MODULE 5 PROJECT: BINARY TREES

Learning Objectives:

CLO 1. Identify fundamental data structures in computer science and their use in real-life applications.

CLO 3. Develop problem solving skills by implementing data structures.

CLO 4. Compare advantages and disadvantages of different data structure implementations.

CLO 5. Design and analyze composite data structures.

Data Structure Implementation

1. Implement `BinaryTree`.

   Files to Modify: BinaryTree.py (new template)

   Directions: Finish the implementation of the following methods:

   - `depth(u)` - returns the depth of node `u`
   - `_height(u)` - recursive, helper method that returns the height of node `u`.
   - `_size(u)` - recursive, helper method that returns the size of the subtree rooted at node `u`.
   - `bf_order()` - returns the list of nodes that results from traversing the binary tree using breadth-first traversal.
   - `_in_order(u)` - recursive, helper method returns the list of nodes that results from traversing the binary tree rooted at node `u` using in-order traversal (left, parent, right).
   - `_post_order(u)` - recursive, helper method returns the list of nodes that results from traversing the binary tree rooted at node `u` using post-order traversal (left, right, parent).
   - `_pre_order(u)` - recursive, helper method returns the list of nodes that results from traversing the binary tree rooted at node `u` using pre-order traversal (parent, left, right).

2. Implement `BinarySearchTree` which is a specialized `BinaryTree` data structure. `BinarySearchTree` will inherit all methods from the `BinaryTree` class.

   Files to Modify: BinarySearchTree.py (new template)

   Directions: Finish the implementation of the following methods:

   - `add(key, value)` - adds a node with given `key` and `value` to the tree, if `key` does not already exist in the tree (i.e. no duplicate keys are allowed). Returns `True` if the key-value pair was added successfully, `False` otherwise.
   - `find(key)` - returns the value corresponding to the given `key` if the key exists in the tree. Otherwise, it returns `None`.
   - `remove(key)` - removes the node with given `key` and returns the value corresponding to the key, if it exists in the tree. Raises a `ValueError` if the key is not found in the tree.
   - `_find_eq(key)` - helper method; returns the node with given `key` if the key exists in the tree, `None` otherwise.

- `_find_last(key)` - helper method; returns the node with given `key` if the key exists in the tree. Otherwise, it returns the node that would have been the parent of the node with with the given `key`, if it existed in the tree.
- `_add_child(p, u)` - helper method; adds node `u` as a child of node `p`, assuming that node `p` has at most one child. If node `p` already has two children, a `ValueError` is raised with message `"Parent node already has two children."`
- `_splice(u)` - helper method, links the parent of given node `u` to the child of node `u`, assuming `u` has at most has at most one child. This is equivalent to removing node `u` from the tree. Returns the node `u`.
- `_remove_node(u)` - helper method, removes node `u` from the binary search tree and returns the node that was removed.

(Optional) Test your data structures:

- Create an empty `BinarySearchTree`.
- Remove one element from an empty `BinarySearchTree`. Should result in `ValueError`
- Attempt to retrieve a value in an empty `BinarySearchTree`: `find(2)` should return `None`
- Add 3 elements: `add(3,"third")`, `add(2, “second")`, `add(1, “first")`
- Check that `size()` returns 3.
- Access one element, `find(2)` should return `second`
- Find an element with a key that does not exist in the tree:
  - `_find_eq(2.5)` should return `None`
  - `find(2.5)` should return `None`
  - `_find_last(2.5).v` should return `"second"`
- Remove one element: `remove(3)` should return `"third"`. Check that `size()` returns 2.
- Access the removed element: `find(3)` should return `None`
- Add 3 elements: `add(3, "third")`, `add(5,"fifth")`, `add(4,"fourth")`.
- Check that `size()` returns 5.
- Find an element with key that does not exist in the tree:
  - `_find_eq(3.4)` should return `None`
  - `find(3.4).v` should return `None`
  - `_find_last(3.4).v` should return `"fourth"`
- Print the key and values using the list returned by `in_order()`. The output should be:

```
1: first
2: second
3: third
4: fourth
5: fifth
```

