CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf

Assignment 7 - Binary Search Trees II

Due Sunday, October 25th 2020, 11:59 PM

OBJECTIVES

1. **Delete nodes in a BST**
2. **Create a super data structure combining BST and LL**

[NOTE: Starter code for MovieTree.cpp will be released after closure of assignment 6]

Overview

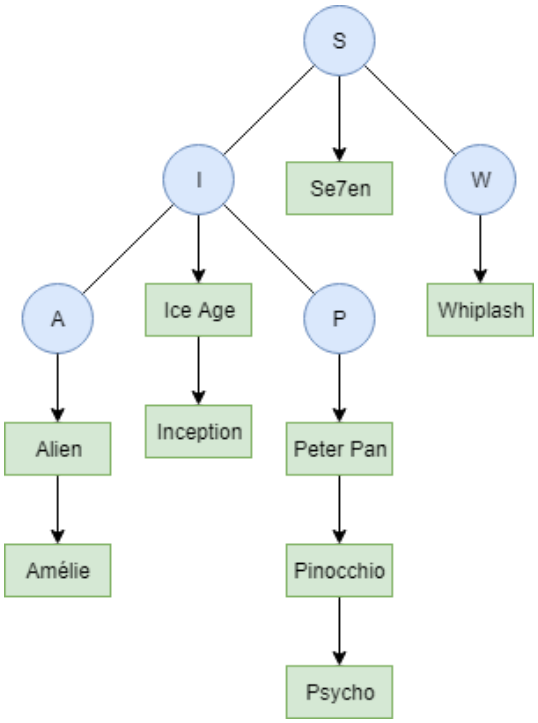
Conceptually, this assignment builds off the previous assignment but with a different data-structure. Please make sure the header files are downloaded from the current assignment and not the previous one. *DO NOT MODIFY THE HEADER FILE. However, you may implement helper functions in your .cpp file.*

MovieTree Class

Your task is to implement a Binary Search Tree (BST) where each node is a Linked List (LL) of movies. You are also required to perform a left rotation of a given node while maintaining the BST property. The nodes of the tree are alphabetically ordered i.e., with '**d**' as a parent node '**c**' appears in its left sub-tree and '**e**' appears in its right sub-tree. The characters are the first letter of the movie. As more than one movie can have the same starting character, within the node, they are stored as an alphabetically sorted Linked List. For example:

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf



`MovieTree::MovieTree()`

➔ Constructor: Initialize any member variables of the class to default

`MovieTree::~MovieTree()`

➔ Destructor: Free all memory that was allocated. This includes deallocating the memory of the underlying nodes' Linked Lists as well.

`void MovieTree::showMovieCollection()`

➔ Print every movie in the data structure in alphabetical order of titles using the following format. For `TreeNode` **t** and `LLMovieNode` **m**:

```
// for every TreeNode (t) in the tree

cout << "Movies starting with letter: " << t->titleChar << endl;

// for every LLMovieNode (m) attached to t
cout << " >> " << m->title << " " << m->rating << endl;
```

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf

Sample output format

```
Movies starting with letter: B
 >> Bowling for Columbine 8
Movies starting with letter: D
 >> Dancin' Outlaw 8.1
 >> Dogtown and Z-Boys 7.7
 >> Down from the Mountain 7.4
```

`void MovieTree::insertMovie(int ranking, string title, int year, float rating)`

- ➔ Add a movie to the data structure in the correct place based on its `title`.
  - ◆ Create a Linked List node with the associated data (`ranking, title, year, and rating`)
  - ◆ If no tree node exists (based on the first character of the title), create a new tree node and insert the newly created Linked List node into it.
  - ◆ Otherwise, insert into an existing Linked List and ensure alphabetical order is maintained.

    ```
    node->title < node->next->title
    ```

    Hint: you can compare strings with <, >, ==, etc. Also, you may assume that no two movies have the same title

- ➔ **When adding a new `TreeNode`, ensure its parent pointers are set.**

`void MovieTree:: removeMovieRecord(std::string title)`

- ➔ Delete the Linked List node that contains the `title`.
  - ◆ If deletion results in an empty Linked List, delete the associated tree node. And replace the tree node with its **inorder successor** (min of the right sub-tree).
  - ◆ If the movie does not exist in the data-structure, print the following message

