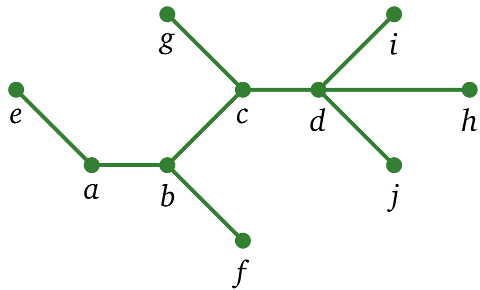
Trees

Tree

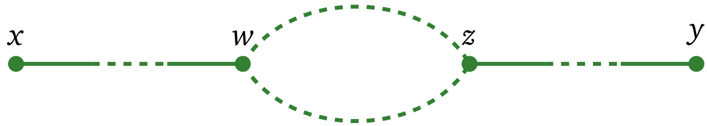


Def. A *tree* is a connected undirected graph with no simple cycles.

It does not need to have a root node, or directed edges. So, it is not hierarchical, and there are no parent/child nodes.

Uniqueness of a simple path

Theorem. There is *exactly one simple path* between each pair of vertices in a tree.



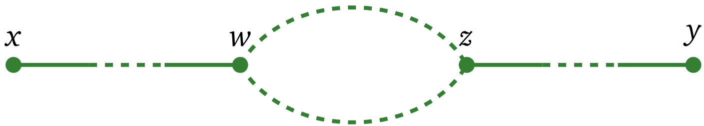
Assume that there are two different simple paths from x to y.

Can we prove that it cannot happen in a tree?

Uniqueness of a simple path

Theorem. There is *exactly one simple path* between each pair of vertices in a tree.

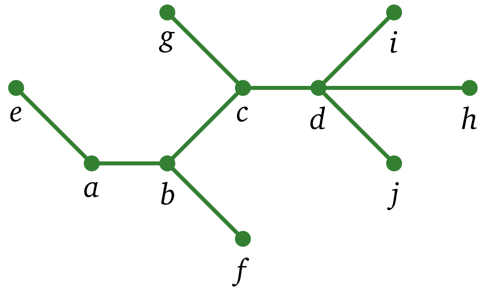
Proof. Trees are connected graphs, so, there is a simple path between each pair of vertices. Are they unique?



Assume that for vertices x and y there are two simple paths between them. There must exist a vertex w, where the paths separate, and a vertex z, where they meet again for the first time. But they form a simple cycle $w \to z \to w$!

This is a contradiction, b/c trees don't have simple cycles, therefore, our assumption was incorrect, and for any x and y, there is no two different simple paths in a tree. □

Properties of trees



Lemma. Removing one edge from the edge set of a tree gives a graph with two connected components, each of which is a tree.

Properties of trees

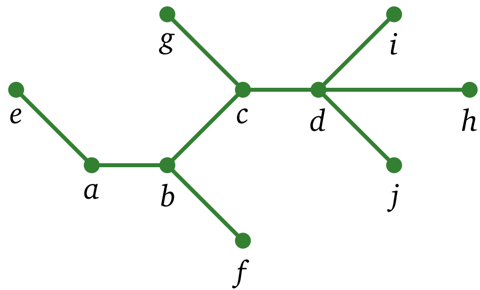
Lemma. Removing one edge from the edge set of a tree gives a graph with two connected components, each of which is a tree.

Proof. By the previous theorem, a removed edge makes the tree disconnected.

When edge (u, v) is removed, each vertex is either connected to u, or to v. So, there are two connected components.

And each connected component is a tree, because we could not introduce cycles by removing the edge (u, v). □

Properties of trees



Theorem. A tree with $n > 0$ vertices has $n - 1$ edges.

Properties of trees

Theorem. A tree with $n > 0$ vertices has $n - 1$ edges.

Proof by (strong) induction using the previous lemma.

Base case: A tree with one vertex has 0 edges, fine.

Inductive step: IH: Assume that for all trees with $0 < m < n$ vertices, there are $m - 1$ edges.

Given a tree with n vertices, remove one edge, getting two connected components C_1 and C_2 with k and $n-k$ vertices respectively. By the IH, C_1 and C_2 have $k-1$ and $n-k-1$ edges. Thus the original tree contained $(k-1) + (n-k-1) + 1 = n - 1$ edges.

