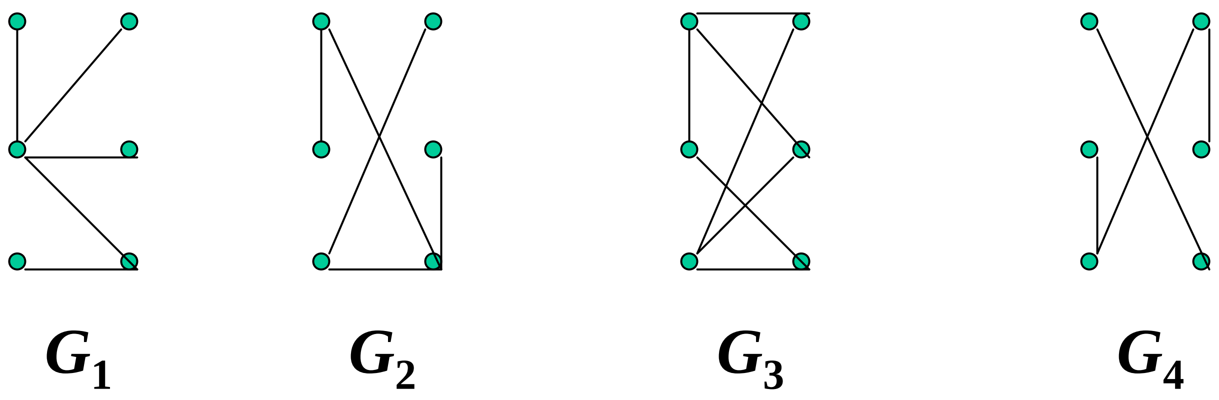
Chapter 10: Trees

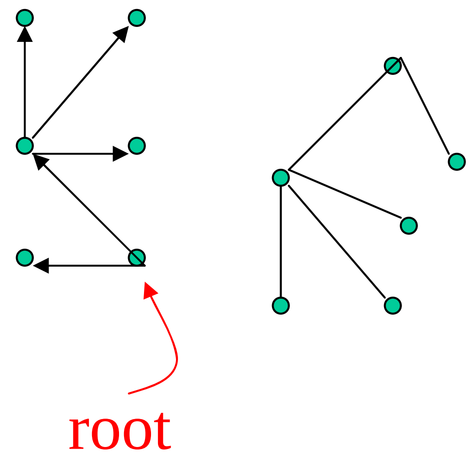
§ 10.1 Trees

- **Definition:** A ***tree*** is a connected undirected graph with no simple circuits.
- Which is a tree? G_1 G_2



Root of a Tree

- **Theorem:** An undirected graph is a *tree* iff there is a unique simple path between any two of its vertices.
- **Definition:** A *rooted tree* (directed graph) is a tree in which one vertex has been designated as the root and every edge is directed away from the root.



Terminologies

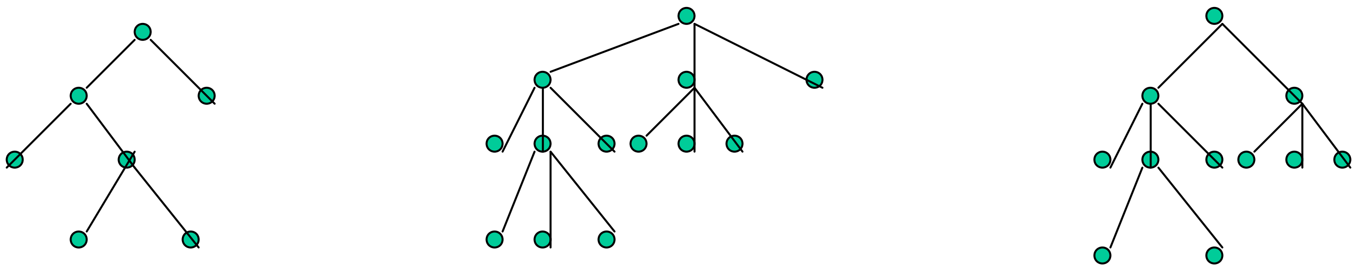
- The ***parent*** of *v* is the unique vertex *u* such that there is a directed edge from *u* to *v*. When *u* is the parent of *v*, *v* is called a ***child*** of *u*.
- Vertices with the same parent are called ***siblings***.
- The ***ancestors*** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself. The ***descendants*** of a vertex *v* are those vertices that have *v* as an ancestor.

m-ary Tree

- A ***leaf*** is a vertex without any child.
- Vertices that have children are called ***internal vertices***.
- **Definition:** A rooted tree is called an *m-ary tree* (*degree of m*) if every internal vertex has no more than *m* children.
- The tree is called a *full m-ary tree* if every internal vertex has exactly *m* children.
- If $m = 2$, it is called a ***binary tree***.

Full *m*-ary trees

- Which of the following trees are ***full m-ary trees*** for some positive integer *m*?



Ordered Rooted Tree

- An ***ordered rooted tree*** is a rooted tree where the children of each internal vertex are ordered.
- In *binary tree*, the first child of an internal vertex with two children is called the left child and the second one is named the right child.
- In *binary tree*, the tree rooted at the left child of a vertex is called the left subtree and the tree rooted at the right child is named the right subtree.

Properties of Tree

- **Theorem:** A tree with n vertices has $(n-1)$ edges.
- **Theorem:** A *full* m-ary tree with i internal vertices contains $n = m \cdot i+1$ vertices.
- **Theorem:** A *full* m-ary tree with

(1) n vertices has $i = (n-1)/m$ internal vertices and $l=[(m-1)n+1]/m$ leaves,

(2) i internal vertices has $n = m \cdot i+1$ vertices and $l = (m-1)i+1$ leaves,

(3) l leaves has $n = (ml-1)/(m-1)$ vertices and $i = (l-1)/(m-1)$ internal vertices

