1 Introduction to trees

A tree T is made up of a set of nodes endowed with parent-child relationship with following properties:

- If T is non-empty, it has a special node called the root that has no parent
- Every node v of T other than the root has a unique parent
- Following the parent relation always leads to the root (i.e., the parent-child relation does not have "cycles")

1.1 Terminology

- Root: node without parent.
- Internal node: node with at least one child.
- External/leaf node: node without children.
- Ancestors: parent, grandparent, great-grandparent, etc.
- Descendants: child, grandchild, great-grandchild, etc.
- Two nodes with the same parent are siblings.
- Depth of a node: number of ancestors not including itself.
- Level: set of nodes with given depth.
- Height of a tree: maximum depth.
- Subtree: tree made up of some node and its descendants.
- Edge: a pair of nodes (u, v) such that one is the parent of the other.
- Path: sequence of nodes such that consecutive nodes in the sequence have an edge

Tree facts

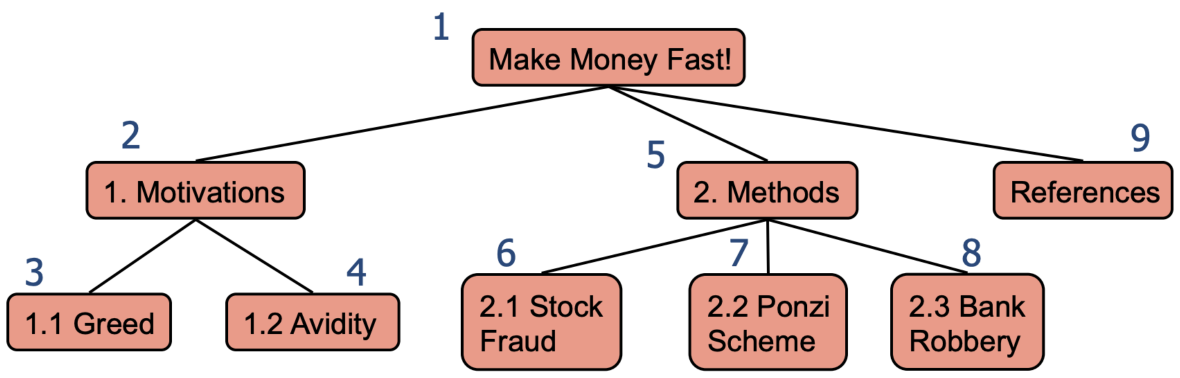
- If node X is an ancestor of node Y, then Y is a descendant of X.
- Ancestor/descendant relations are transitive.
- Every node is a descendant of the root.
- There may be nodes where neither is an ancestor of the other.
- Every pair of nodes has at least one common ancestor.
- The lowest common ancestor (LCA) of x and y is a node z such that z is the ancestor of x and y and no descendant of z has that property.
- In an ordered tree there is a prescribed order for each node's children.

2 Traversal

2.1 Preorder Traversal

To do a preorder traversal starting at a given node, we visit the node before visiting its descendants If tree is ordered visit the child subtrees in the prescribed order. Visit does some work on the node such as print node data, aggregate node data or modify node data. The example shows a pre_order traversal called at root.

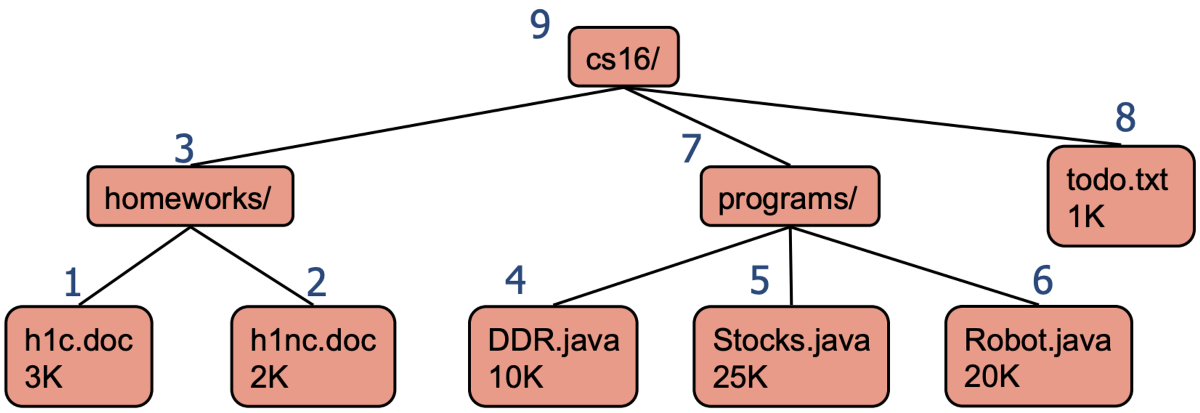
```
1. def pre_order(v):
2.     visit(v)
3.     for each child w of v do
4.         pre_order(w)
```