Corollary. A finite tree with more than one vertex has at least one vertex of degree 1.

Therefore, trees have leaves (terminal vertices).

Euler formula for planar graphs

Theorem (Euler formula). Let G be a connected planar simple graph with e edges and v vertices. Let r be the number of faces in a planar representation of G. Then

$$v - e + f = 2.$$

Euler formula for planar graphs

Proof. If there is more than one face, there is an edge separating two faces. Remove it, merging the faces. f and e are decreased by one, but $v - e + f$ remains the same.

Continue, until there is only one face. Then, since there is only one face, there is no simple cycles, so this is a tree.

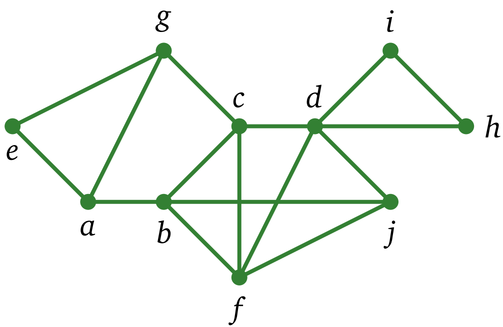
In the tree, there is at least one vertex with degree 1 (a leaf). Remove it from the tree. e and v are both decreased by 1. Therefore, $v - e + f$ remains the same.

Continue, until there is only one vertex, $v = 1$, $e = 0$, $f = 1$:

$$v - e + f = 2$$

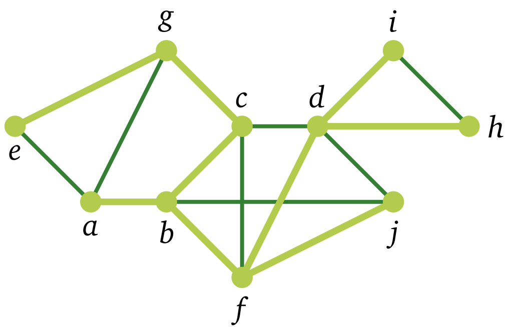
In the original graph, $v - e + f$ was the same. □

Spanning trees



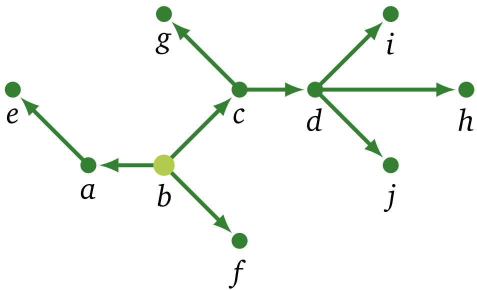
Def. Let G be a simple graph. A *spanning tree* of G is a subgraph of G that is a tree containing every vertex of G.

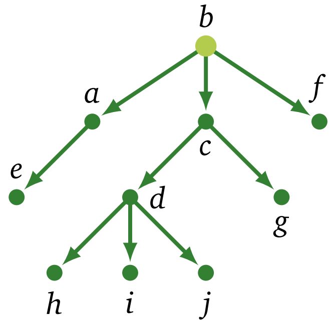
Spanning trees



Def. 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.

Rooted tree



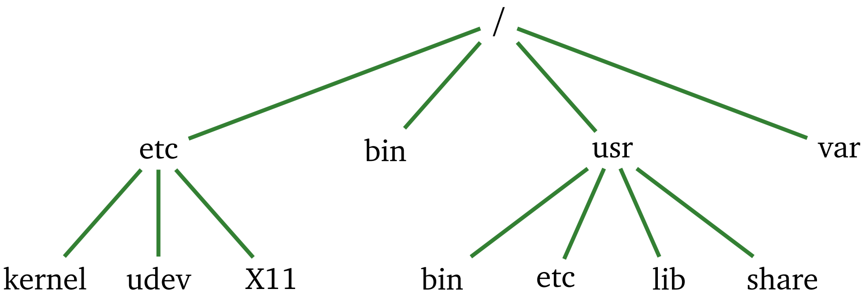


Def. A *rooted tree* is a tree in which one vertex has been designated as *the root* and every edge is directed away from the root.

