**Programming Assignment 2 Report**

**By: Tai Dao**

**Red-Black Tree Design and Concepts**

**Node Class**

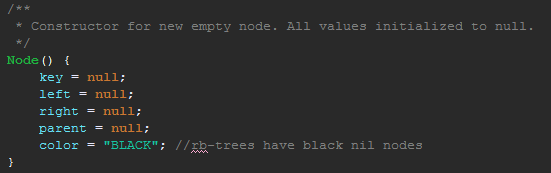
Original Binary Tree Implementation

I made a Node class to represent each node in the binary search tree. This node class typically contains pointers to the left child, right child, and parent node with getters and setters for each. These pointers typically point to nothing until the tree is built up. It also contains a key which is an object called Process.

I made this class implement compareTo for whenever I needed to do comparisons between one node and another. This compareTo compares based on the key’s (The key is the Process) priority.

New Modifications

For this class I added another String variable called color into the class. In addition to that I made the empty constructor.



This was needed for the creation of nil nodes. Nil nodes have a black color by default. I also created getters and setters for color.

**Process Class (No Changes from Binary Search Tree)**

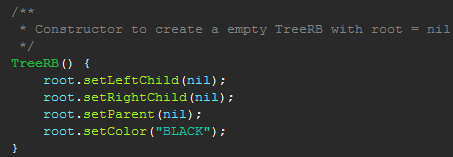
In my design I have one Object called Process. When this object is created with the constructor Process(int currentPID) a random priority from 0-9999 is assigned and a processID is assigned based on the currentPID. The currentPID is tracked in main and starts from 0. It implements comparable in order to make it possible to compare each process to each other based on priority. It has getters for the processID & priority and a setter for priority.

It also has another constructor Process() which is utilized to create a Process with the lowest possible priority of 0 and a null process PID.

**TreeRB Class**

In addition to the pseudocode in CLRS, I also modified the constructor for when the rb tree is first created.





This constructor was necessary because the initial node has to have its left, right, and parent nodes set to the same nil. In regular BST we didn’t need to point to nil nodes but in rb tree we have to.

**RedBlackTree**

Well 20 processes are supposed to be created with random priority and inserted into the rbTree on start. Then a menu with options appear to test the Tree.

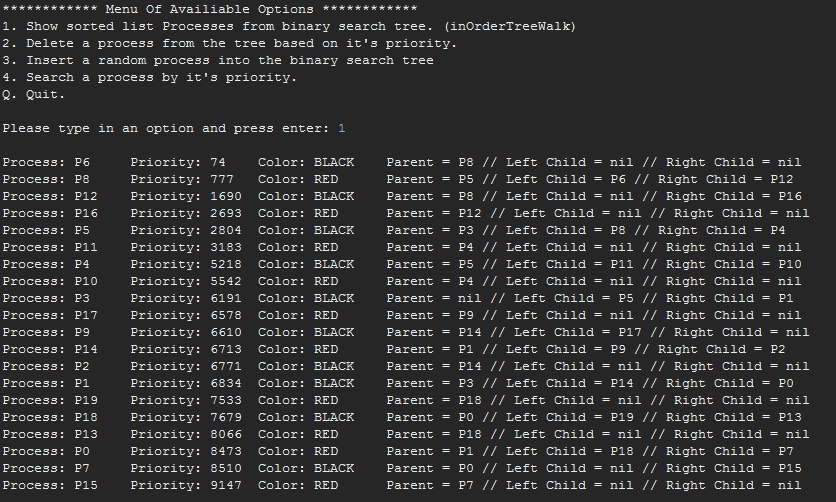
The menu allows the user to display a sorted list of processes in the BST. It also allows the user to enter in the priority of a process that they want deleted from the BST. This calls a rbTreeSearch to find the Node to delete then calls rbTreeDelete to delete that Node. It also allows the user to search for a process by its priority. Lastly this menu allows the user to insert a random process into the rbTree.

However since my program fails with more than 3 processes. I only chose to insert 3 processes.