Level and Height

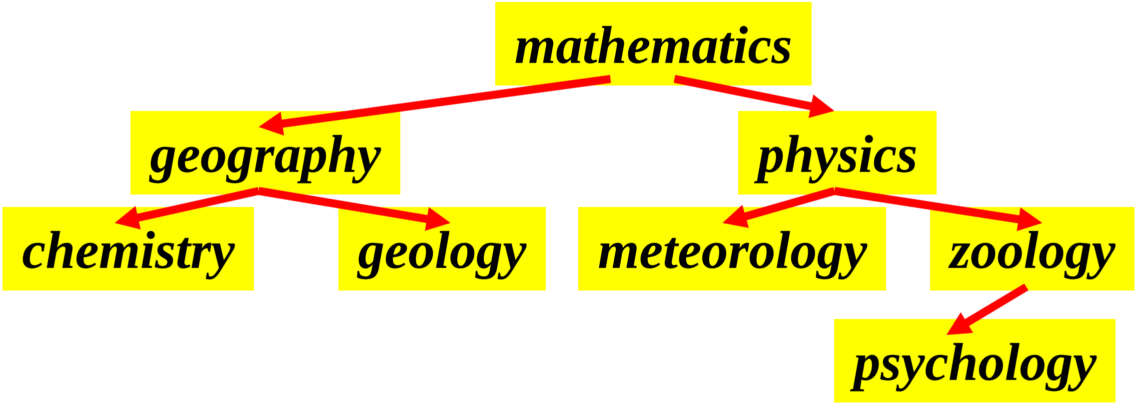
- The ***level*** of a vertex *v* in a rooted tree is the length of the unique path from the root to this vertex. *The level of root is zero.*
- The ***height*** (***depth***) of a rooted tree is the maximum of the levels of all vertices.
- A rooted *m*-ary tree of height *h* is *balanced* if all leaves are at levels *h* or *h*-1.

Level and Height

- **Theorem:** *There are at most m^h leaves in an m-ary tree of height h.*
- **Corollary:** *If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$.*

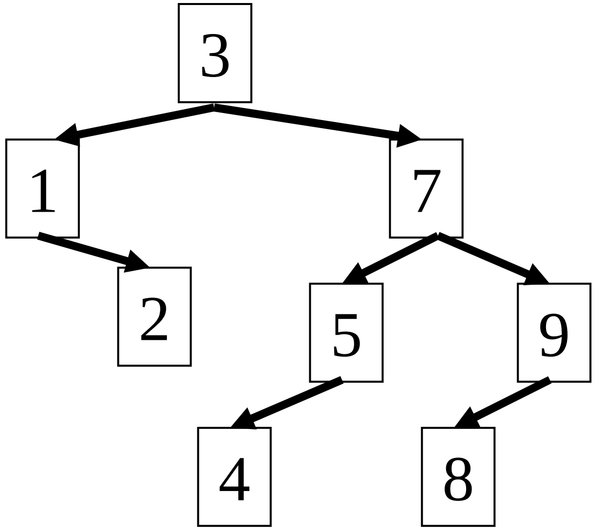
§10.2：Applications of Trees

- Form a *binary search tree* for the words *mathematics, physics, geography, zoology, meteorology, geology, psychology*, and *chemistry* (using alphabetic order).



Binary Sort Trees

- Sort the list: 3,7,5,1,4,2,9,8



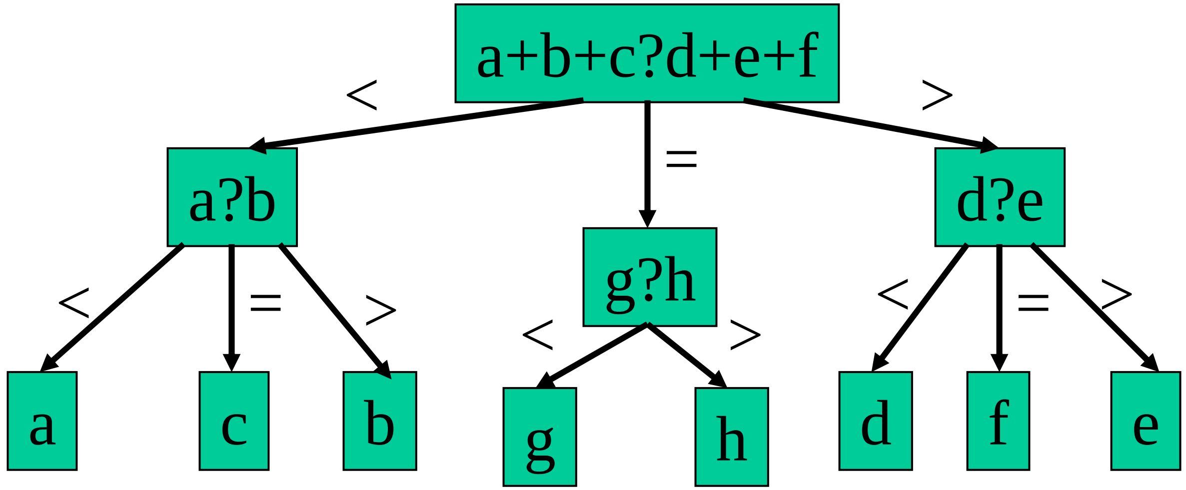
- Read by the order *Left-Root-Right*:1,2,3,4,5,7,8,9

Decision Trees

- Suppose there are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one?

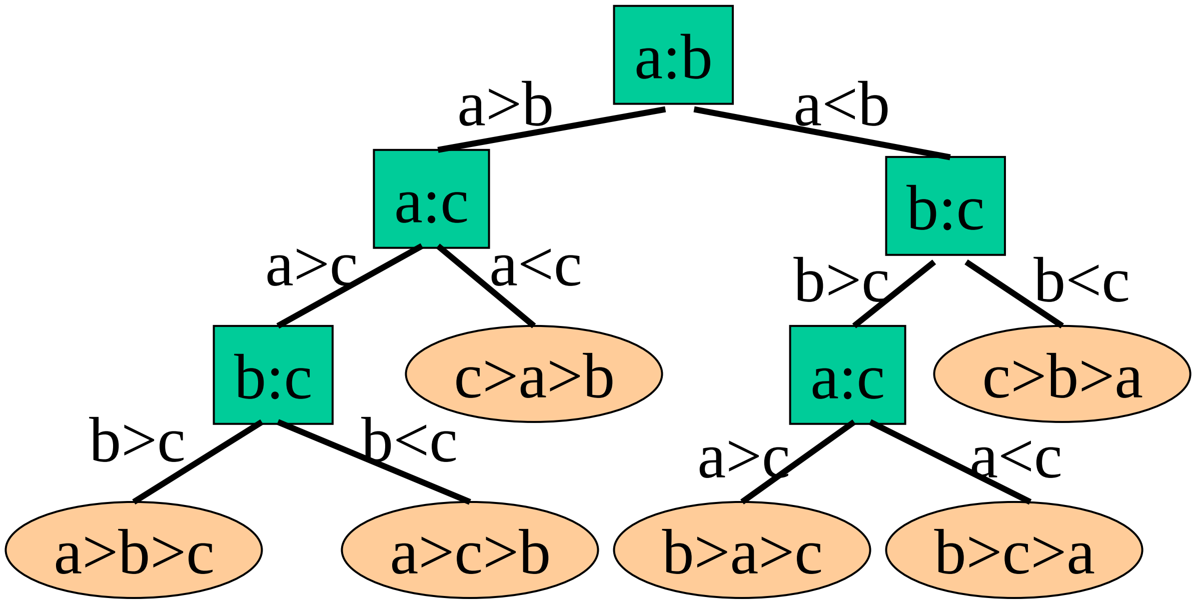
Decision Trees

Let the eight coins be a,b,c,d,e,f,g,h.