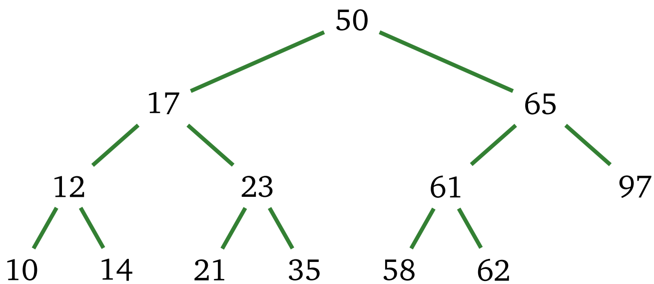
A node has: the parent node, children, siblings, ancestors, descendants.

A vertex is called a *leaf* if it has no children, otherwise it is called an *internal vertex*.

Examples: File system



Examples: Binary search tree

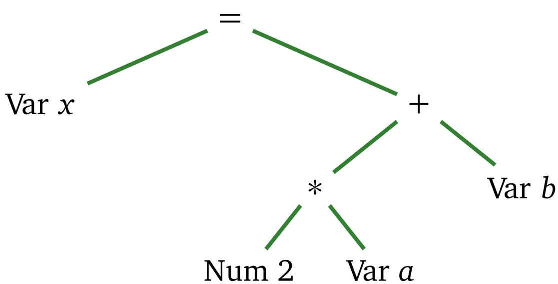


Def. A rooted tree is called a *binary* tree if if every internal vertex has no more than 2 children.

(It's called "full", if every internal node has exactly 2 children)

Examples: Programs are trees

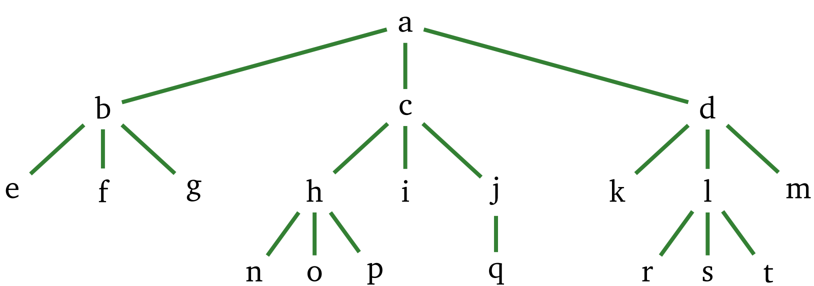
$$x = 2 * a + b;$$



Rooted trees

Def. A rooted tree is called an *m-ary* tree if if every internal vertex has no more than m children. (when $m = 2$, the tree is called binary).

Def. An m-ary tree is celled *full* if every internal vertex has exactly m children.



Rooted trees

Theorem. A full m-ary tree with i internal vertices contains

$$n = m \cdot i + 1 \quad \text{vertices.}$$

In particular, in binary trees, $n = 2i + 1$.

Rooted trees

Theorem. A full m-ary tree with i internal vertices contains

$$n = m \cdot i + 1 \quad \text{vertices.}$$

In particular, in binary trees, $n = 2i + 1$.

Proof. Every vertex, except the root, is the child of an internal vertex. Because each of the i internal vertices has m children, there are mi vertices in the tree other than the root. Therefore, the tree contains $n = mi + 1$ vertices. □

Rooted trees

Theorem. A full m-ary tree with i internal vertices contains

$$n = m \cdot i + 1 \quad \text{vertices.}$$

In particular, in binary trees, $n = 2i + 1$.

How can we use the theorem?

Note that the number of leaves is $l = n - i$.

Question: What is the number of internal nodes and the number of leaves in a full binary tree with n vertices?

Rooted trees

Question: What is the number of internal nodes and the number of leaves in a full binary tree with n vertices?

Solution: $n = 2i + 1$, so the number of internal nodes is

$$i = (n-1)/2,$$

and the number of leaves is

$$l = n - i = n - (n-1)/2 = (n+1)/2.$$

Thus in a large full binary tree, the number of internal nodes is almost the same as the number of leaves.

Rooted trees

Question: Find the least $n > 0$ such that there exist two trees:

a full 19-ary tree with n vertices, and
a full 32-ary tree with the same number of vertices.

