Trees

Trees form the most widely used subclasses of graphs. In CS, we make extensive use of trees. Trees are useful in organizing and relating data in databases, file systems and other applications.

Formal Definition:
A (free) tree $T$ is a simple graph satisfying the following:
- If $v$ and $w$ are vertices in $T$, there is a unique simple path from $v$ to $w$.
- A rooted tree is a tree in which a particular vertex is designated the root.

In contrast to natural trees, which have their roots at the bottom, in graph theory rooted trees are typically drawn with their roots at the top.
To draw a tree, first, we place the root at the top ($v_1$). Under the root and on the same level, we place vertices $v_2, v_3$ … that can be reached from the root $v_1$ by a simple path of length 1. Under each of these vertices, we place vertices that can be reached from root $v_1$ on a simple path of length 2. We continue this way…

Level of a vertex $v$: the length of the simple path from the root to $v$.
The height of a rooted tree is the maximum level.
A rooted tree is often used to specify hierarchical relationships.
When a tree is used this way, if vertex $a$ is on a level one less the level of vertex $b$ and $a$ and $b$ is adjacent then $a$ is “just above” $b$ and a logical relationship exists between $a$ and $b$:
Examples of tree like structures are computer file systems.
Folders or directories contain other folders or files. Files are leaves in that they do not have any descendents.

Theorem 1:
An undirected graph is a tree if and only of there is a unique simple path between any 2 of its vertices.

**Tree terminology**
Let T be a tree with root v0. Suppose that x, y and z are vertices in T and that (v0, v1… vn) is a simple path in T, then:

a) \( v_{n-1} \) is the parent of \( v_n \)
b) \( v_0,\ldots,v_{n-1} \) are the ancestors of \( v_n \)
c) \( v_n \) is a child of \( v_{n-1} \)
d) if x is an ancestor of y, y is a descendant of x
e) if x and y are children of z, x and y are siblings
f) If x has no children, it is a terminal vertex or a leaf.
g) If x is not a terminal vertex, x is a internal (or branch) vertex.
h) The subtree of T rooted at x is the graph with vertex set V and edge set E where V is x together with the descendents of x and
i) \( E= \{ e \mid e \text{ is an edge on a simple path from } x \text{ to some vertex in } V \} \)

Let T be a graph with n vertices. The following are equivalent:

a) T is a tree.
b) T is connected and acyclic
c) T is connected and has n-1 edges
d) T is acyclic and has n-1 edges

A rooted tree is called an m-ary tree if every internal vertex has no more than m children.
The tree is called full m-ary tree if every internal vertex has exactly m children.
An m-ary tree with m=2 is called a binary tree.
An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered, that is are shown from left to right. In an ordered binary tree, if an internal vertex has 2 children, the first child is called the left child and the second child is called the right child.

A full m-ary tree with
I. \(N\) vertices has \(i=(N-1)/m\) internal vertices and
   \(l=[(m-1)N+1]/m\) leaves
II. \(i\) internal vertices has \(N=mi+1\) vertices and \(l=(m+1)i+1\) leaves
III. \(l\) leaves has \(N=(mL-1)/(m-1)\) vertices and \(i=(l-1)/(m-1)\) internal vertices

Example:
Suppose that someone starts a chain letter. Each person who receives the letter is asked to send it to 4 other people. Some people do this, but others do not send any letter. How many people have seen the letter, including the first person, if no one receives more than 1 letter and if the chain letter ends after there have been 100 people who read it and did not send it out? How many people sent out the letter?

Solution:
4-ary tree
People who saw the letter = nodes
People who did not send the letter= leaves (100)
People who sent the letter: internal nodes (n= (4.100-1)/(4-1) =133 nodes
i= (100-1)/(4-1) = 33

Huffman Codes:
The most common way to represent characters internally is by using fixed-length bit strings. ASCII represents each character by a string of seven bits. Huffman Codes, which represent characters by variable-length bit strings, provides alternative to ASCII. Consider using bit strings of different lengths to encode letters. The idea is to use short bit strings to represent the most frequently used characters and to use longer bit strings to represent less frequent characters. This way we can save memory and reduce the transmittal time.

When letters are encoded using varying number of bits, some method has to be used to determine where the bits of each character start and end. For instance, if e was encoded with 0, a with 1 and t with 01, then the bit string 0101 could correspond to eat, tea, tt, eaea

One way to ensure that no bit string corresponds to more than 1 sequence of letters is to encode letters so that the bit string of a letter never occurs as the first part of the bit string of another letter. Codes with this property are called prefix codes. Encoding e as 0, a as 10 and t as 11 is a prefix code. 10110 can only be interpreted as ate

A prefix code can be represented using a binary tree, where:
- The characters are the labels on the leaves of the tree.
- The edges of the tree are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned 1.
- The bit string used to encode a character is the sequence of labels of the edges in the unique path from the root to the leaf, which has the character as its label.

(0,e),(10, a),(110, t),(1110, n),(1111,s)
11111011100

A Huffman code is most easily defined by a rooted tree. To decode a bit string, we begin at the root and move down the tree until a
character is encountered. The bit, 0 or 1, tells us whether to move right or left.

**Constructing an Optimal Huffman Code:**
This algorithm constructs an optimal Huffman code from a table giving the frequency of occurrences of the characters to be represented. The output is a rooted tree with the vertices at the lowest levels labeled with the frequencies and with the edges labeled with bits.
The coding tree is obtained by replacing each frequency with the character having that frequency.

**Input:** A sequence of n symbols with frequencies w, n > 1;
**Output:** A rooted tree that defines an optimal Huffman code

**Procedure Huffman** (F:= forest of n rooted trees, each consisting of the single vertex aᵢ and assigned weight wᵢ)
**while** F is not a tree
  **begin**
  Replace the rooted tree T and T’ of least weights from F with w(T)≥w(T’) with a tree having a new root that has T as its left subtree and T’ as its right subtree.
  Label the new edge to T with 0 and the new edge to T’ with 1.
  Assign w(T) +w(T’) as the weight of the new tree.
  **end**
[The Huffman code for the symbol aᵢ is the concatenation of labels of the edges in the unique path from the root to the vertex aᵢ]
The algorithm begins by repeatedly replacing the smallest two frequencies with their sum until a tree is obtained.