2.2 Postorder Traversal

To do a postorder traversal starting at a given node, we visit the node after its descendants If tree is ordered visit the child subtrees in the prescribed order.

```
1. def pre_order(v):
2.     for each child w of v do
3.         pre_order(w)
4.     visit(v)
```



3 Binary Trees

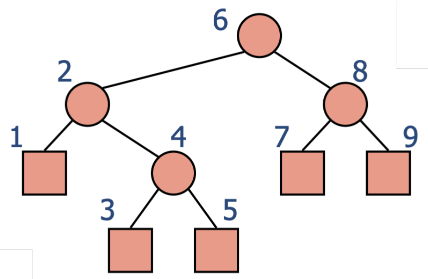
A binary tree is an ordered tree with the following properties: Each internal node has at most two children. Each child node is labeled as a left child or a right child. Child ordering is left followed by right. The right/left subtree is the subtree root at the right/left child. We say the tree is proper if every internal node has two children.

```
1. def is_external(v):                                  # Tests if v is a leaf
2.     return v.left = null and v.right = null
```

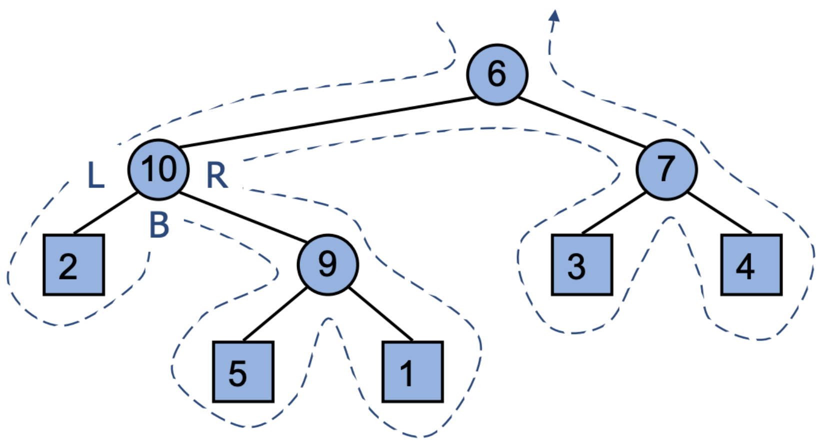
3.1 Inorder Traversal

To do an inorder traversal starting at a given node, the node is visited after its left subtree but before its right subtree.

```
1. def in_order(v):
2.     if v.left != null then
3.         in_order(v.left)
4.     visit(v)
5.     if v.right != null then
6.         in_order(v.right)
```



4 Euler Tour Traversal & Some code



6,10,2,2,2,10,9,5,5,5,9,1,1,1,9,10,6,7,3,3,3,7,4,4,4,7,6

Preorder (first visit): 6, 10, 2, 9, 5, 1, 7, 3, 4

Inorder (second visit): 2, 10, 5, 9, 1, 6, 3, 7, 4

Postorder (third visit): 2, 5, 1, 9, 10, 3, 4, 7, 6

```
1. def height(v):                                  # compute height of subtree at v
2.     if v.parent = null then                             # root's depth is 0
3.         return 0
4.     else
5.         return depth(v.parent) + 1
```

```
1. def depth(v):                                   # compute height of subtree at v
2.     if v.isExternal() then                            # a leave's height is 0
3.         return 0
4.     else
5.         h ← 0
6.         for each child w of v do
7.             h ← max(h, height(w))
8.         return h + 1
```