MLDS 422 - Intro to Python
Lab 5

Sungsoo Lim

October 25, 2023

NORTHWESTERN
UNIVERSITY

Today's Lab Materials

- Binary Tree and Hash Table
  - Details
  - Implementation

Binary Tree

- A tree data structure for hierchical data

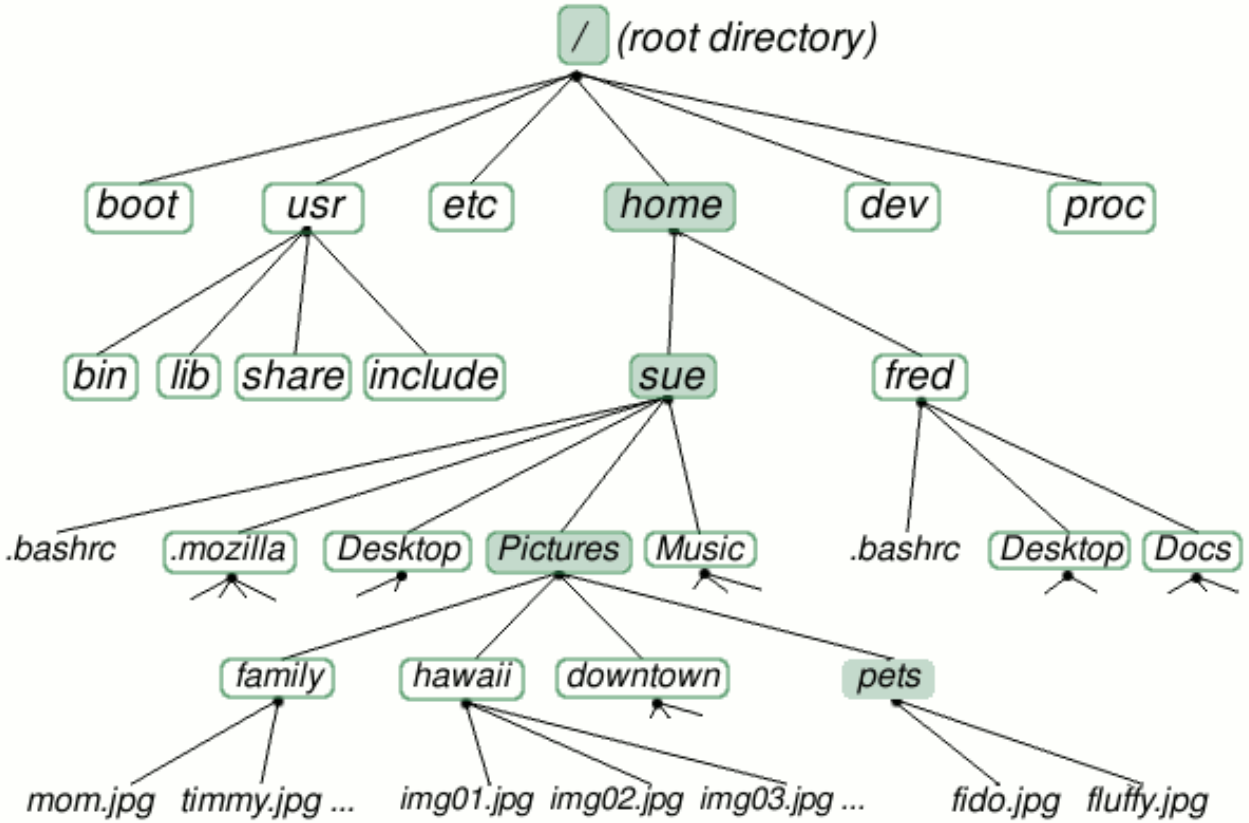


Figure: https://astikanand.github.io/techblogs/data-structures/binary-tree

Binary Tree - Terminalogy

- **node**: any element of a tree
- **root**: Topmost node
- **children**: nodes directly under a node
- **parent**: node directly above the node or nodes
- **leaves**: nodes with no children
- **edge** (link from node to node), **path** (list of links from one node to another)
- **level** (distance from the root), **height** (distance from the root to leaf)

