CSCI 2270 – Data Structures

Instructor: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Assignment 5 - Binary Search Trees

OBJECTIVES

1. **Build a binary search tree (BST)**
2. **Search and traverse a BST**

Background

In 2009, Netflix held a competition to see who could best predict user ratings for films based on previous ratings without any other information about the users or films. The grand prize of US$1,000,000 was given to the BellKor's Pragmatic Chaos team which bested Netflix's own algorithm for predicting ratings by 10.06%. This kind of data science is facilitated with the application of good data structures. In fact, cleaning and arranging data in a conducive manner is half the battle in making successful predictions. Imagine you are attempting to predict user ratings given a dataset of IMDB's top 100 movies. Building a binary search tree will enable you to search for movies and extract their features very efficiently. The movies will be accessed by their titles, but they will also store the following features:

- IMDB ranking (1-100)
- Title
- Year released
- IMDB average rating

Your binary search tree will utilize the following struct with default and overloaded constructors:

```
struct MovieNode
{
    int ranking;
    std::string title;
    int year;
    float rating;

    MovieNode *parent = nullptr;
    MovieNode *leftChild = nullptr;
    MovieNode *rightChild = nullptr;

    MovieNode() {}
    MovieNode(int r, std::string t, int y, float q) : ranking(r), title(t),
        year(y), rating(q) {}
};
```

CSCI 2270 – Data Structures

Instructor: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

MovieTree Class

Your code should implement a binary search tree of movies. A header file that lays out this tree can be found in *MovieTree.hpp* on Moodle. 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()

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

~MovieTree()

➔ Destructor: Free all memory that was allocated

MovieNode *search(string title)

➔ This private function is meant to be a helper function. Return a pointer to the node with the given **title**, or nullptr if no such movie exists

void printMovieInventory()

➔ Print every node in the tree in alphabetical order of titles using the following format

```
// for every Movie node (m) in the tree
cout << "Movie: " << m->title << " " << m->rating << endl;
```

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

➔ Add a node to the tree in the correct place based on its **title**. Every node's left children should come before it alphabetically, and every node's right children should come after it alphabetically. *Hint: you can compare strings with <, >, ==, string::compare() function etc.* You may assume that no two movies have the same title

void findMovie(string title)

➔ Find the movie with the given **title**, then print out its information:

```
cout << "Movie Info:" << endl;
cout << "==================" << endl;
cout << "Ranking:" << node->ranking << endl;
cout << "Title  :" << node->title << endl;
cout << "Year   :" << node->year << endl;
cout << "rating :" << node->rating << endl;
```

If the movie isn't found print the following message instead:

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

void queryMovies(int rating, float year)

➔ Print all the movies with a rating at least as good as **rating** that are newer than **year** in the **preorder** fashion using the following format

```
cout << "Movies that came out after " << year << " with rating at
least " << rating << ":" << endl;
// each movie that satisfies the constraints should be printed with
cout << m->title << "(" << m->year << ") " << m->rating << endl;
```

void averageRating()

➔ Print the average rating for all movies in the tree. If the tree is empty, print 0.0. Ue the following format

```
cout << "Average rating:" << average << endl;
```

Driver

Your main function should first read information about each movie from a file and store that information in a MovieTree. **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. Of course, they will still be inserted in alphabetical order, but a different insertion order may produce a different tree. **After** reading in the information on each movie from the file, display a menu to the user.

```
cout << "======Main Menu======" << endl;
cout << "1. Find a movie" << endl;
cout << "2. Query movies" << endl;
cout << "3. Print the inventory" << endl;
cout << "4. Average Rating of movies" << endl;
cout << "5. Quit" << endl;
```

The options should have the following behavior:

- **Find a movie:** Call your tree's *findMovie* function on a movie specified by the user. Prompt the user for a movie title using the following code:

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

- **Query movies:** Call your tree's **queryMovies** function on a rating and year specified by the user. Prompt the user for a rating and year using the following code:

```
cout << "Enter minimum rating:" << endl;
// get user input
cout << "Enter minimum year:" << endl;
```

- **Print the inventory:** Call your tree's **printMovieInventory** function
- **Average Rating of movies:** Call your **averageRating** function
- **Quit:** Exit after printing a friendly message to the user:

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