Decision Trees

- Form a binary decision tree that orders the distinct elements of the list *a*, *b*, *c*.



Complexity of Sorting Algorithm

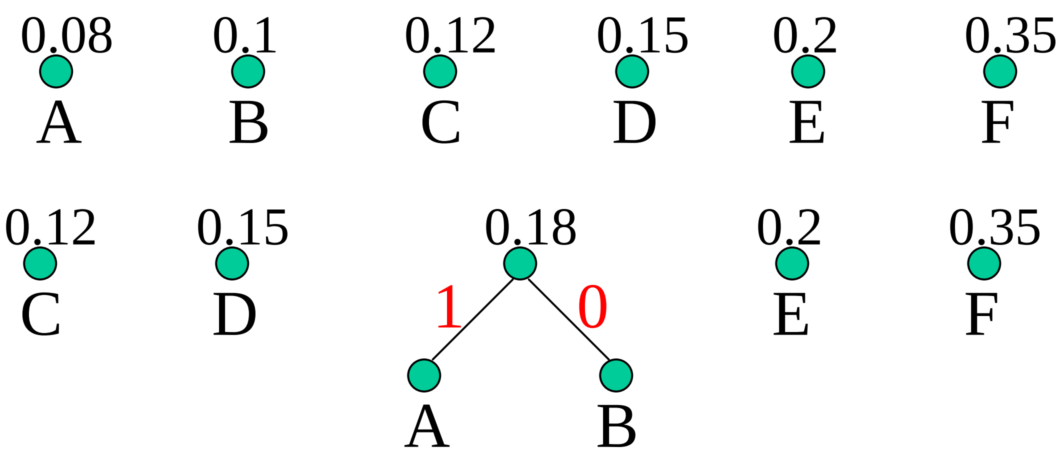
- **Theorem:** A sorting algorithm based on binary comparisons requires at least $\lceil \log_2 n! \rceil$ comparisons.

Complexity of Sorting Algorithm

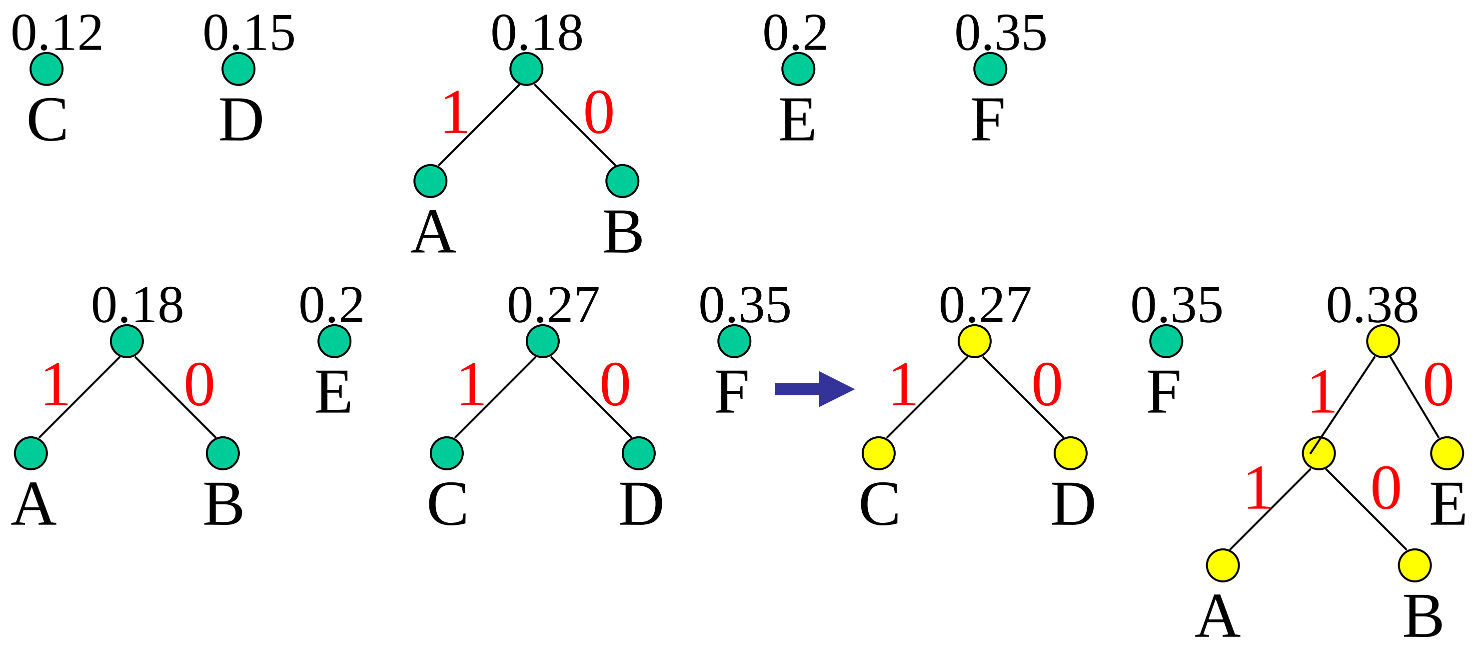
- **Corollary:** The number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n\log n)$.
- **Theorem:** The average number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n\log n)$.

Huffman Code

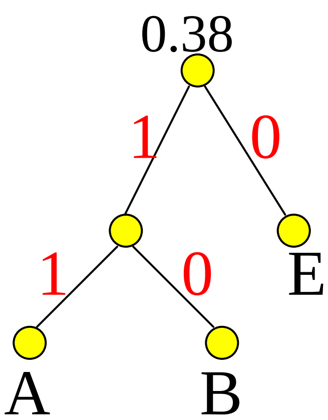
- Use Huffman coding to encode the following symbols A, B, C, D, E and F with the frequencies listed 0.08, 0.1, 0.12, 0.15, 0.2 and 0.35. What is the average number of bits used to encode a character?

