**Leo Le**

**Programming Assignment 3**

**Table of Contents**

**How to start** --------------------------------------------------- **(2)**

**Main program testing (how to use)** ------------------ **(3-6)**

**Drawing** -------------------------------------------------------- **(7-22)**

**How it works** -------------------------------------------------- **(23-30)**

**Problems encountered & lessons learned** --------- **(30-31)**

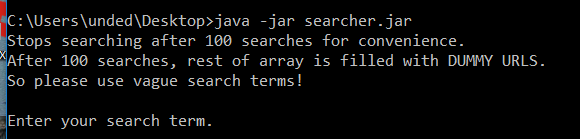
**Miscellaneous screenshots** ------------------------------------- **(32 - end)**

**File download:**

<https://drive.google.com/open?id=1BIOWXlxaT9H8toKrQd_nma23EwXtNYQ->

**HOW TO START:**

I use winrar to extract and zip my files.



1. Extract the files (With winrar, right click the zip and click “extract here”)
2. Open command prompt
3. Navigate to directory where searcher.jar is. I held it in my desktop
4. Type “**java -jar searcher.jar**”
5. The program will start in your command prompt

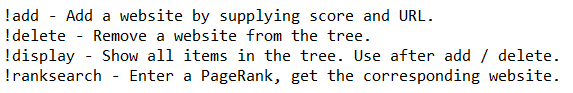
…

Enter your search term, as prompted.

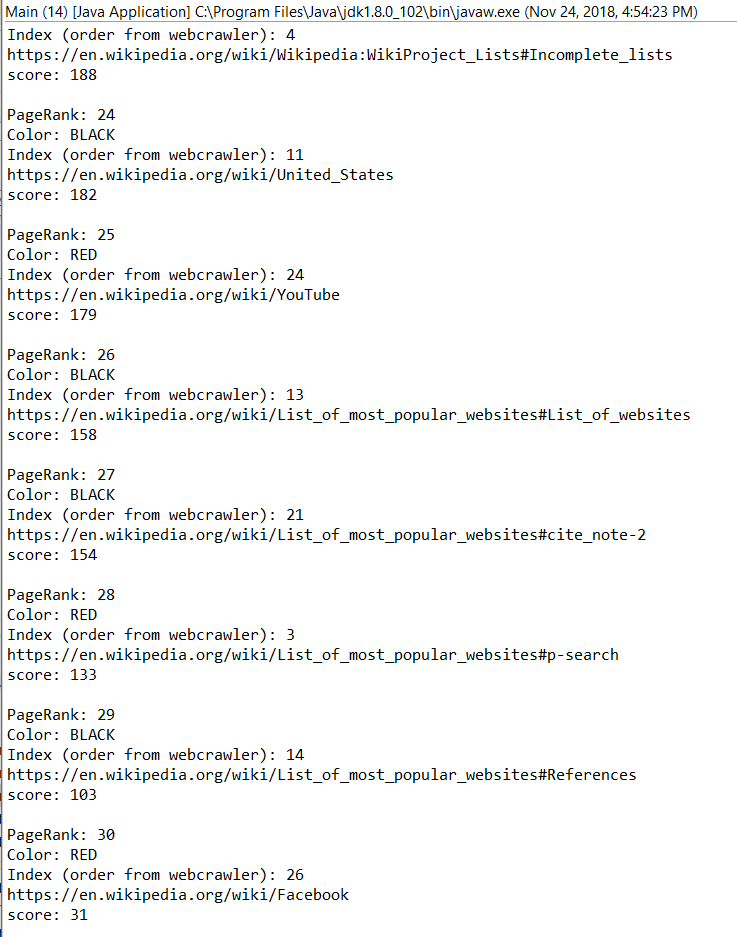


^ If you did it right, you should see that.

**Type “!commands” for a list of commands**

****

**TESTING (aka how to use)**

****

Type “**!delete”** to delete a node from the tree

Enter the ID of the node you want to delete. Entering an invalid ID will give an index OOB error