- Print the key and values using the list returned by `pre_order()`. The output should be:

```
2: second
1: first
3: third
5: fifth
4: fourth
```

- Print the key and values using the list returned by `post_order()`. The output should be:

```
1: first
4: fourth
5: fifth
3: third
2: second
```

- Print the key and values of the list returned by `bf_order()`. The output should be:

```
2: second
1: first
3: third
5: fifth
4: fourth
```

- Print the height of the tree. It should return 4.

Changes to the BookStore System

Files to Modify:

(i) BookStore.py
(ii) BinarySearchTree.py

Directions:

1. Modify the constructor in `BookStore` so that it includes an additional attribute called `self.sortedTitleIndices`. Initialize it to an empty `BinarySearchTree` object. This binary search tree will be used to map `Book` titles to their corresponding index in `self.BookCatalogue` (i.e. it will allow us to keep track of the index where each `Book` object is located based on its title).
2. Recall that `Book` objects have attributes `key`, `title`, `group`, `rank`, and `similar`. Modify the `loadCatalog()` method so that when a `Book` gets loaded into the catalogue, a new element consisting of the book `title` and the book's index in the catalogue is stored in the binary search tree. The pseudocode is as follows:

   ```
   For each row i in books.txt
        b = Book(key, title, group, rank, similar)
        bookCatalog.append(b)
        sortedTitleIndices.add(title, bookCatalog.size()-1)
   ```

   The attribute `self.bookCatalogue` should be an `ArrayList`.
3. Add a method to `BinarySearchTree` that returns the node containing the given `key` if it exists; otherwise it returns the node with the smallest key greater than the given `key`. For example, if a binary search tree has nodes: `[(7, 7), (3, 3), (10, 10), (1, 1), (5, 5), (9, 9), (12, 12), (11, 11), (14, 14)]`, then the node with smallest key greater than or equal to 8 is (9, 9). The node with smallest key greater than or equal to 12 is (12, 12), etc.

   To motivate the algorithm, you can begin by searching at the root for a node with a key greater than or equal to the given `key`. If in this process you find a node with key matching the given `key`, then return the node. Otherwise, once you have found a node, say `u`, with a key larger than the given `key`, search the left subtree of node `u` for a smaller key that is still greater than or equal to the given `key`. Again, if during this process you happen to find a node matching the given `key`, return the node. Otherwise, update `u` to be the newly found node that contains a smaller key than the key in `u`, but is still larger than the given `key`. Below is a template of the algorithm to get you started:

```
Initialize current to be the root.
Initialize smallest to be None.
While current is not None:
  • If key < current.key, then
        # What should happen?
  • Else if key > current.key, then
        # What should happen?
  • Otherwise,
        # What should happen?
return smallest
```

4. Add a method to `BookStore` called `addBookByPrefix(prefix)` that adds the first matched book containing `prefix` in the title, when the book titles are sorted in alphabetical order. The method should return the **title of the book** that was added to the cart, if the `prefix` was matched, `None` otherwise.

   For example, if the catalogue contains books with titles,

   "World of Borrowed Time"
   "World of Byzantine Architecture"
   "World of Cats"
   "World of Chaos & Darkness"
   "World of Chocolates"
   "World of Daisies"

   Then, `addBookByPrefix(‘‘World of C")` should add the `Book` with title "World of Cats", and return the title. Furthermore, `addBookByPrefix(‘‘World of Co")` should not add any book to the cart, and should return `None`.

   HINTS:

   I. If the given `prefix` exists as a title itself, then `prefix` will match a key in `self.sortedTitleIndices`. Otherwise, a `book` with a title that may contain the `prefix` must be alphabetically greater than `prefix`, i.e., `prefix` < `book.title`. If we want to find the **first** alphabetically, matched book potentially containing the `prefix`, then we are looking for the smallest key (i.e. title) in `self.sortedTitleIndices` that is greater than or equal to `prefix`.
   II. Your method from problem 3. of the BookStore System section might come in handy.
   III. You should make sure that a book title begins with `prefix` in order for it to be considered a match, i.e. you should be checking if `title[0:n] == prefix` where `n = len(prefix)`.