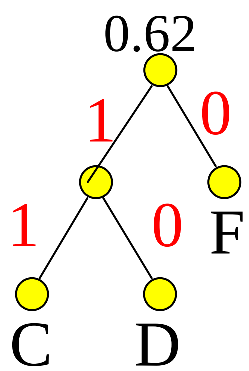


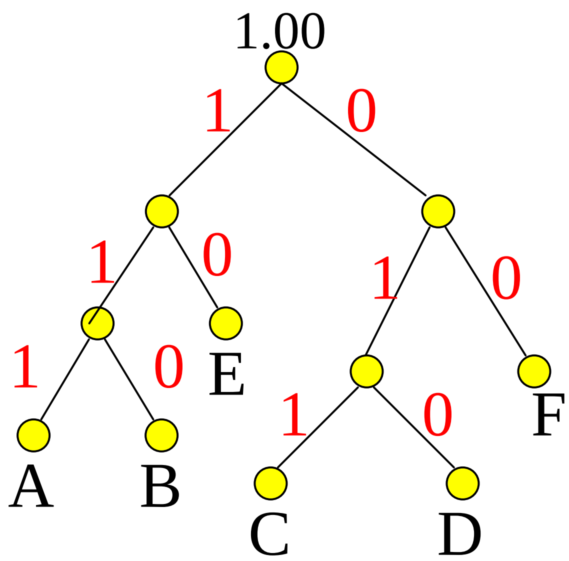
Huffman Code



Huffman Code







A=

B=

C=

D=

E=

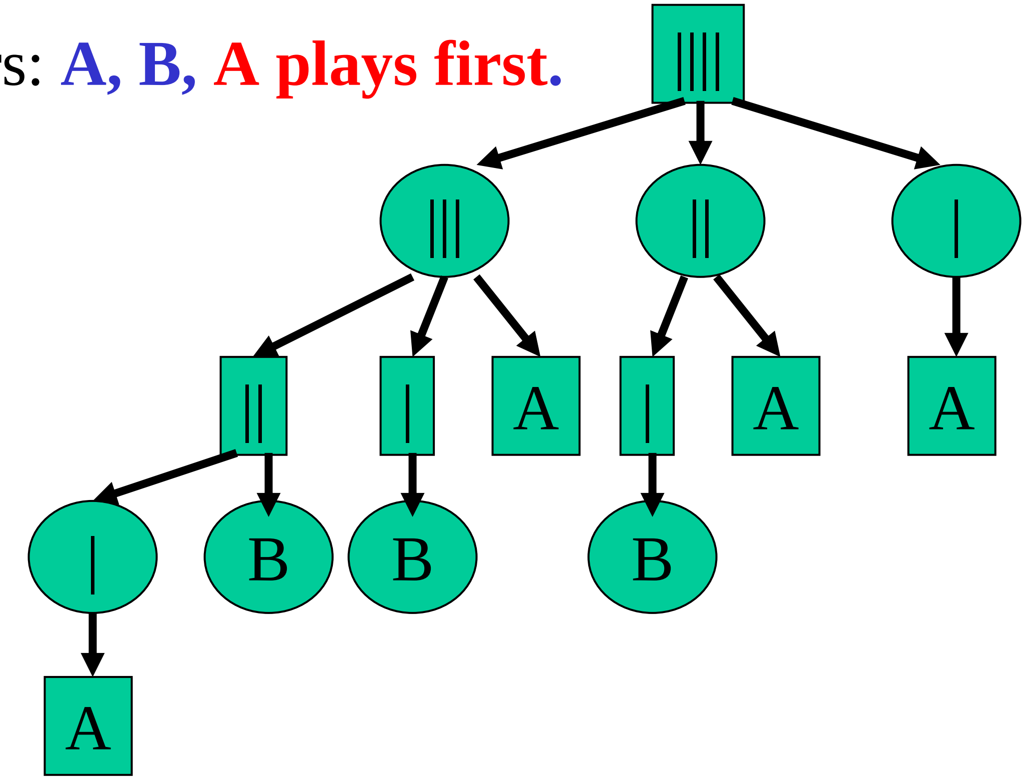
F=

Game Trees

- ***Nim***: At the start there are 4 stones. Two players take turns taking out; a legal playing consists of taking out 1, 2, or 3 stones. The player taking out the last one loses the game.

Game Trees

Two players: **A, B, A plays first.**



Game Trees

- **Definition:** The value of a vertex in a game tree is defined recursively as:

  (1) *the value of a leaf* is the payoff to the first player when the game terminates in the position represented by this leaf.

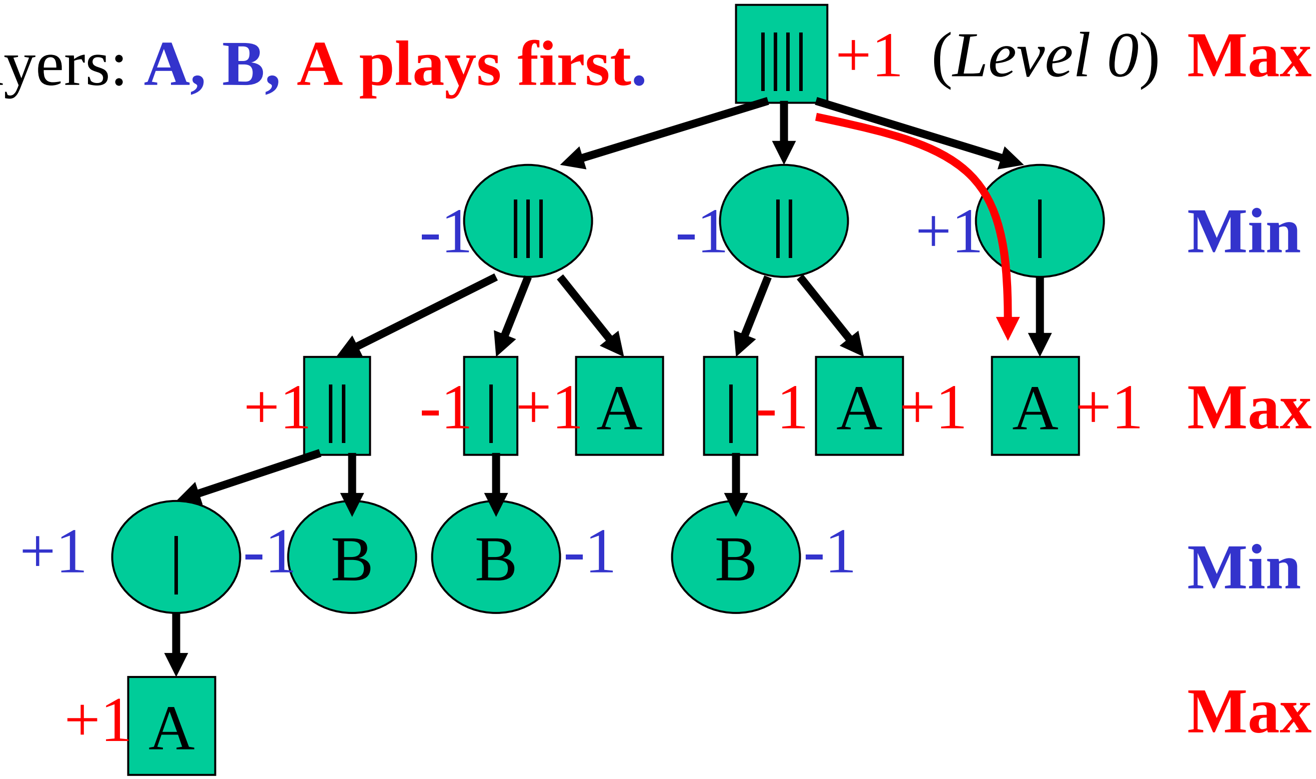
  (2) *the value of an internal vertex* at an *even level* is the *maximum* of the values of its children, and the values at an *odd level* is the *minimum* of the values of its children.

