CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman
Assignment 7
Due Sunday, March 15 by 5pm

Red-black trees

In this assignment, you will continue working with the movie data from assignments 5 and 6 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.

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 assignments 3, 4, 5, and 6 from which the user can select options. In this assignment, your menu needs to include options for renting a movie, printing the inventory, deleting a movie, counting number of movies, counting levels in the tree, and quitting the program.

Your program needs to incorporate the following functionality. Use the same Assignment5Movies.txt file that you used for assignments 5 and 6. Your program also needs to produce an output file that contains a json object of all operations performed.

Both the name of the input file and the name of the output file should be handled as command-line arguments.

1. **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. 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.*
2. **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. If the movie is not found, your program should display, "Movie not found."

3. **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.
4. **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.
5. **Count movies in the tree.**
   When the user selects this option, your program should do an inorder tree traversal and count the total movie nodes in the tree, print the value, and add the information to the json object.
6. **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 T.nil nodes.
7. **Quit the program.**
   When the user selects this option, your program should generate the json output file and exit. The name of the file is passed into the program as a command-line argument.
8. **Delete the tree in the destructor using a postorder traversal.**
   When the user selects quit, the destructor for the MovieTree class should be called and in the destructor, all of the nodes in the tree should be deleted. You need to use a postorder tree traversal or you will get segmentation fault errors.
9. **Output the operations in a json file.**
   Write the information about each add, rent, traverse, delete, count, and height operation to a json object and then output the object to a file. Information about what information to include in each operation is included in Appendix A. The only new operation in this assignment is height, all other json operations are the same as they were in Assignment 6. However, we did correct a few grading script bugs for this assignment, so you may need to update your json file to incorporate these changes.
10. **Use a T.nil node to initialize the parent, left child, and right child pointers instead of using NULL.**
    In MovieTree.h, 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 NULL.

Implementation details

Your red-black tree should be implemented in a class. You are provided with a MovieTree.h file on Moodle that includes the class prototype for this assignment. You need to implement the class functionality in a corresponding MovieTree.cpp file and Assignment7.cpp file. To submit your work, zip all files together and submit them to COG. If you do not get your assignment working on COG, you will have the option of a grading interview.

Helper Function to check tree balancing

There is a file on Moodle called isValid.cpp that you can use to check that your tree balancing algorithm is correct. The MovieTree.h 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 isValid.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. Do not write the results of this function to your json file. Leaving the code in your program when you submit to COG should not affect your results.

Appendix A – json key:value pairs that COG expects

When your program performs an operation, such as deleting a node, or adding a node to the tree, you should add information specific to that operation to your json object that will eventually be written to a file just before the program quits. Each operation performed should be assigned the operations counter as the key, and then depending on the type of operation, other information is included. The operation object should contain a key called “operation”, which can have the values “add”, “delete”, “traverse”, “rent”, “count”, or “height”. A sample for each operation type is shown here:

```
{
  "1":{
    "operation":"add",
    "parameter":"movie title",
    "output":[
      "movie title 1",
      "movie title 2",
      "movie title 3",
      "movie title z"
    ]
  },

  "2":{
    "operation":"delete",
    "parameter":"movie title",
    "output":[
      "movie title 1",
      "movie title 2",
      "movie title 3",
```

```
      "movie title z"
    ]
  },

  "3":{
    "operation":"traverse",
    "output":[
      "movie title 1",
      "movie title 2",
      "movie title 3",
      "movie title z"
    ]
  },
  "4":{
    "operation":"count",
    "output":”50”
  },
  "5":{
    "operation":"rent",
    "parameter":"movie title",
    "output":”10”
  }
  "6":{
    "operation":"height",
     "output":”10”
  }

}
```

In this example, the first operation performed is an “add”. The value for “parameter” is the movie title of the added movie node, and the “output” value is the array of movie titles observed as we traverse the tree searching for the correct spot to add the movie node. The operation has a “1” because we did it first. If we perform 50 add operations, each of them would be given a number 1…50 in the order they were performed.

The “delete” operation should be called when the user deletes a node, either through the menu or due to the Quantity becoming 0. The “delete” operation includes the same data as the “add” operation.

The “traverse” operation should be called when the user selects Print Inventory from the menu. The “output” is the titles observed for each movie node in the traversal.

The “count” operation should be called when the user selects Count Movies from the menu. The “output” value is the count of the number of movie nodes in the tree. Use a string for the output value.

The “rent” operation should be called when the user selects Rent a Movie from the menu, but only if the movie exists in the tree. The “parameter” value is the title of the movie to rent, and the “output” is the quantity of that movie remaining after the rental.

The “height” operation should be called when the user selects Count Longest Path from the menu. The “output” value is the number of nodes along the longest path from the root of the tree to the bottom of the tree, not including T.nil nodes. Use a string for the output value.

Example:
For example, when you are building the tree from the data in the text file, the first movie you add is “Shawshank Redemption”. You would add an object to your json, such as:

```
“1”:{
      "operation":"add",
      "parameter":"Shawshank Redemption",
      "output":[

      ]
  },
```

The “output” array contains all movie titles observed up to the one added, but not the title added. For the first movie, the “output” should be an empty array. The next movie you add is The Godfather. You would add another object to your json that looked like:

```
"2":{
      "operation":"add",
      "parameter":"The Godfather",
      "output":[
                "Shawshank Redemption",

      ]
},
```

The process continues for each movie added, with the operations counter increasing each time. When the user selects an option from the menu, you maintain the same counter, increment the counter for the next object added to the json object, and add to the same json object. For example, if the user selects print the tree after building the tree, you would add an object to your json that looked like:

```
"51":{
     "operation":"traverse",
     "output":[<array of strings of all movies titles in
     inorder tree traversal>]
},
```

Appendix B – Menu ordering

COG will select items from the menu when it runs your program. The exact text in the menu doesn't matter, but the numbering of the menu items does. Use the following menu order:

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