Changes to the Calculator System

Files to Modify: Calculator.py.

Directions:

Implement the following methods:

1. `_build_parse_tree(expr)` - builds and returns a `BinaryTree` object representing the parse tree of the given expression `expr`.
2. `_evaluate(u)` - recursive, helper method; evaluates the given node `u`. If `u` holds a variable, it returns the stored value for the variable (if it exists). If `u` holds an operator, then recursively evaluates the expression `value of u.left, operator value of u, value of u.right`

Changes to Main

Files to Modify: main.py

Directions:

1. Modify `menu_bookstore_system()` so that it includes a new option to add a book to the shopping cart by prefix. Your new list of options should read as follows:

```
s FIFO shopping cart
r Random shopping cart
1 Load book catalog
2 Remove a book by index from catalog
3 Add a book by index to shopping cart
4 Remove from the shopping cart
5 Search book by infix
6 Get cart best-seller
7 Add a book by key to shopping cart
8 Add a book by title prefix to shopping cart
0 Return to main menu
```

   If the user chooses option 8, the system should prompt for a prefix. If the prefix exists, the system will add the corresponding book to the cart and display the message,

```
Added first matched title: <Insert book title>
```

   If no book with given prefix exists, the message `Error: Prefix was not found.` is displayed.

   (Optional) Test Your Progress:
   If you implemented `BookStore`, `BinarySearchTree`, and option 8 of `menu_bookstore_system()` correctly, you should see the following results:

   Example 1:

```
Introduce the prefix: Soup for
Added first matched title: Soup for President
```

   Example 2:

```
Introduce the prefix: World of the
Added first matched title: World of the American Pit Bull Terrier
```

   Example 3:

```
Introduce the prefix: Tomorrow will be
Error: Prefix was not found.
```

   Example 4:

```
Introduce the prefix: Sound of Music
Added first matched title: Sound of Music, The
```

2. Modify `menu_calculator()` so that it includes a new option to evaluate an expression. Your new list of options should read as follows:

```
1 Check mathematical expression
2 Store variable values
3 Print expression with values
4 Evaluate expression
0 Return to main menu
```

   If the user chooses option 4, the system should prompt them for an expression. If all the variables in the expression have been defined, then the system will display the expression with the values substituted in, and the final evaluation. For example, assuming the user has already stored variable values `alpha1 = 2`, `alpha2 = 4`, `beta1 = 3`, `beta2 = 5`, option 4 would look like the following:

```
Enter the expression:  <user enters ((alpha1+beta2)*(alpha2-beta1))>
Evaluating expression:  ((2.0+5.0)*(4.0-3.0))
Result:  7.0
```

   If one or more variables are missing defined values, then the system must display the error message: `"Error - Not all variable values are defined."` using the following format,

```
Enter the expression: <user enters ((alpha1+beta2)*(alpha2-beta1)//lambda)>
Result: Error - Not all variable values are defined.
```

SUBMISSION PROCESS

Submit to CodePost

- `BinaryTree.py`
- `BinarySearchTree.py`
- `main.py`
- `BookStore.py`
- `Calculator.py`

RUBRIC

	Full Credit	Partial Credit Pts. vary; See CodePost	No Credit 0 pts.
`BinaryTree` implementation	7 pts: Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
`BinarySearchTree` implementation	11 pts: Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
`_build_parse_tree(e)` implementation	2 pts: Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
`_evaluate(u)` implementation	2 pts Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
`Bookstore Main Menu` implementation	4 pts. Implementation is correct and passes all CodePost tests	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
`Calculator Main Menu` implementation	2 pts. Implementation is correct and passes all CodePost tests	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.