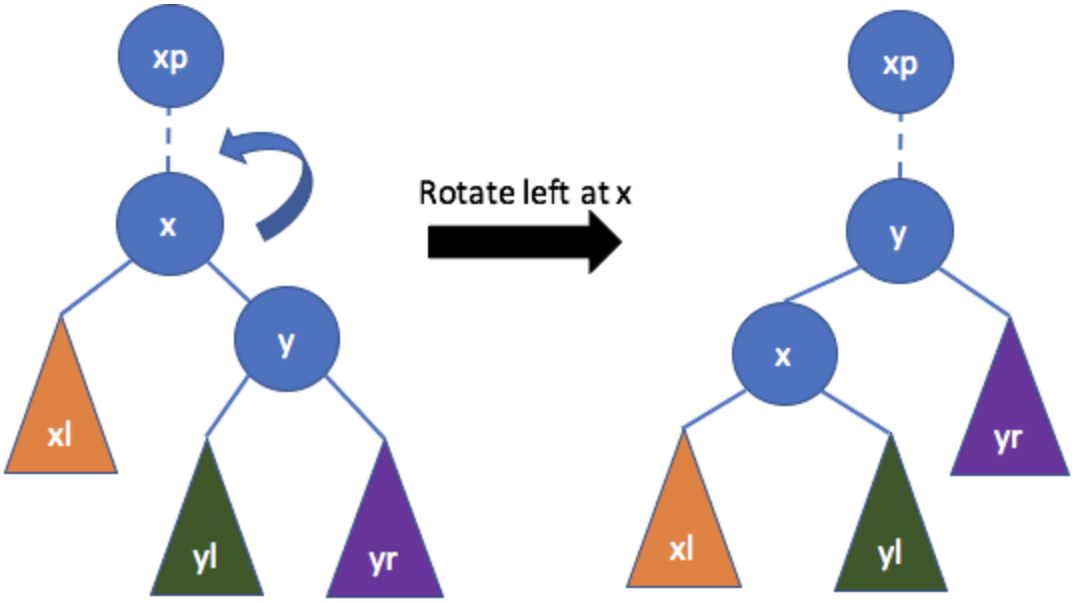
```
cout << "Movie not found."<< endl;
```

- ➔ **Make sure to adjust the parent pointer when deleting a tree node.**

`d MovieTree::leftRotation(TreeNode* curr)`

Rotate the node `curr` towards the left. Refer to the following illustration. A left rotation is performed at node **x.**

- ➔ Set the parent pointers accordingly:
  - ◆ Parent of `x` becomes the parent of `y`
  - ◆ Parent of `x` becomes `y`.
- ➔ Set the subtree (left and right children) pointers accordingly.
  - ◆ The left subtree of `y` becomes the right subtree of `x`.
  - ◆ `x` and its descendants become the left subtree of `y`.
- ➔ If `x` was the left (or right) subtree of `xp`, make `y` the left (or right) subtree of `x` respectively. This can be checked by comparing the title characters of the parent and the child's node.
- ➔ Ensure boundary conditions are accounted for:
  - ◆ `X` is root.
  - ◆ `X` has no right child.



The following functions are available as a part of the starter code.

`void MovieTree::inorderTraversal()`

This function performs an inorder traversal in the tree, printing the title character at each node. This will print the title characters in an inorder fashion. This will be useful for debugging purposes. If you want, you may use this function to create your `preorderTraversal()` and `postorderTraversal()` functions.

`TreeNode* MovieTree::searchCharNode(char key)`

This function returns a node in the tree with the titleChar key.

Driver

Code for the driver (main function) is given as part of the starter code. Here is a brief description about the main function.

Your main function should first read information about each movie from a file and store that information in a `MovieTree` object. The name of the file is passed as a command-line argument. An example file is *Movies.csv* on Moodle. It is in the format:

```
<Movie 1 ranking>,<Movie 1 title>,<Movie 1 year>,<Movie 1 rating>
<Movie 2 ranking>,<Movie 2 title>,<Movie 2 year>,<Movie 2 rating>
Etc...
```

Note: For autograding's sake, insert the nodes to the tree in the order they are read in. **After** reading in the information on each movie from the file, display a menu to the user.

```
cout << "======Main Menu======" << endl;
cout << "1. Delete a movie" << endl;
cout << "2. Show the inventory" << endl;
cout << "3. Left rotate the tree" << endl;
cout << "4. Quit" << endl;
```

The options should have the following behavior:

- **Print the inventory:** Call your tree's **`showMovieCollection`** function
- **Delete a movie:** Call your **`removeMovieRecord`** function on a title specified by the user. Prompt the user for a movie title using the following code:

```
cout << "Enter a movie title:" << endl;
```

- **Left rotate the tree:** Ask the user to enter the `titleChar` for the node to rotate left. If that is a valid node perform the rotation.

```
cout << ""give the titleChar of the node:"" << endl;
string nodename;
getline(cin, nodename);
TreeNode* rNode = movies.searchCharNode(nodename[0]);
if (rNode)
      movies.leftRotation(rNode);
```

- **Quit:** Exit after printing a friendly message to the user:

```
cout << "Goodbye!" << endl;
```

"Please note that once you are done with your assignment on code runner you need to click on 'finish attempt' and the 'submit all and finish'. If you don't do this, you will not get graded."