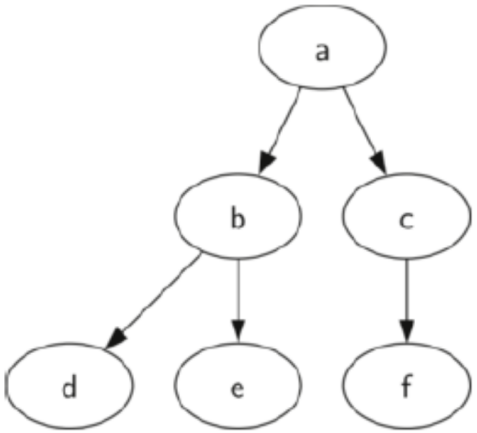
Binary Tree - Methods

- **BinaryTree()** - create a new instance
- **get_left_child()** - return the binary tree corresponding to the left child of the current node
- **get_right_child()** - return the binary tree corresponding to the right child of the current node
- **set_root_val(val)** - set root node as val
- **get_root_val()** - return the root node value
- **insert_left(val)** - create a new binary tree and install it as the left child of the current node
- **insert_right(val)** - create a new binary tree and install it as the right child of the current node

Implementations - List of Lists

- Functional programming
- my_tree= [root, [left tree], [right tree]]

```
my_tree = ['a', #root
     ['b', #left subtree
      ['d' [], []],
      ['e' [], []] ],
     ['c', #right subtree
      ['f' [], []],
      [] ]
    ]
```



Implementations - Class BinaryTree

- Attributes: root value, left subtree, right subtree

- Method definitions

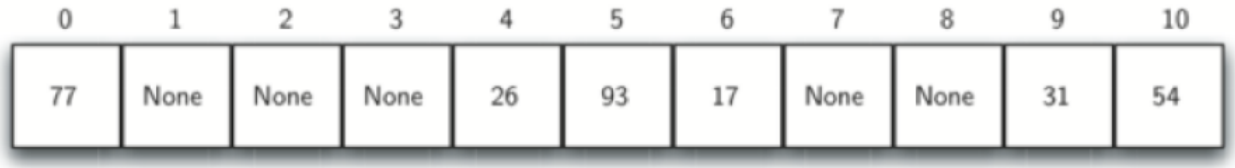
Hash Table

- A collection of items that are stored in a way that can be fast to find ($\mathcal{O}(1)$)
- Label each position of the hash table by an integer
- Implement a hash table by a list of some size

Hash Function

- A mapping between an item and its integer label

- Takes the item as input and returns an integer in the range between 0 and $m - 1$, where m is the predefined size of list



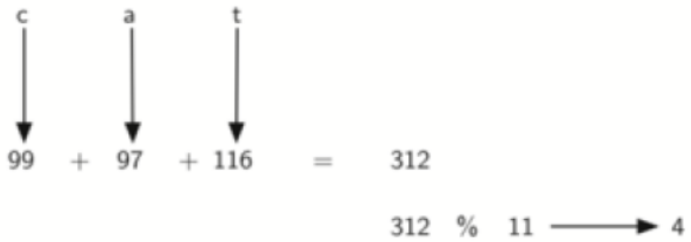
Hash Function

- **perfect hash function**: takes each unique item to unique label
- Goal:
  - Minimize the number of collisions (number of items in each unique label)
  - Easy to compute
  - Evenly distributes the items in the hash table
  - Larger hash table (list) than items - memory issues

Hash Function

- hash functions for **character-based items** such as strings.
  - Map string to numbers using ***Ordinal*** value.
  - Example: "cat"

```
>>> ord('c')
99
>>> ord('a')
97
>>> ord('t')
116
```




```
def hash(a_string, table_size):
    sum = 0
    for pos in range(len(a_string)):
        sum = sum + ord(a_string[pos])

    return sum % table_size
```

Collision Resolution

- Allow each slot to hold a reference to a chain of items