CSCI 2270 – Data Structures

Assignment 7

Binary Search Tree II

OBJECTIVES

Deleting a node in BST
Left rotate the tree
Right rotate the tree

NOTE: This assignment is an extension of Assignment 6.

Continuation of Previous Assignment: Binary Search Tree

Background

Conceptually, this assignment builds off the previous assignment. In this assignment, you will be concentrating on deleting a node in BST, left and right rotate BST on a given show title.

Assume that you are given the task of predicting user ratings given a dataset of TV shows. To accomplish this goal, build a Binary Search Tree (BST) that can be used to search for shows and extract their features in an efficient manner. The shows should be accessible by their titles, but they will also store the following features:

- Title
- Year released
- Show rating (G, PG, PG-13, R, etc.)
- User rating

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

```
struct ShowItem {
	string title;
	int year;
	string showRating;
	float userRating;
	ShowItem* left = NULL;
	ShowItem* right = NULL;

	ShowItem(string t, int y, string sr, float ur) {
		title = t;
		year = y;
		showRating = sr;
		userRating = ur;
	}
};
```

ShowCatalog Class

Your code should implement a binary search tree of ShowItem data type. A header file that lays out this tree can be found in *ShowCatalog.hpp*. 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 ShowCatalog class*.

ShowCatalog()

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

~ShowCatalog()

➔ Destructor: Free all memory that was allocated

void printShowCatalog()

➔ Print every node in the tree using Preorder traversal of titles using the following format:

```
 // for every Show node (s) in the tree
 cout << "Show: " << s->title << " " << s->userRating <<
endl;
```

If there is no show entry in the tree, print the following message instead:

```
 cout << "Tree is Empty. Cannot print" << endl;
```

void addShowItem(string title, int year, string showRating, float userRating)

- ➔ 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.*
- ➔ *For example*, if the root node of the tree is the show "Boys", then the show

  "Stranger Things" should be in the root's right subtree and "Arcane: League of Legends" should be in its left subtree.
- ➔ You can assume that no two shows have the same title

void getShow(string title)

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

```
cout << "Show Info:" << endl;
cout << "==================" << endl;
cout << "Title :" << node->title << endl;
cout << "Year :" << node->year << endl;
cout << "Show Rating :" << node->showRating << endl;
cout << "User Rating :" << node->userRating << endl;
```

If the show isn't found, print the following message instead:

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

void removeShow(**string title**) // TODO in this assignment

- ➔ Delete the BST node which contains show with **title**.
- ➔ Consider conditions where the node to delete can be :
  - ◆ A leaf node.
  - ◆ Node having only left child.
  - ◆ Node having only right child.
  - ◆ Node having both right child and left child.

void rightRotate(string title) // TODO in this assignment

- ➔ Rotate the node with **title** towards the right. Refer to the following illustration. A right rotation is performed at node **y**.
- ➔ Set the parent pointers accordingly:

  Parent of **x** becomes the parent of **y**

  Parent of **y** becomes **x**
- ➔ Set the subtree (left and right children) pointers accordingly.

