CSCA48 Assignment 2

Summer 2018

Due: July 28th 2018, 5 pm

Description:

In this assignment you will practice working with search trees and developing recursive functions.

Multiway Search Tree

You have seen and worked with search trees in which each internal node has at most two children. They were called binary search trees (BST). BSTs are a special kind of Multiway Search Trees (MST). An MST is a search tree in which internal nodes not only may have more than two children but also can store more than one key in each node. In this assignment you are going to implement a Multiway Search Tree that has the following properties:

A. An entry (key, value) is stored in the tree.
B. Values are stored in a list in each node.
C. Keys are stored in a list in each node.
D. Each internal node has at most four children.
E. In each node at most three ordered keys are stored.
F. Duplicate entries are allowed.

MSTs can be used to access data quickly. You probably remember that for every ADT that you implement, you should develop three methods, one to search for the given data, one to insert and one to remove data. In this assignment, we only ask you to implement search and insert.

Searching an MST:

Suppose you have K_1, K_2 and K_3 as the keys stored at node V. To find a data, whose key is K:

- If $K \in [K_1, K_2, K_3]$, then the node is found.
- If $K<K_1$, then follow the first child.
- If $K_1 < K < K_2$ then follow the second child.
- If $K_2 < K < K_3$ then follow the third child.
- Otherwise follow the fourth child.

Inserting into an MST:

To insert data, not only you should follow the search tree's rule to make sure that data is inserted in a right place, you should take care of the number of the keys that you store in each node.

To insert a new item *(k,v)*, with key *k,* perform a search for *k:*

1- If *k* was found, add *v* to the list that represent k's values. See part a of the following figure, where ('F', 'Frank') is inserted. Please Note that ['Frank', 'Fred'] is not the child of the node rather it is inside the node. It's been drawn like this due to space limitation of the page.

2- If *k* was not found your search is terminated at node `Z`,

   a. Insert `k` into `z`, and add the value `v` to the list. See part b of the following figure, where ('S', 'Sheila') is inserted. Again due to space limitation, I had to draw it like this. ['Shiela'] is stored inside the node, it is not the child of the node.

   b. [**Bonus**] However, `z` may already have 3 keys stored in it. This means that the rule for this MST is violated (rule E), if you add one more key. In this case we say an **overflow** at node `z` has happened. In case an overflow happens, you need to **split** the nodes**.** Let c_1, c_2, c_3, c_4, c_5 be the children of node z, and let k_1, k_2, k_3, k_4 be the keys stored at `z` after insertion, where overflow has happened. To split:

      i. Replace node `z` with two nodes `v` and `w`.
      ii. `v` contains k_1, k_2 with children c_1, c_2, c_3.
      iii. `w` contains k_4 with children c_4, c_5 as its first and second children.
      iv. k_3 is added to the parent of `z`, which is now the parent of `v` and `w`. if `z` was the root of the tree, a new node is created as the root and k_3 is added to this node.
      v. if overflow happened in the parent, repeat from step i.

      See the figure below, which for simplicity, only the keys are presented. Suppose that the current tree looks like c, before a node such as ('u', 'Ursula') is added. Since this node creates an overflow, a split such as what is seen in d is required, which also requires u to be added to the root. That will create an overflow in the root. Therefore, the root should be split and a new root is created. See part e of the figure.

Your job for this assignment is to write a code that implements this MST, which contains people's contact that is given in the starter code. To ensure that you focus on MST implementation, a contact simply is an object that contains a name and a phone number. Please do not touch `contact` code and use it as it is. The key for a contact is the first letter of the name, while `contact` itself is the value.

Methods to implement for MSTNode:

1- `add(contact)` **:** adds a contact to the node assuming that no overflow happens.

2- `__eq__(mst_node)` **:** returns true if two nodes are the same. It means they contain the same keys and values, points to the same children and have the same parent.

Methods to implement for MST:

1- `search(k, v)`: search for an entry whose key is k and the value is v. This method returns a Boolean. You've probably noticed that we don't need to pass the key, as it can be retrieved from the value. We'd like to keep it as it is to be consistent with other search trees' methods.