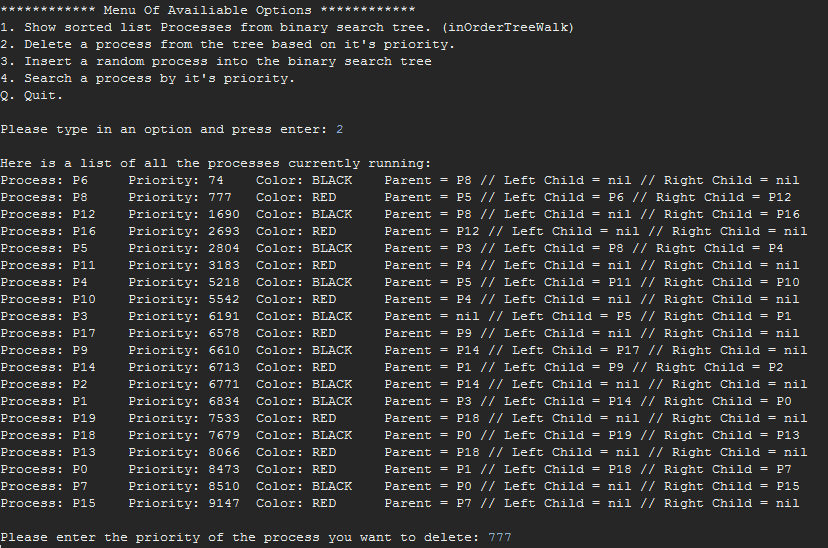
**Screenshots**

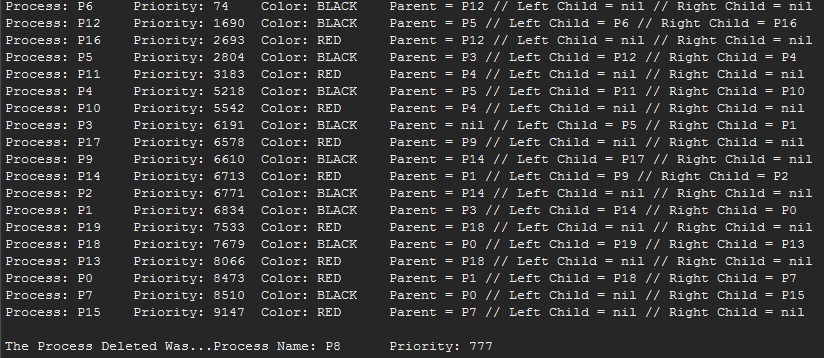
**(3 Processes… explanation under problems encountered)**

**Displaying processes in order**

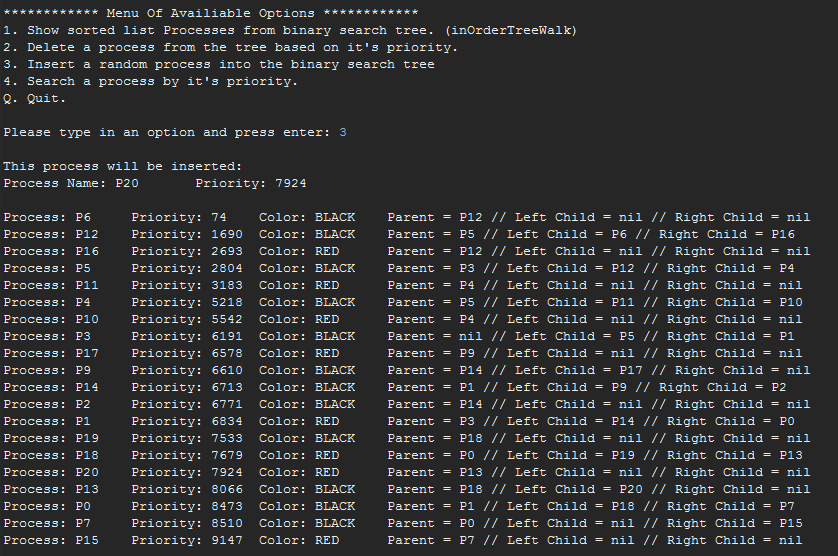


**Deleting a process**

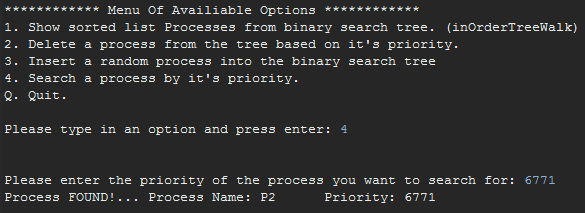




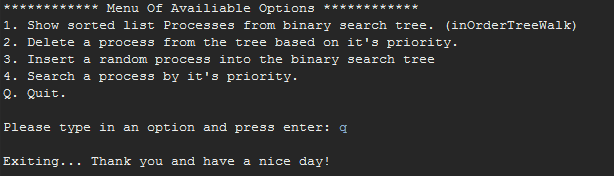
**Inserting a random process**



**Searching for a process by priority**



**Exiting the application**



**Procedures to run code**

Unzipping (Windows)

To unzip files go to https://ninite.com/. And checkmark the box next to "7-Zip". If you don't have "Java 8" checkmark the box next to Java 8 too. Click on "Get Your Ninite" and run the downloaded file to install the required applications.