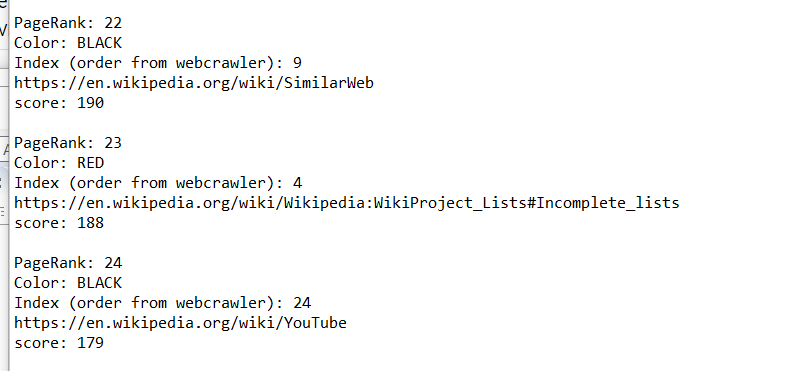


Select a website, I will select United\_States, AKA ID: 23

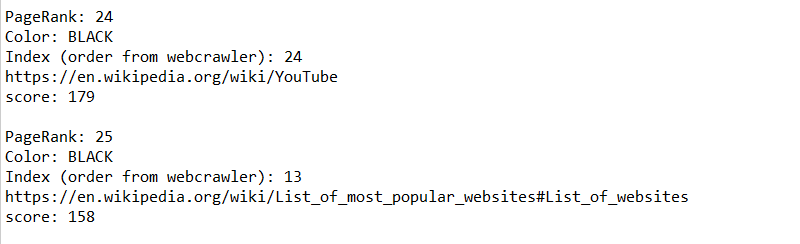
Type “23”

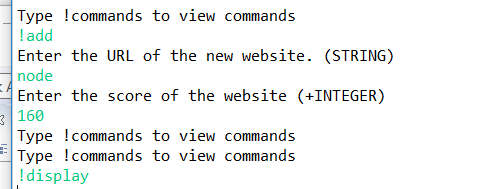


Type “**!display**” to check if the node is still there

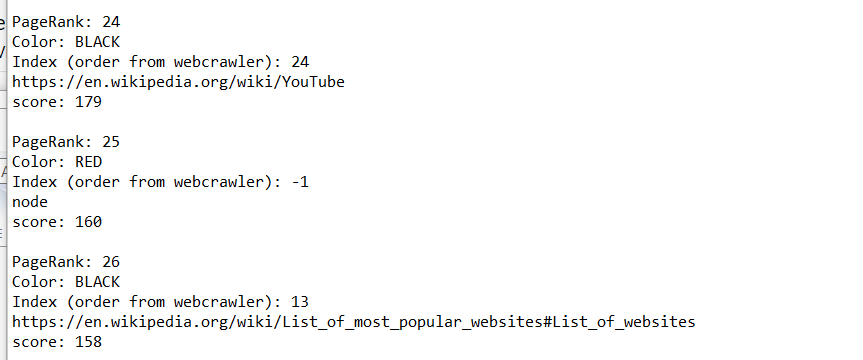


Type “!**add**” to add a node to the tree

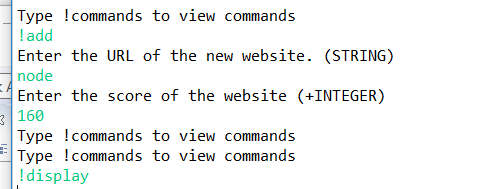
Let’s add a node with score 160, above list of popular websites and below youtube 



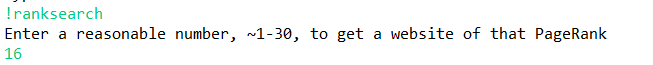
Type “!**display**” to view the results

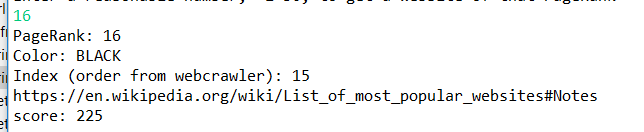


Type “!**ranksearch”**  to get a node based off of its pagerank

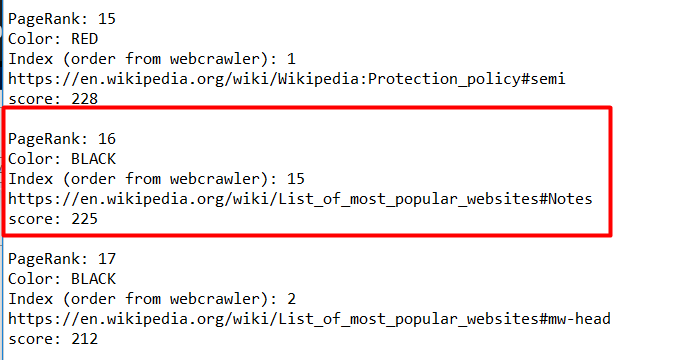


Enter a reasonable pagerank, or else you get an index OOB error.