2- `find(node, k)`: looks for key k and return the pointer to the node that holds this key, if it is found. Otherwise None is returned. This is a recursive method.

3- `insert(k, v)`: insert an entry with the key and value equal to k and v to the tree according to the given insertion rule. Implementing 2-b is optional. You will receive extra 20% If part 2-b is correctly implemented.

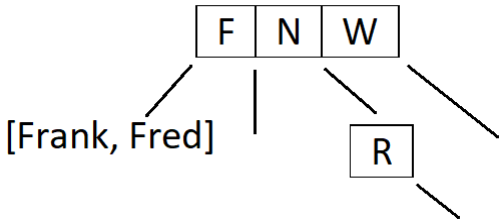
4- `__eq__(mst_tree)`: returns true if two trees are the same. This must be implemented recursively

5- `BFS()`: returns a string representing the MST tree resulted from traversing the tree by BFS. Here is an example of how the output of this method might look like, when it is called on the MST shown in figure e. It is obvious that in figure e, the contact information cannot be seen due to space limitation. We are not interested to present the phone number in the output. Therefore, contact name is sufficient. No space between any character is required.

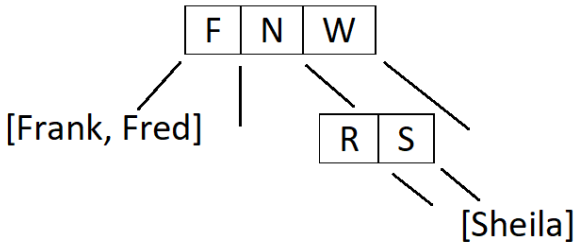
```
Level_0=[T:[Tanya]]
Level_1=[F:[Frank,Fred]-N:[Neda]-W:[Will,Wendy,Will]]
Level_2=[R:[Randy,Ray,Roy]-S[Sheila]-U[Ursula]]
```

Extra implementation information.

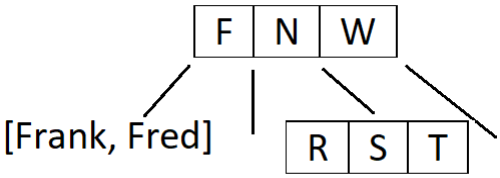
- You are allowed to add any helper method that you require in any of the classes that is given in the starter code.
- No *print* statements and any other *import* statements other than what is inside the starter code should exist in your code.
- All the attributes in all the classes are **private** variables.
- Name of the methods should be exactly the same as what is mentioned in this description.
- We are looking for some recursion methods therefore where applicable and where stated explicitly, recursion should be used.
- When comparing the keys, case should not matter (i.e. "j" is the same as "J")
- Do not modify the code that is already present in the starter code.
- We assume a third party always validate our input. So you don't need to worry about this, instead focus on finding the right algorithm that solves this problem.
- Submit a file called MST.py.
- Do not forget to fill in the declaration part of the this code.



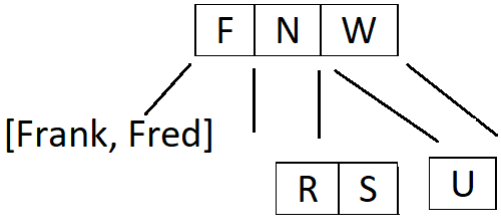
a) Insert ('F', 'Frank'), assuming that Fred was added before.



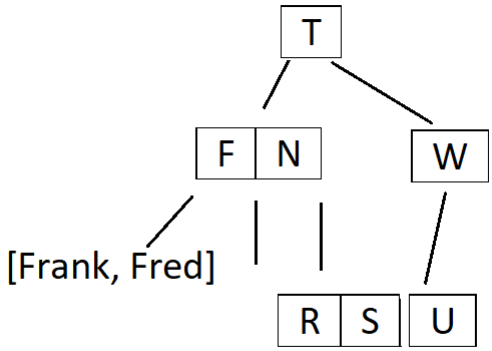
b) Insert ('S', 'Sheila') to the tree shown in part a



c) Original tree



d) Split is required if u is added. T should be added to the parent (rule 2.b)



e) The final tree