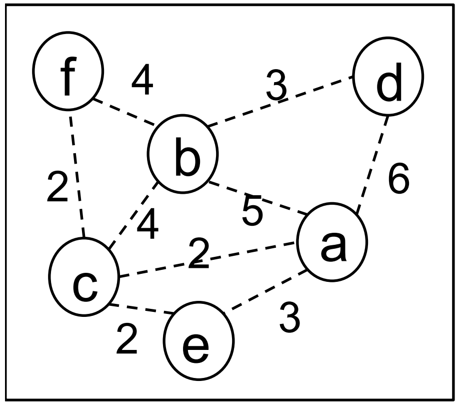
Prim's Algorithm

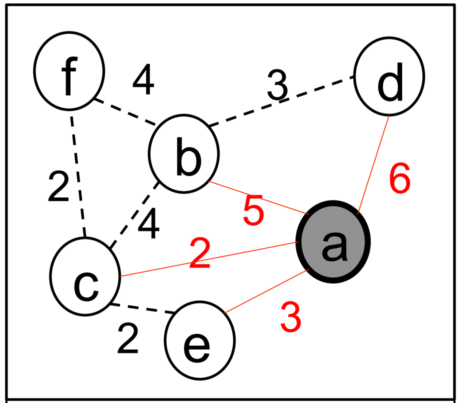
```
Algorithm: Prim's
Input: connected undirected graph G = (V,E) with edge
       weights and n vertices
Output: a spanning tree T=(V,F) for G

1. T := {v}, where v is any vertex in V
2. F := { }
3. for each i from 2 to n
4.     e := {w,y}, an edge with minimum weight in E such
           that w is in T and y is not in T
5.     F := F ∪ {e}
6.     T := T ∪ {y}
7. return (T,F)
```

step 4: find all edges that connect to vertices in T, but which would not create a cycle in the tree, and choose the one with lowest weight

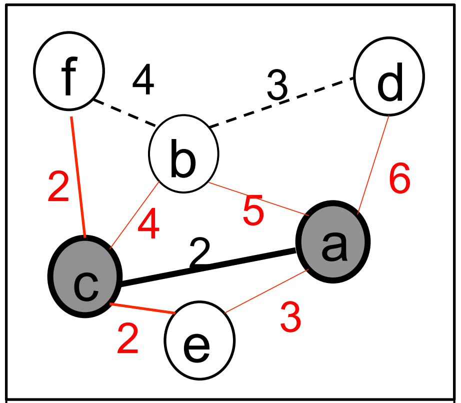
It is possible to prove that Prim's algorithm does find the minimum spanning tree.