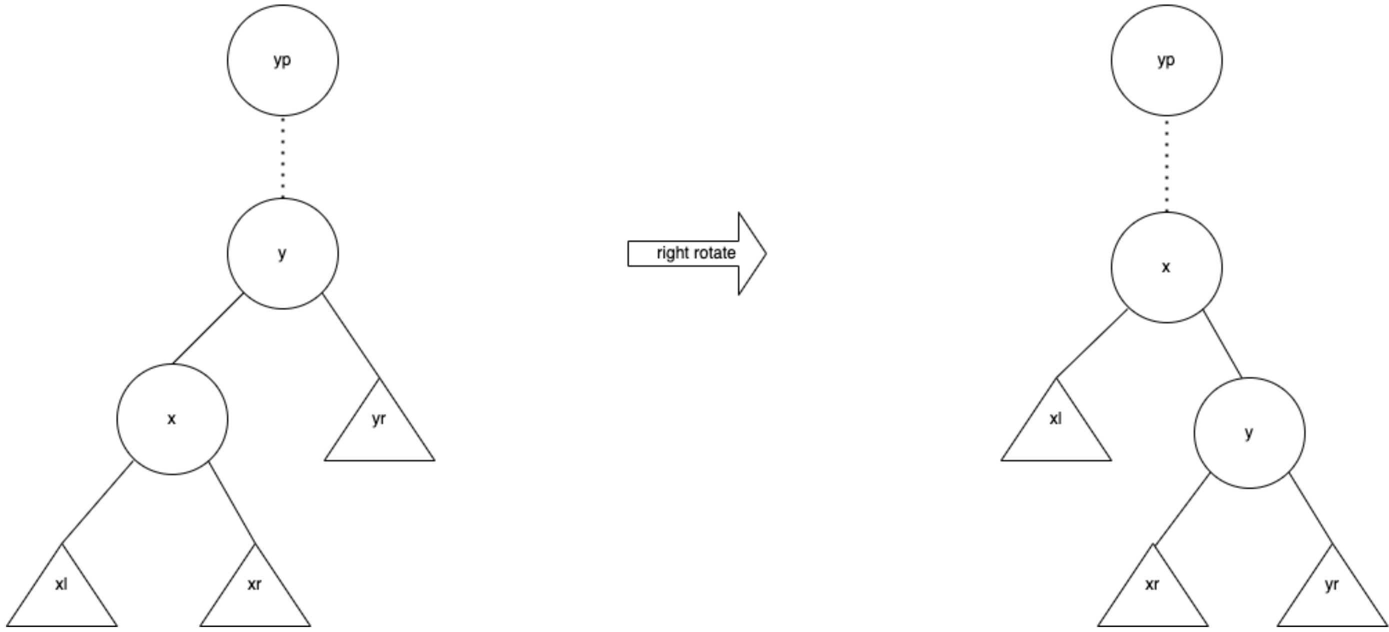
  The right subtree of **x** becomes the left subtree of **y**.

y and its descendants become the right subtree of **x**.

- ➔ If **y** was the left (or right) subtree of **yp**, make **x** the left (or right) subtree of **yp** respectively. This can be checked by comparing the title characters of the parent and the child's node.
- ➔ Ensure boundary conditions are accounted for:

  y is root.

  y has no left child



void leftRotate(string title) // TODO in this assignment

- ➔ Rotate the node with **title** 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**

