day 17

Trees

* A tree data structure has a root branches, leaves.

* All the children of one node are independent of the children of another node.

* Each leaf node is unique. (Example : file manager)

Node:

* A node is a fundamental part of a tree. It can have a name, which we call the 'key'.

* A node may also have additional information. We call this additional information the 'payload'

Edges :

* An edges connects two nodes to show that there is a relationship between them.
* Everynode (Enpect the root) is Connected by enactly one incoming edge from another node
* Each node may have several outgoing edges.

Root :

Root of the tree is a Only node in the tree that has no incoming edges.

Path :

A path is an ordered list of nodes that are Connected by edges.

Child :

the set of nodes that have incoming edges from the same node to are said to be the children of that node.

Parent:

A node is the parent of all nodes it connects to with outgoing edges.

Sibilings:

Nodes in the tree that are children of the same parent are said to be sibilings.

Subtree:

A subtree is a set of nodes and edges comprised of a parent and all descendants of that parent.

leaf:

leaf is a node that has no children

level:

the level of a node is the number of edges on the path from the root node to n.

Height:

Height of a tree is equal to the maximum level of any node in the tree.

Application of tree:

* tree is a hierarchial (non-linear) data structure.
* One reason to use trees might be because you want to store information that naturally forms hierachy.
* If we organize keys in form of a tree with some oridinary, we can search for a given key in moderate time (quicker from linked list)
* we can insert / delete key in moderate time.

Binary tree implementation

```
def binarytree(r):
    return [r, [], []]

def insertleft(root, new_branch):
    t = root.pop(1)
    if len(t) > 1:
        root.insert(1, [new_branch, t, []])
    else:
        root.insert(1, [new_branch, [], []])
    return root

def insertright(root, new_branch):
    t = root.pop(2)
    if len(t) > 1:
        root.insert(2, [new_branch, t, []])
    else:
        root.insert(2, [new_branch, [], []])
return root.
```

```
def getroot(root):
    return root[0]

def setroot(root, new-val):
    root[0] = new-val

def getleftChild(root):
    return root[1]

def getrightChild(root):
    return root[2]
```

input:

```
r = binarytree(3)
insertleft(r, 4)
insert left(r, 5)
insert right(r, 6)
insert right(r, 7)
print (r)
```

Output:

```
[3, [5, [4, [], []], []], [7, [], [6, [], []]]
```