Game Trees

- The strategy where *the first player* moves to a position represented by a child *with maximum* value and *the second player* moves to a position of a child *with minimum* value is called the ***minmax strategy***
- **Theorem:** The value of a vertex of a game tree tells us the payoff to the first player if both players follow the min-max strategy and play starts from the position represented by this vertex.

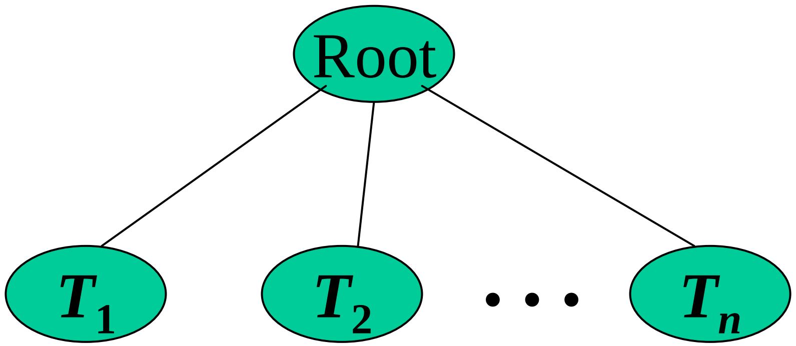
Game Trees

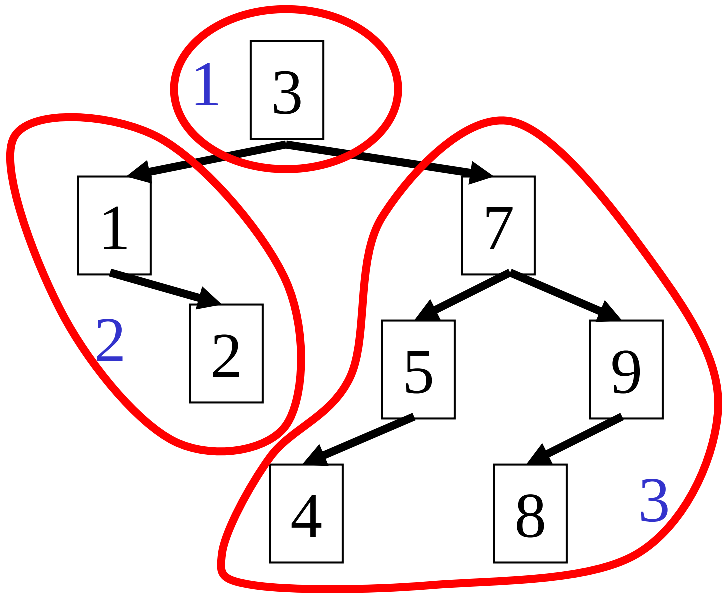
Two players: **A, B, A plays first.**



§10.3：Tree Traversal