Prim's algorithm: example

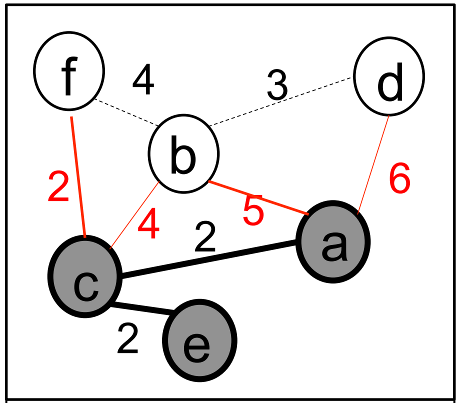




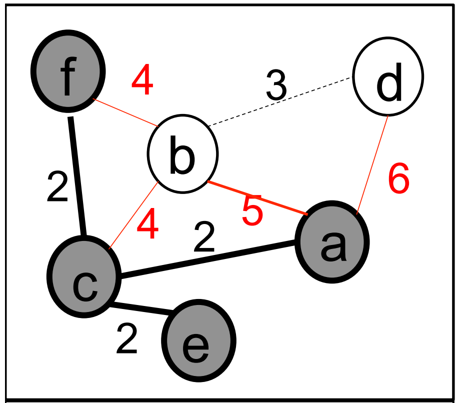
T={a}
F={ }



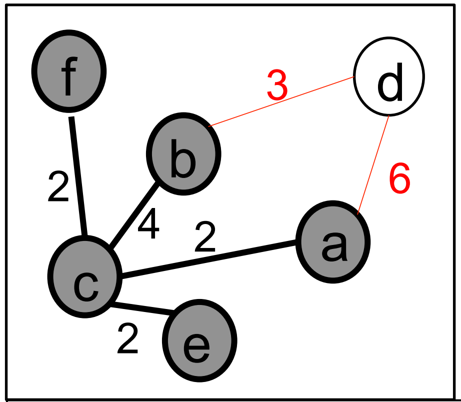
T={a,c}
F={ {a,c}}



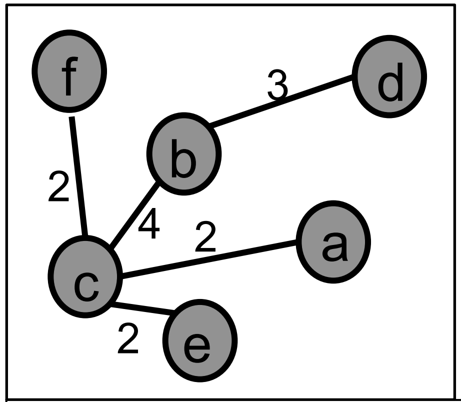
T={a,c,e}
F={ {a,c},{c,e}}



T={a,c,e,f}
F={ {a,c},{c,e},{c,f}}

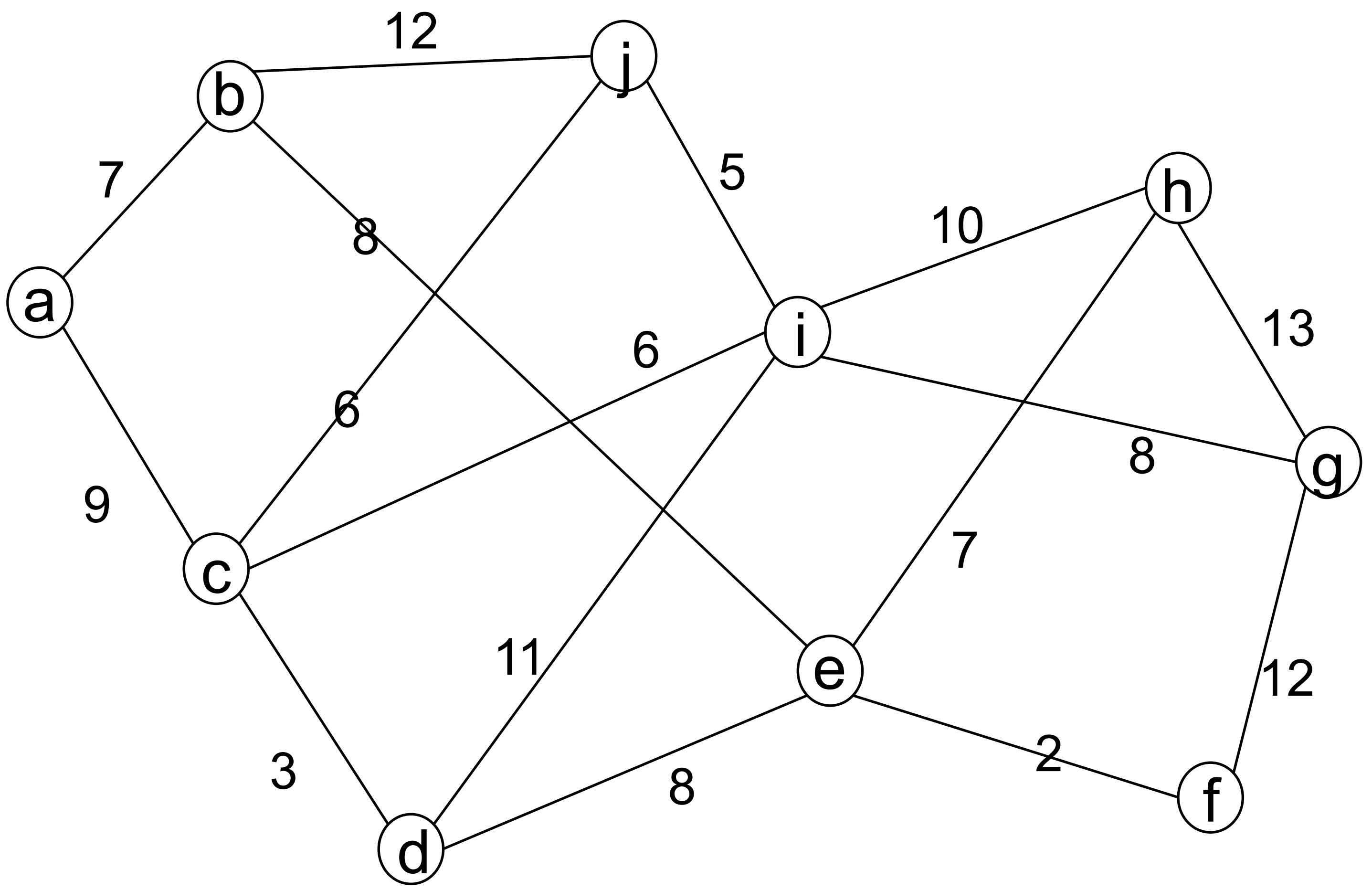


T={a,c,e,f,b}
F={{a,c},{c,e},{c,f} {c,b}}



T={a,c,e,f,b,d}
F={ {a,c},{c,e},{c,f} {c,b},{b,d}}

Exercise: find the minimum spanning tree using Prim's algorithm (start with vertex a)



Next lecture ...

Counting, and analysing algorithms