After 7-Zip is finished installing right click on the zip file you want to unzip. Then select Extract To "\Dao-PA3.zip". A new folder called "Dao-PA3" will be created in the same directory as the zip file.

Running the application (Windows)

Java 8 is required. If you haven't installed it from the previous step with ninite. Go back to ninite.com and include Java 8 for the "Get Your Ninite.”

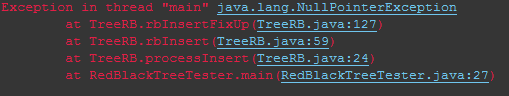
Navigate into the "RedBlackTree" folder which is inside the folder you extracted with 7-Zip. While holding 'Shift'... Right-click on a blank space inside the folder, then click "Open command window here"

Now type in the following below without quotes:

| 'java - jar RedBlackTreeTester.jar'

Then press Enter to run the application.

**Problems Encountered & Lessons Learned (BST)**



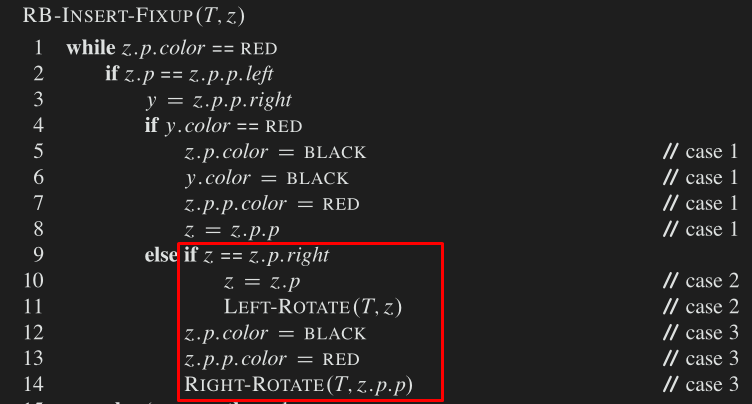
One of the major causes of my NullPointerExceptions was due to not creating the constructor for TreeRB(). I explained that the constructor was needed because the left, right, and parent of the initial root node should point to nil. If they pointed to “null” instead of “t.nil” then t.getColor() wouldn’t work because null doesn’t have a color. T.nil on the other hand has a color and it’s “BLACK”.

Because of this issue I now realize why CLRS uses T.nil. I also learned that all nil nodes pointed to the same T.nil because doing so saves memory. There’s no need to instantiate multiple nil nodes, reducing spce complexity.

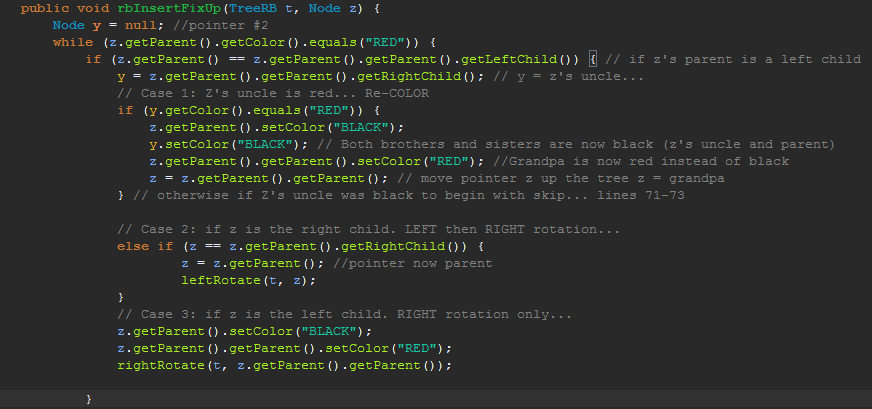
Another major problem I couldn’t solve was this…

When I inserted the first 3 processes into the RB Tree everything works fine. However whenever I decide to insert a 4th process the program would be stuck in an infinite loop in rbInsertFixup. I was so frustrated. It always happens on the 4th process because it is the case where root has two red children.

I realized it’s because the pseudocode for rbInsertFixup was misleading. I fixed it by including the. if block inside the else block.



**Before Fixing**



**After Fixing**