Use the same theorem:

Theorem. A full m-ary tree with i internal vertices contains

$$n = m \cdot i + 1 \quad \text{vertices.}$$

Encode a message

Consider a situation when we are sending messages like this:

$$THISISAMESSAGETESTTESTTEST$$

Conventional **char** type takes 8 bits. So, every letter is encoded as a bit-string of length 8.

We would like to find a way to efficiently encode the letters of the English alphabet with shorter bit-strings.

Possible solution

We encode 26 letters as bit-strings of length 5. Since $2^5 = 32 > 26$, this is enough to represent each letter.

$A = 00000$, $B = 00001$, $C = 00010$, $\ldots$

Can we do better if we know how frequently each letter occures in the message?
Consider using bit strings of different lengths to encode letters.

Frequncies of letters

Letter	Frequency
A	8.167%
B	1.492%
C	2.782%
D	4.253%
E	12.702%
F	2.228%
G	2.015%
H	6.094%
...	

We can try

$$E = 0$$
$$A = 1$$
$$H = 00$$
$$D = 01$$
$$C = 10$$
$$F = 11$$
$$G = 000$$
$$B = 001$$

Try to decode:

011000

Prefix codes

How to resolve the problem? Make a code so that there is no such collisions.

Encode letters so that the bit string for a letter never occurs as the prefix (first part) of the bit string for another letter.

$$
\begin{aligned}
E &= 0 \\
A &= 10 \\
H &= 110 \\
D &= 1110 \\
C &= 11110 \\
&\ldots
\end{aligned}
$$

Codes with this property are called *prefix codes*.

Prefix codes

$E = 0$

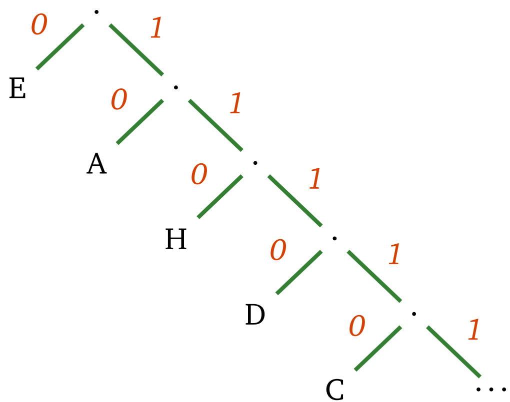
$A = 10$

$H = 110$

$D = 1110$

$C = 11110$

$\ldots$



Try to decode:

$$1100101110$$

Huffman coding

Input:

Letter	Frequency
A	8.167%
B	1.492%
C	2.782%
D	4.253%
E	12.702%
F	2.228%
G	2.015%
H	6.094%
…	

Huffman coding is an algorithm that takes as input the frequencies of symbols in a string and produces as output a prefix code that encodes the string using the *fewest possible bits*.

Output: prefix code.

Given symbols and their frequencies, our goal is to construct a rooted binary tree where the symbols are the labels of the leaves.

Huffman coding

Letter	Frequency
A	8%
B	10%
C	12%
D	15%
E	20%
F	35%

Huffman coding

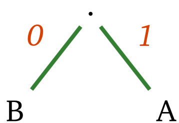
Initial state. Start with disjoint trees:

(8%)	(10%)	(12%)	(15%)	(20%)	(35%)
A	B	C	D	E	F

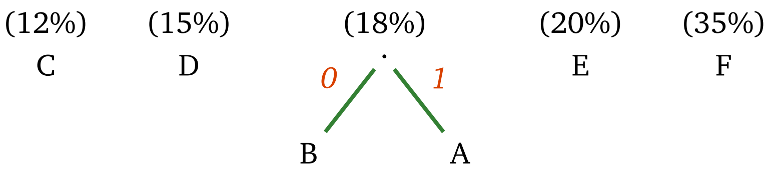
Step 1. Take two trees with the least frequencies: A and B here, and combine them in a single tree: The tree that has the higher frequency becomes the left branch.