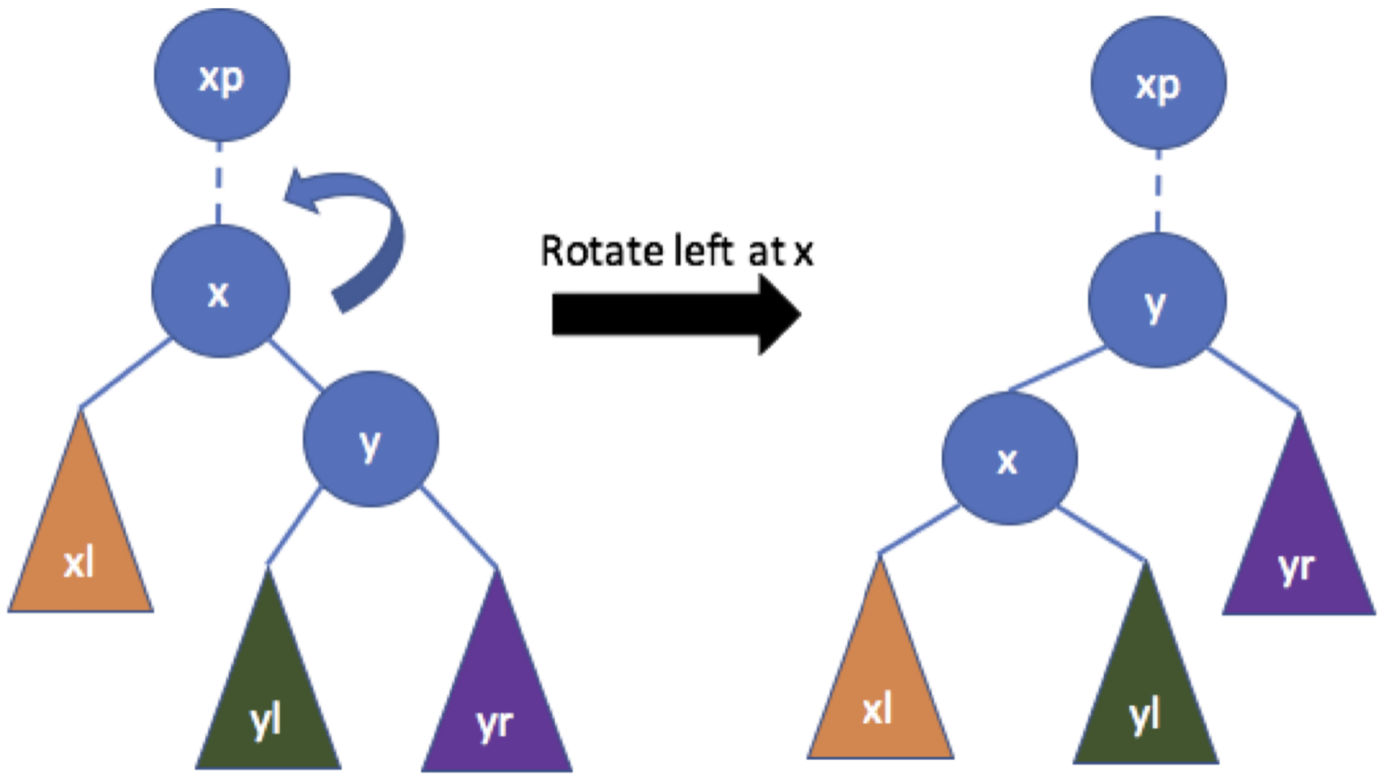
  Parent of **x** becomes **y**
- ➔ Set the subtree (left and right children) pointers accordingly.

  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.



Driver

For this assignment, the driver code has been provided to you in the app_1/*main_1.cpp* file. The main function will read information about each show from a CSV file (use shows.csv) and store that information in a ShowCatalog using your **addShowItem** function implementation.

The name of the CSV file with this information should be passed in as a command-line argument. An example file, *shows.csv*, has been provided to you. It contains the following format:

```
 <Show 1 title>,<Show 1 year>,<Show 1 show rating>,<Show 1
user rating>
 <Show 2 title>,<Show 2 year>,<Show 2 show rating>,<Show 2
user rating>
 Etc…
```

Note: For autograding's sake, insert the nodes to the tree in the order they are read in.

After reading the information on each show from the file and building the tree, the user is displayed the following menu:

```
======Main Menu======
    1. Delete a show
    2. Print show catalog
    3. Right rotate the tree
    4. Left rotate the tree
    5. Quit
    >
```

- **Delete a show:** Call your tree's **removeShow** function on a show specified by the user. The user is prompted for a show title.

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

- **Print the catalog:** Call your tree's **printShowCatalog** function
- **Right rotate on a show title** : Call your tree's rightRotate function on a show specified by the user. The user is prompted for a show title.

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

- **Left rotate on a show title**: Call your tree's leftRotate function on a show specified by the user. The user is prompted for a show title.

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

- **Quit:** Program exits after printing a friendly message to the user.

For running run_app_1(Which runs main_1.cpp):

CSCI 2270 – Data Structures

Order of function implementation

- ➔ Destructor and Constructor
- ➔ addShowItem and printShowCatalog - To be taken from assignment - 6
- ➔ getShow - To be taken from assignment - 6
- ➔ removeShow
- ➔ leftRotation , rightRotation