Preorder Traversal: Root, T_1, T_2, T_3, …, T_n

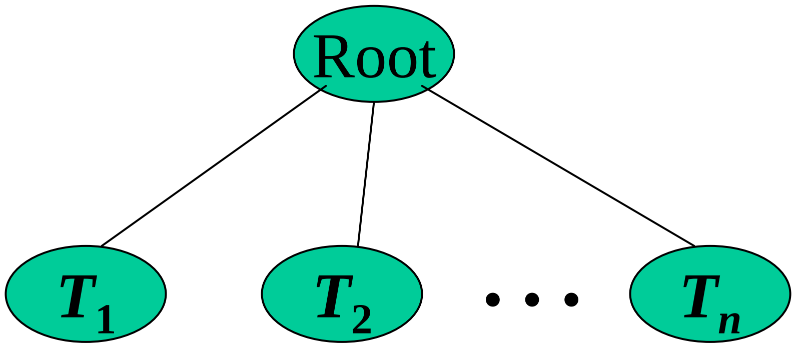


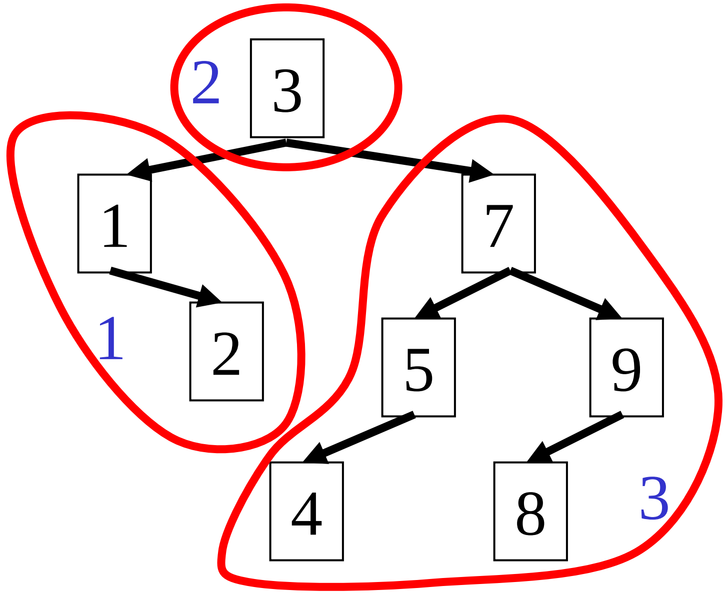


Output: 3,1,2,7,5,4,9,8

Tree Traversal

Inorder Traversal: T_1, Root, T_2, T_3, …, T_n

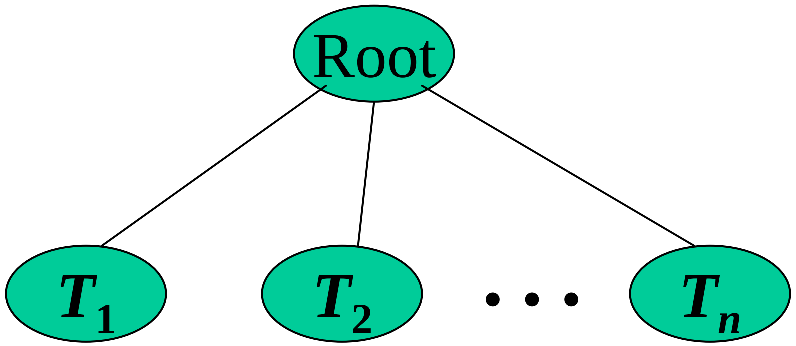


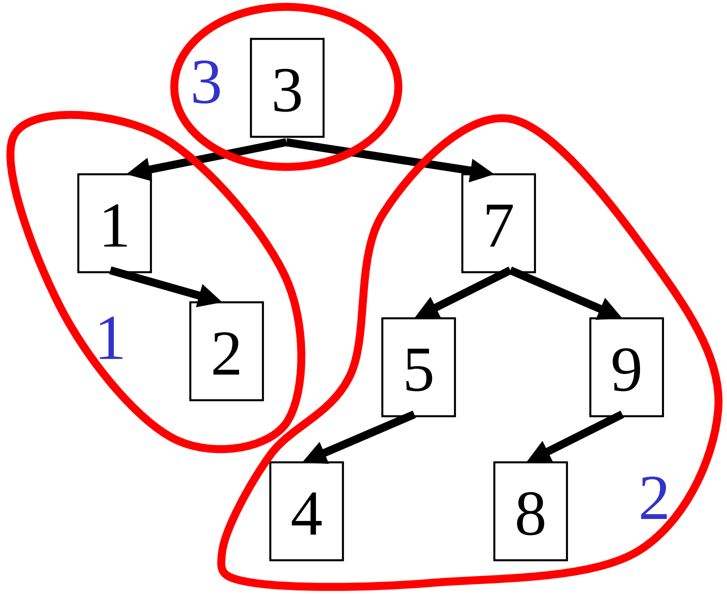


Output: 1,2,3, 4,5,7,8,9

Tree Traversal

Postorder Traversal: T_1, T_2, T_3, …, T_n , Root

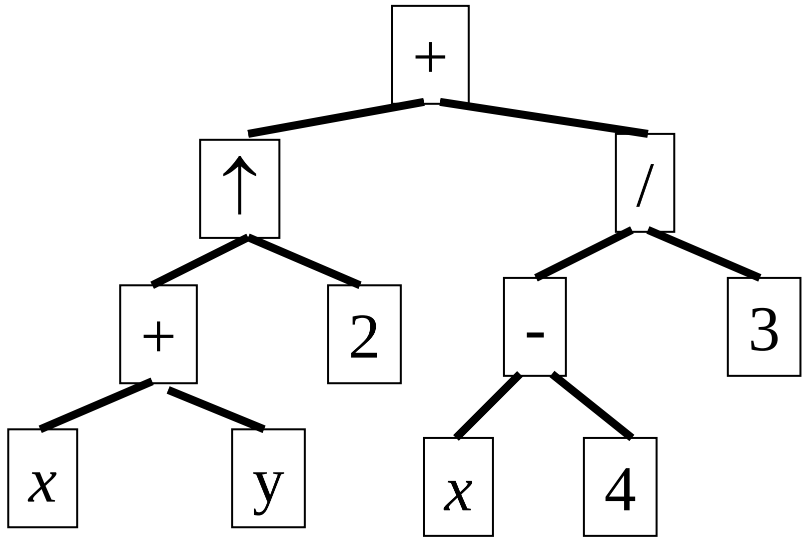




Output: 2,1,4,5,8,9,7,3

Arithmetic Expressions

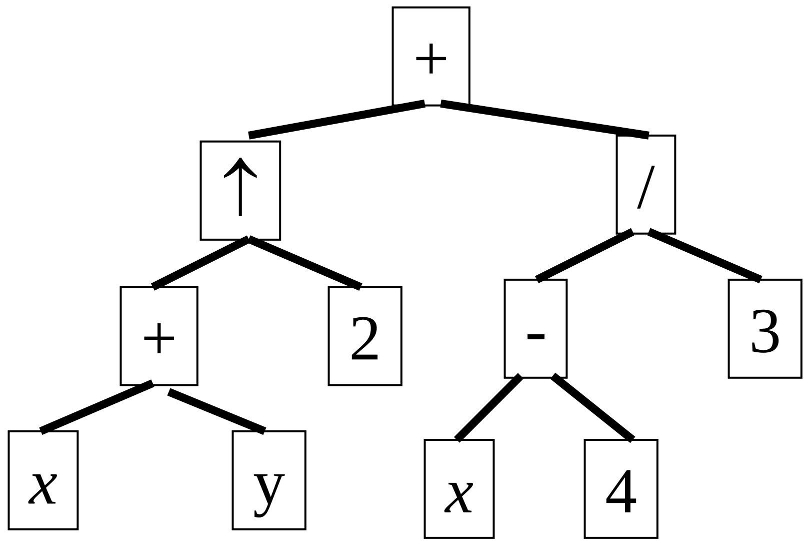
Infix: $((x+y)\uparrow 2)+(x-4)/3$ (Inorder Traversal)