(12%) (15%) (18%) (20%) (35%)

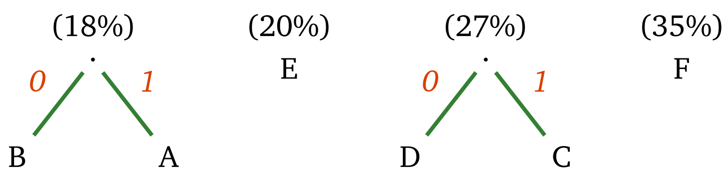
C D E F



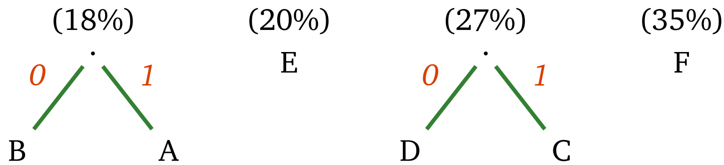
Huffman coding



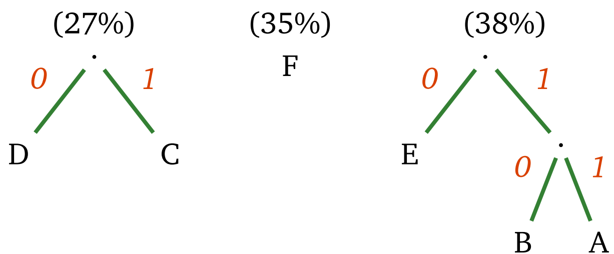
Step 2.



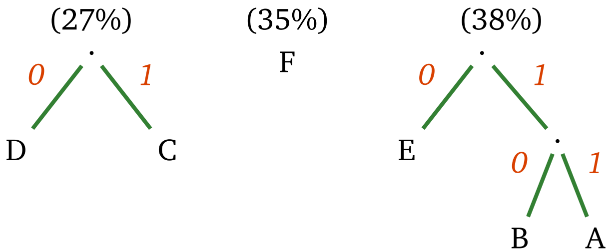
Huffman coding



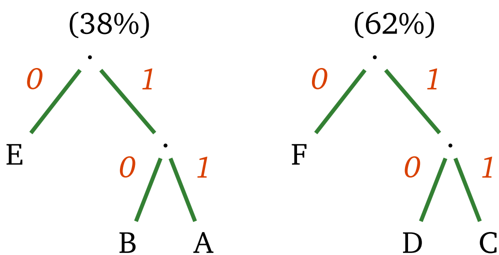
Step 3.



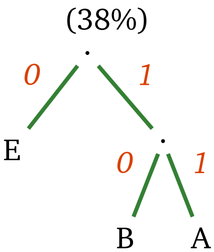
Huffman coding

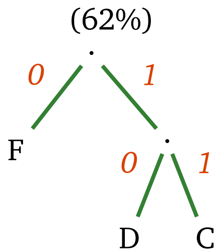


Step 4.

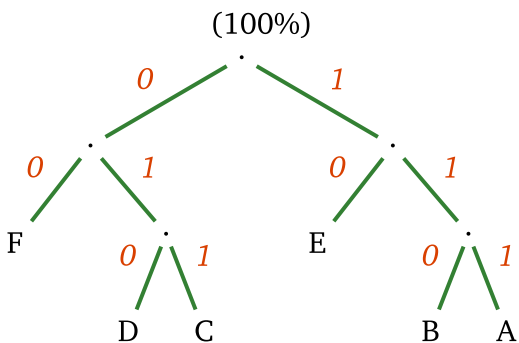


Huffman coding

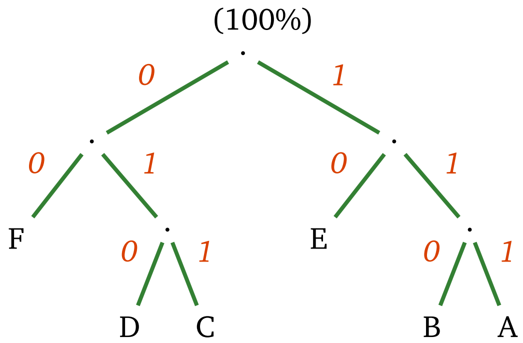




Step 5.



Huffman coding



Letter	String	Freq.
A	111	8%
B	110	10%
C	011	12%
D	010	15%
E	10	20%
F	00	35%

What is the average number of bits used to encode a character?

$$3 \cdot (0.08 + 0.10 + 0.12 + 0.15) + 2 \cdot (0.20 + 0.35) = 2.45.$$

Huffman coding is optimal code in the sense that no binary prefix code for these symbols can encode these symbols using fewer bits.