CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

Assignment 7 - Binary Search Trees II

Due Sunday, March 15th 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

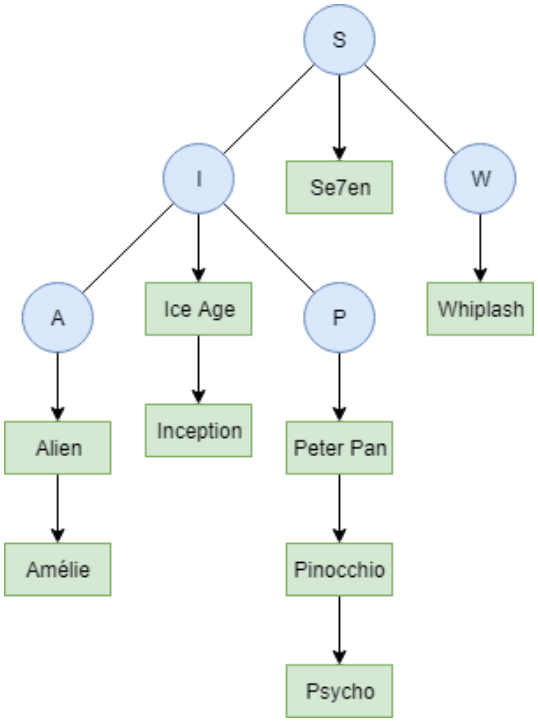
This assignment builds off of the previous one conceptually, although the data structure is fundamentally different. Make sure to use the ***MovieTree.hpp*** file uploaded on Moodle for assignment 7, **not** the ***MovieTree.hpp*** of the previous assignment 6. As usual, **do not** modify the header file. *You may implement helper functions in your .cpp file to facilitate recursion if you want as long as you don't add those functions to the MovieTree class*.

MovieTree Class

Your task is to implement a binary search tree of linked lists of movies and perform left rotation on the nodes of this tree. *Disclaimer: this diabolical super data structure is not practical; it is meant to challenge you and improve your skills manipulating and traversing BST's and LL's.* Tree nodes will contain a letter of the alphabet and a linked list. The linked list will be an alphabetically sorted list of movies which start with that letter. For example:

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi



MovieTree::MovieTree()

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

MovieTree::~MovieTree()

➔ Destructor: Free all memory that was allocated

void MovieTree::printMovieInventory()

➔ 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;
```

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::addMovie(int ranking, std::string title, int year, float rating)

➔ Add a movie to the data structure in the correct place based on its **title**.

- If there is no tree node corresponding to the first letter of **title**, create it and insert it in the tree in the alphabetically correct position
- Create a linked list node with **ranking**, **title**, **year** and **rating**, and insert it in the linked list associated with the tree node associated with the first letter of **title**. The linked list must also be in alphabetical order, such that for each **node,**

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

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

- **Make sure to set the parent pointers when adding a new node to the tree.**

void MovieTree:: deleteMovie(std::string title)

➔ Delete the linked list node that contains the title. If as a result of this deletion, the linked list becomes empty, delete the associated tree node. **When deleting a tree node with both children take the replacement from 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: " << title << " not found, cannot delete." << endl;
```

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

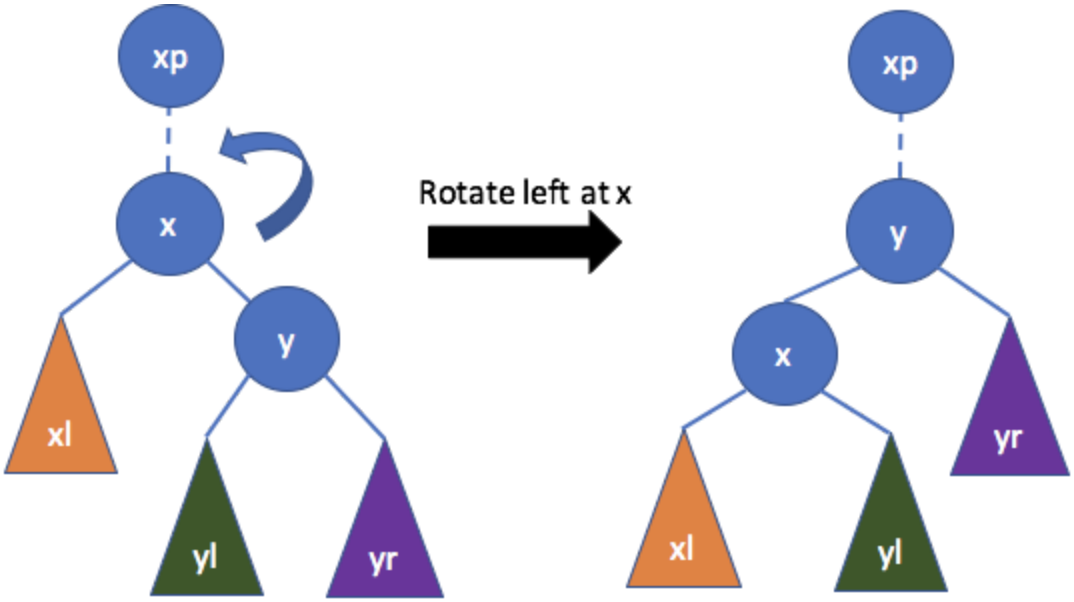
void MovieTree::leftRotation(TreeNode* curr)

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: After the rotation, the parent of **x** now becomes the parent of **y**. **y** becomes parent of **x**.
- Set the children pointers: Three children pointers are modified. i) The left subtree of **y** becomes the right subtree of **x**. ii) **x** and it's left descendants become the left subtree of **y**, with **x** as the left child. iii) If **x** was the left (or right) child of **xp** before

the rotation, make **y** the left (or right) child of **xp** after rotation accordingly. This can be implemented by comparing the titles of **y** and **xp**: if **y.title** < **xp.title**, make **y** the left child of **xp**, else make **y** the right child of **xp**. A less riskier approach is to keep track of whether **x** was the left child or the right child before the rotation, and make **y** the left or the right child accordingly.

- Take care of the boundary cases: i) If **x** was the root before rotation, make **y** the new root. ii) If **x** does not have a right child, don't perform any rotation.



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 function is available in the starter code.** This will be useful for debugging purposes. If you want, you may use this function to create your preorderTraversal() and postorderTraversal() functions.

TreeNode* MovieTree::searchChar(char key)

This function returns a node in the tree with the titleChar key. **This function is available in the starter code.**

Driver

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

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

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 with this information should be passed in 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. Print 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 **printMovieInventory** function
- **Delete a movie:** Call your **deleteMovie** 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.searchChar(nodename[0]);
  if (rNode)
        movies.leftRotation(rNode);
  ```

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

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

CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

"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."