Arithmetic Expressions

Infix: $((x+y)\uparrow 2)+(x-4)/3$

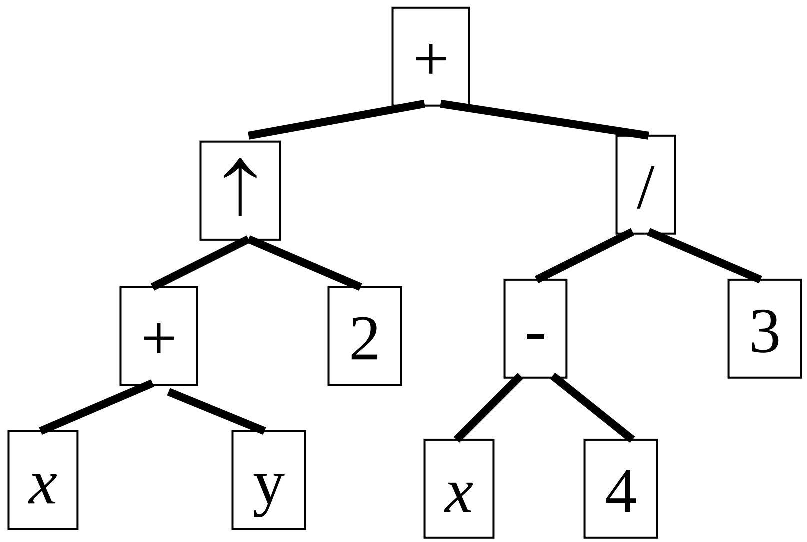
Prefix: (Preorder Traversal)



Arithmetic Expressions

Infix: $((x+y)\uparrow 2)+(x-4)/3$

Postfix: (Postorder Traversal)



Arithmetic Expressions

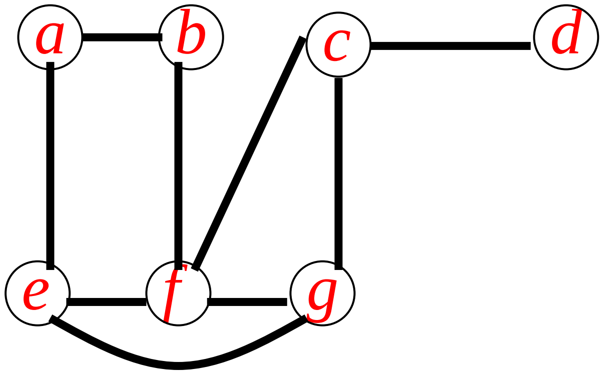
Ex: What is the value of the postfix expression $7\,2\,3 * - 4 \uparrow 9\,3 / +$? *Ans:*

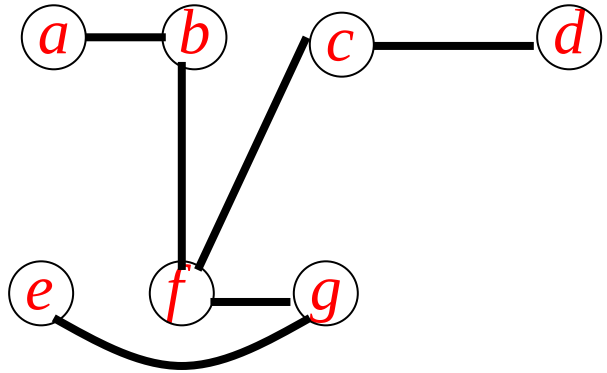
Ex: What is the value of the prefix expression $+ - * 2\,3\,5 / \uparrow 2\,3\,4$? *Ans:*

§10.4：Spanning Tree

Definition: Let *G* be a simple graph. A *spanning tree* of *G* is a subgraph of *G* that is a tree containing every vertex of *G*.

Ex: Find a spanning tree of *G*.

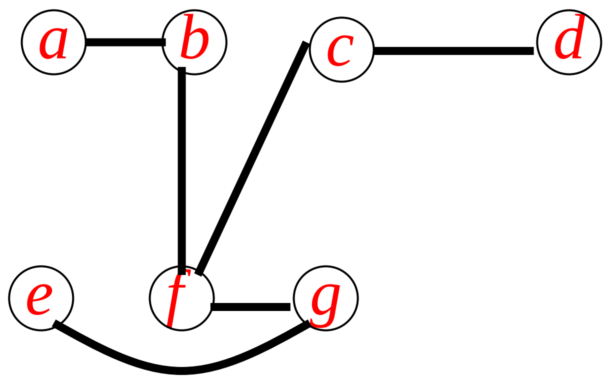




Spanning Tree

Theorem: A simple graph is connected ***iff*** it has a *spanning tree*.

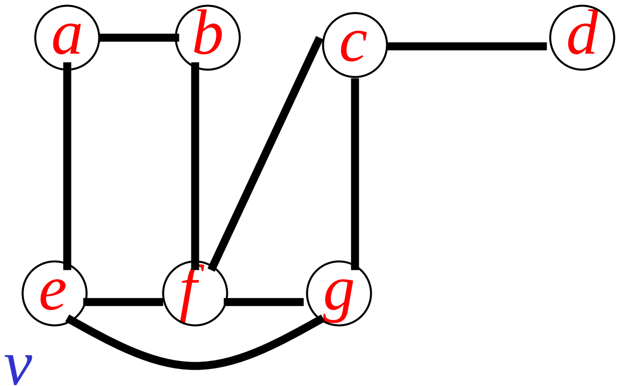
Applications: IP Multicasting/Broadcasting.

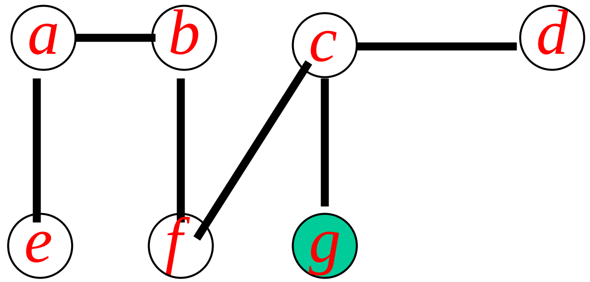


Depth First Search

Ex: Use the DFS to find a *spanning tree*.

