CSCI 2270 Data Structures and Algorithms Summer 2017
Instructor: Holler/MonteroQuesada
Assignment 6

Due Friday, July 14th, by 11am Assignment 6 Part A (Pseudocode)
Due Wednesday, July 19th, by 8am Assignment 6 Part B (Implementation)

Red-black Trees

In this assignment, you will continue working with the movie data from the BST assignment and store that data in a red-black tree. A red-black tree is a self- balancing binary search tree. The main difference between this assignment and previous assignments is that you will need to balance the tree each time a new movie node is added or deleted.

For each of the movies in the movie nodes in the movie tree, the following information is kept:

- IMDB ranking
- Title
- Year released
- Quantity in stock
- Node color

Your program will have a menu similar to previous assignments from which the user can select options. In this assignment, your menu needs to include options for finding a movie, renting a movie, printing the inventory, deleting a movie, counting the number of movies, counting levels in the tree, and quitting the program.

Use the same Assignment5Movies.txt file that you used for the BST movie assignment. The name of the input file needs to be handled as a command-line argument

Your program needs to incorporate the following functionality.

Insert all the movies in the tree.

When the user starts the program they will pass it the name of the text file that contains all movie information. Your program needs to handle that command-line argument, open the file, and read all movie data in the file. From this data, build the red-black tree ordered by movie title. As movies are added to the tree, the red-black tree-balancing algorithm should be applied. All information about the movie should also be included in the movie node in the tree. Note: the data should be added to the tree in the order it is read in.

Find a movie.

When the user selects this option from the menu, they should be prompted for the name of the movie. If the movie is found in the tree, display the movie information. If the movie is not found, your program should display, "Movie not found." This option is similar to rent movie, however, you are not updating the quantity.

Rent a movie.

When the user selects this option from the menu, they should be prompted for the name of the movie. If the movie is found in the tree, your program should update the Quantity in stock property of the movie and display the new information about the movie. When the Quantity is 0, the movie should be deleted from the tree. When a movie is deleted, the tree should be rebalanced. If the movie is not found, your program should display, "Movie not found."

Print the entire inventory.

When the user selects this option from the menu, your program should display all movie titles and the quantity available in sorted order by title. See the lecture notes on in-order tree traversal for more information.

Delete a movie.

When the user selects this option, they should be prompted for the title of the movie to delete. Your code should then search the tree for that movie, delete it if it's found, and then perform any necessary tree balancing to restore the red-black tree properties. If the movie is not found in the search process, print "Movie not found." and do not attempt to delete.

Count movies in the tree.

When the user selects this option, your program should do an in-order tree traversal and count the total movie nodes in the tree and print the value.

Count longest path.

When the user selects this option, your program needs to determine the longest path from the root of the tree to the bottom of tree, not including empty nodes.

Quit the program.

When the user selects this option, your program should delete the tree using a post-order traversal.

Use the cout statements in Appendix A to set the order of the menu options.

Helper Function to check tree balancing

There is a file on Moodle called rbValid.cpp that you can use to check that your tree-balancing algorithm is correct. This code checks that the red-black properties have been restored to the tree. The MovieTree.hpp file includes a definition for the helper function as a private method in the MovieTree class. You will need to copy the code out of the rbValid.cpp file and incorporate it into MovieTree.cpp. Call the function after adding a movie to the tree to verify that your tree balancing works. This is for debugging purposes. Leaving the code in your program when you submit to coderunner should not affect your results. There are additional BST properties that are not being checked with this code – it is not checking that the values in the tree are valid for all left and right sub-trees of a node.

There is also a website that shows visually the red-black balancing process: https://www.cs.usfca.edu/~galles/visualization/RedBlack.html. Note: this website uses the maximum value in the left sub-tree as the replacement for a deletion with two children. Your code needs to use the minimum value in the right sub-tree.

Implementation details

The MovieTree.hpp file on Moodle includes the prototype for the red-black tree class for this assignment. You need to implement the class functionality in a corresponding MovieTree.cpp file and Assignment6.cpp file. You can compile your code on the command-line using g++

```
g++ -std=c++11 MovieTree.cpp Assignment6.cpp -o Assignment6
```

and then run your program on the command-line using

```
./Assignment6 <file_name>
```

In MovieTree.hpp, you'll notice that there is now a node called *nil. You will need to allocate memory for *nil in the MovieTree constructor and then use it in all cases where you previously used nullptr.

There are several components to this assignment that can be treated independently. Our advice is to tackle these components one by one, starting with updating the menu from the last assignment to meet the requirements for this assignment.

Also, remember to start early.

Pseudocode (20%)

For this assignment, the pseudocode portion will correlate to the functionality of balancing a node in red black trees.

Implementation (80%)

Submit your Assignment 6 code by opening the link to the CodeRunner question and pasting your code on the text box. Make sure your code is commented enough to describe what it is doing. Include a comment block at the top with your name, assignment number, and course instructor.

If you do not get your assignment to run on CodeRunner, you will have the option of scheduling an interview grade with your TA to get a grade for the assignment. Even if you do get the assignment to run on CodeRunner, you can schedule the interview if you just want to talk about the assignment and get feedback on your implementation. Consider the TA office hours if you think that the discussion would be longer than 10min.

What to do if you have questions

There are several ways to get help on assignments in 2270, and depending on your question, some sources are better than others. The Piazza discussion forum is a good place to post technical questions, such as how to add a node to a linked list. When you answer other students' questions on the forum, please do not post entire assignment solutions. The CAs and TAs are also a good source of technical information, especially questions about C++. If, after reading the assignment write-up, you need clarification on what you're being asked to do in the assignment, the TAs and the Instructor are better sources of information than the discussion forum or the CAs.

Appendix A – cout statements that COG expects

Display menu

```
cout << "======Main Menu======" << endl;
cout << "1. Find a movie" << endl;
cout << "2. Rent a movie" << endl;
cout << "3. Print the inventory" << endl;
cout << "4. Delete a movie" << endl;
cout << "5. Count the movies" << endl;
cout << "6. Count the longest path" << endl;
```

```
cout << "7. Quit" << endl;
```

Find a movie

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

Display found movie information

```
cout << "Movie Info:" << endl;
cout << "===========" << endl;
cout << "Ranking:" << foundMovie->ranking << endl;
cout << "Title:" << foundMovie->title << endl;
cout << "Year:" << foundMovie->year << endl;
cout << "Quantity:" << foundMovie->quantity << endl;
```

If movie not found

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

Rent a movie

```
//If movie is in stock
cout << "Movie has been rented." << endl;
cout << "Movie Info:" << endl;
cout << "===========" << endl;
cout << "Ranking:" << foundMovie->ranking << endl;
cout << "Title:" << foundMovie->title << endl;
cout << "Year:" << foundMovie->year << endl;
cout << "Quantity:" << foundMovie->quantity << endl;

//If movie not found in tree
cout << "Movie not found." << endl;
```

Print the inventory

```
//For all movies in tree
cout<<"Movie: "<<node->title<<" "<<node->quantity<<endl;
```

Count movies in the tree

```
cout<<"Tree contains: "<<mt.countMovieNodes()<<" movies."
<< endl;
```

Longest path

```
cout << "Longest Path: " << mt->countLongestPath() << endl;
```

Delete movie

```
cout << "Enter title:" << endl;
//If movie not found in tree
cout << "Movie not found." << endl;
```

Delete all nodes in the tree

```
//For all movies in tree
cout<<"Deleting: "<<node->title<<endl;
```

Quit

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

File I/O Error

```
cout << "Could not open file\n";
```