

Programming assignment 4: Map ADT with binary search tree

Using recursion is not a requirement, but recommended in traversing trees.

90%

Implement the class **BSTMap** with a binary search tree data structure.

The class must fully implement the **Map** ADT, including the following operations:

- **insert(key, data) 20%**
 - Adds this value pair to the collection
 - If equal key is already in the collection, **raise ItemExistsException()**
- **update(key, data) 10%**
 - Sets the data value of the value pair with equal **key** to **data**
 - If equal key is not in the collection, **raise NotFoundException()**
- **find(key) 10%**
 - Returns the **data** value of the value pair with equal **key**
 - If equal key is not in the collection, **raise NotFoundException()**
- **contains(key) 5%**
 - Returns **True** if equal **key** is found in the collection, otherwise **False**
- **remove(key) 20%**
 - Removes the value pair with equal **key** from the collection
 - If equal key is not in the collection, **raise NotFoundException()**
- **__setitem__(self, key, data) 5%**
 - Override to allow this syntax:
 - `some_bst_map[key] = data`
 - If equal **key** is already in the collection, update its **data** value
 - Otherwise add the value pair to the collection
- **__getitem__(self, key) 5%**
 - Override to allow this syntax:
 - `my_data = some_bst_map[key]`
 - Returns the **data** value of the value pair with equal **key**
 - If equal key is not in the collection, **raise NotFoundException()**
- **__len__(self) 5%**
 - Override to allow this syntax:
 - `length_of_structure = len(some_bst_map)`
 - Returns the number of items in the entire data structure
- **__str__(self) 10%**
 - Returns a string with the items **ordered** by **key** and separated by a single space.
 - Each item is printed on the following format: {value_of_key:value_of_data}
 - ```
m[5] = "five"
m[3] = "three"
m[7] = "seven"
print("output: " + str(m))
```
    - **output: {3:three} {5:five} {7:seven}**

**5% Bonus for 100% correct output in all test cases.**

**5% Bonus for a correct solution that uses no unnecessary repetition of code.**

*Note that in some cases similar code can be necessary, but minimize it as much as possible.*

**10%**

*This assignment part is not directly connected with the previous part (**BSTMap**).*

*This class should not be used in any way in that implementation. As a part of testing that part, this key class could be used as a key type sent into that implementation, but as assignments they are completely separate.*

Implement the class **MyComparableKey**, constructed with an integer value and a string value. Implement the following operations:

- **`__init__(self, int_value, string_value)`**
  - A constructor that takes an integer value and a string value
- **`__lt__(self, other)`**
  - Compares two instances of MyComparableKey and returns **True** if the value of self is lower, otherwise **False**.
  - A key value is considered lower if the **integer** value is lower.
    - In case of *equal integers* the order of the **strings** is used.
  - *It is OK to use built in operators for base types in this implementation.*