Procedure DFS(V,E)
{ T:= tree (v_1)
  visit (v_1) }
Procedure *visit*(v)
for each vertex w adjacent to v
and not yet in T
{ add vertex w and edge (v,w) to T
  visit(w)
}

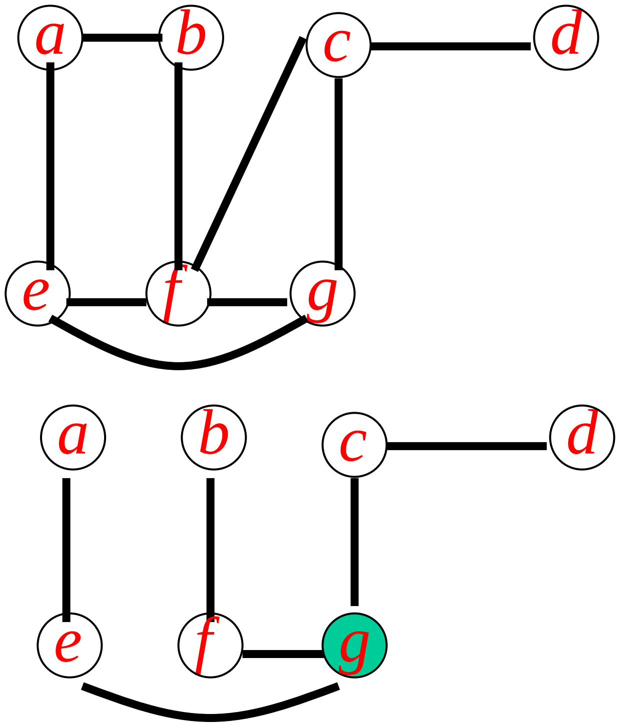




Breadth First Search

Ex: Use the BFS to find a *spanning tree*.

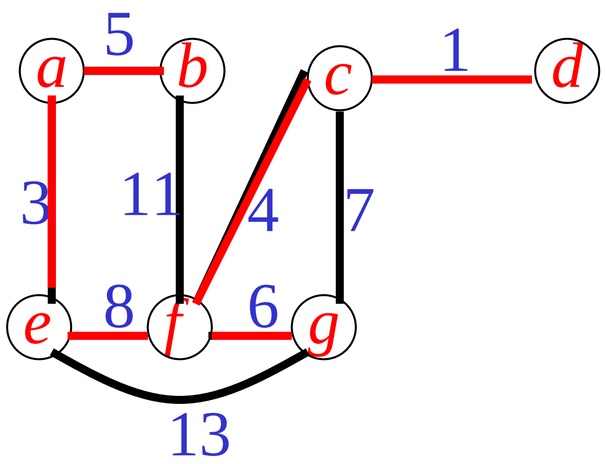
Procedure BFS(V,E)
{ T:= tree (v_1); L:= empty;
put v_1 in the list L
while (L is not empty)
{ remove the first vertex v from L
for each neighbor w of v
if w is not in L and not in T then
{ add w to the end of L
add w and edge (v,w) to T}
} }



L={ ~~g~~, ~~c~~, ~~f~~, ~~e~~ , ~~d~~ , ~~b~~ , ~~a~~ }

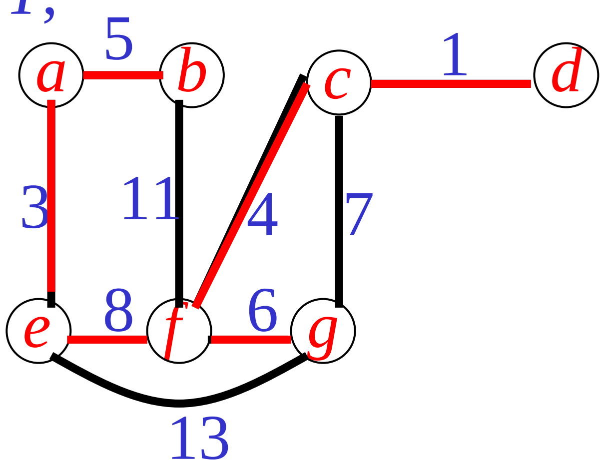
§10.5：Minimum Spanning Tree

Definition: A *minimum spanning tree* in a connected weighted graph is a spanning tree that *has the smallest possible sum* of weights of its edges.



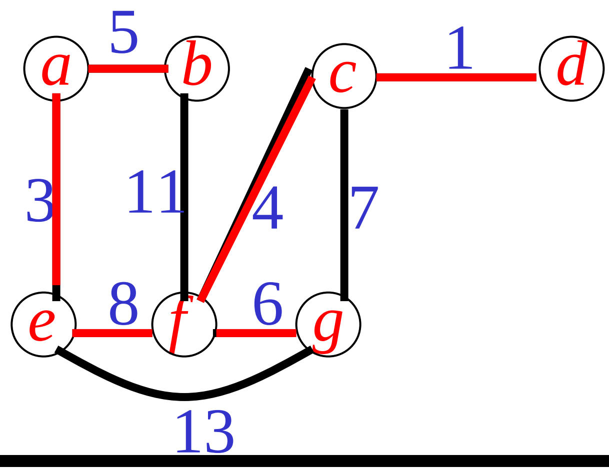
Kruskal's Algorithm

Procedure *Kruskal*(*G* with *n* vertices)
{ *T*:= empty graph;
  for (*i*=1 to *n*-1)
  { *e* = any edge in *G* with smallest weight that does not form
      a simple circuit when added to *T*;
    delete *e* from *G*;
    T = *T* with *e* added;
  }
}



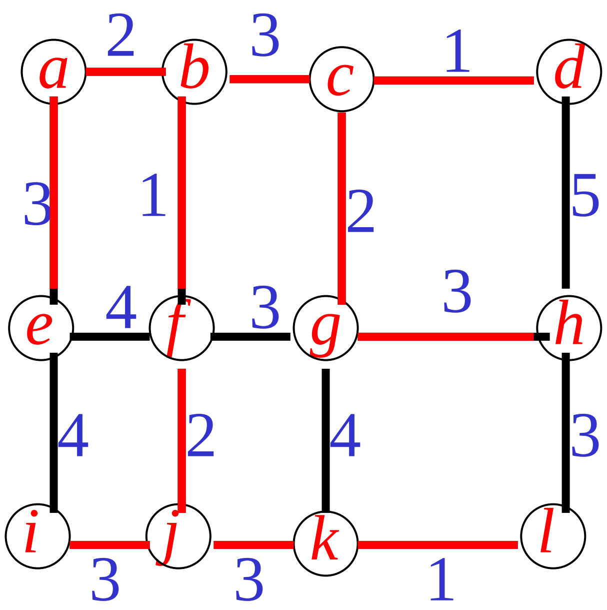
Prim's Algorithm

Procedure *Prim*(*G* with *n* vertices)
{ *T*:= a minimum-weight edge;
for (*i*=1 to *n*-2)
{ *e* = an edge of min weight incident to a vertex in *T* and not forming a simple circuit in *T* if added to *T*
T = *T* with *e* added
}
}



Example

Kruskal's Algorithm



Example

Prim's Algorithm