Type “!**display**” to see make sure you got the right pageRank

****

**DRAWING**

For all my drawing tests, I used the array in the order

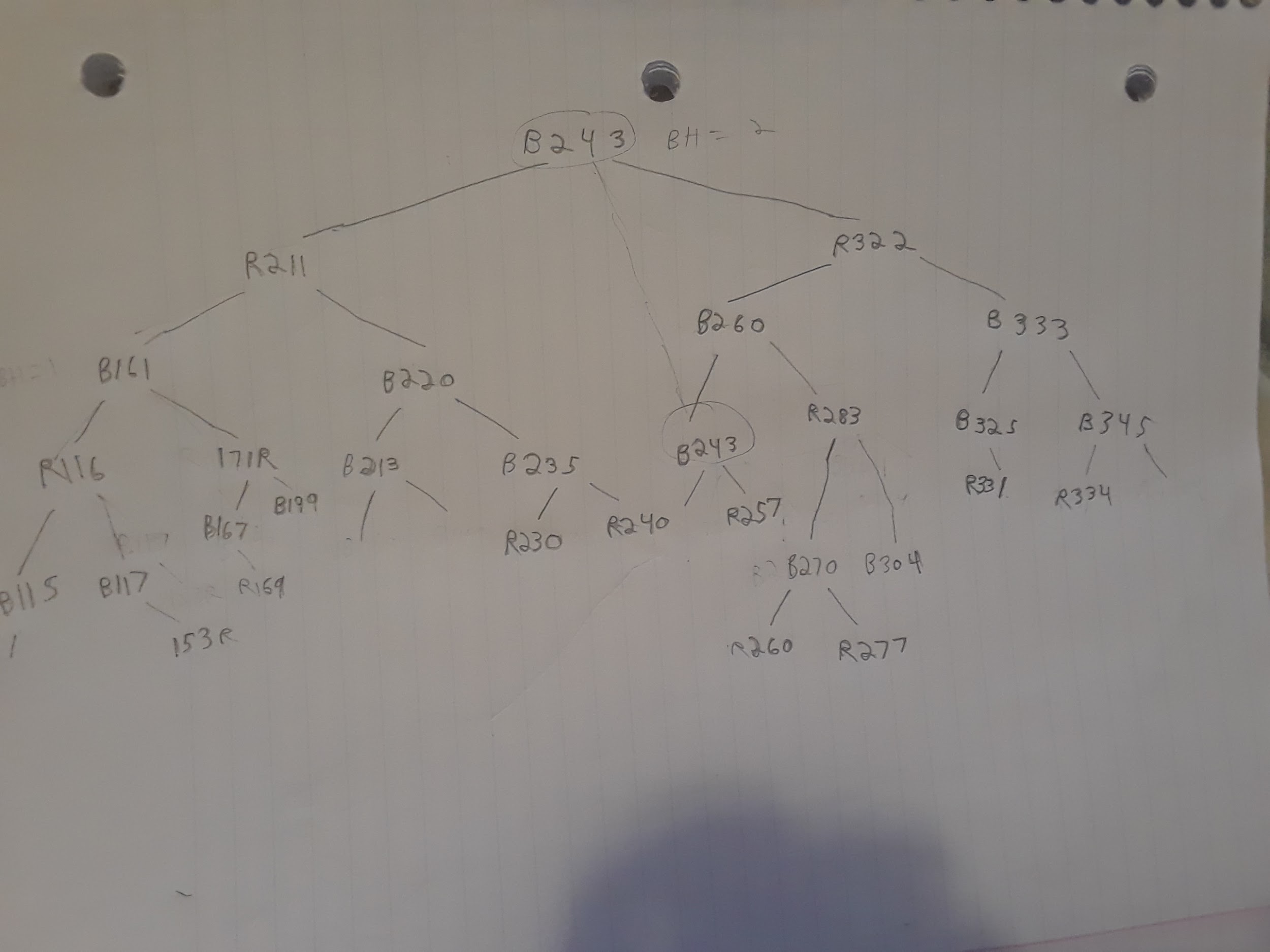
230, 333, 243, 345, 304, 220, 161, 167, 240, 322, 334, 115, 211, 213, 325, 235, 117,

260, 116, 243, 283, 171, 257, 277, 199, 270, 260, 153, 169, 331

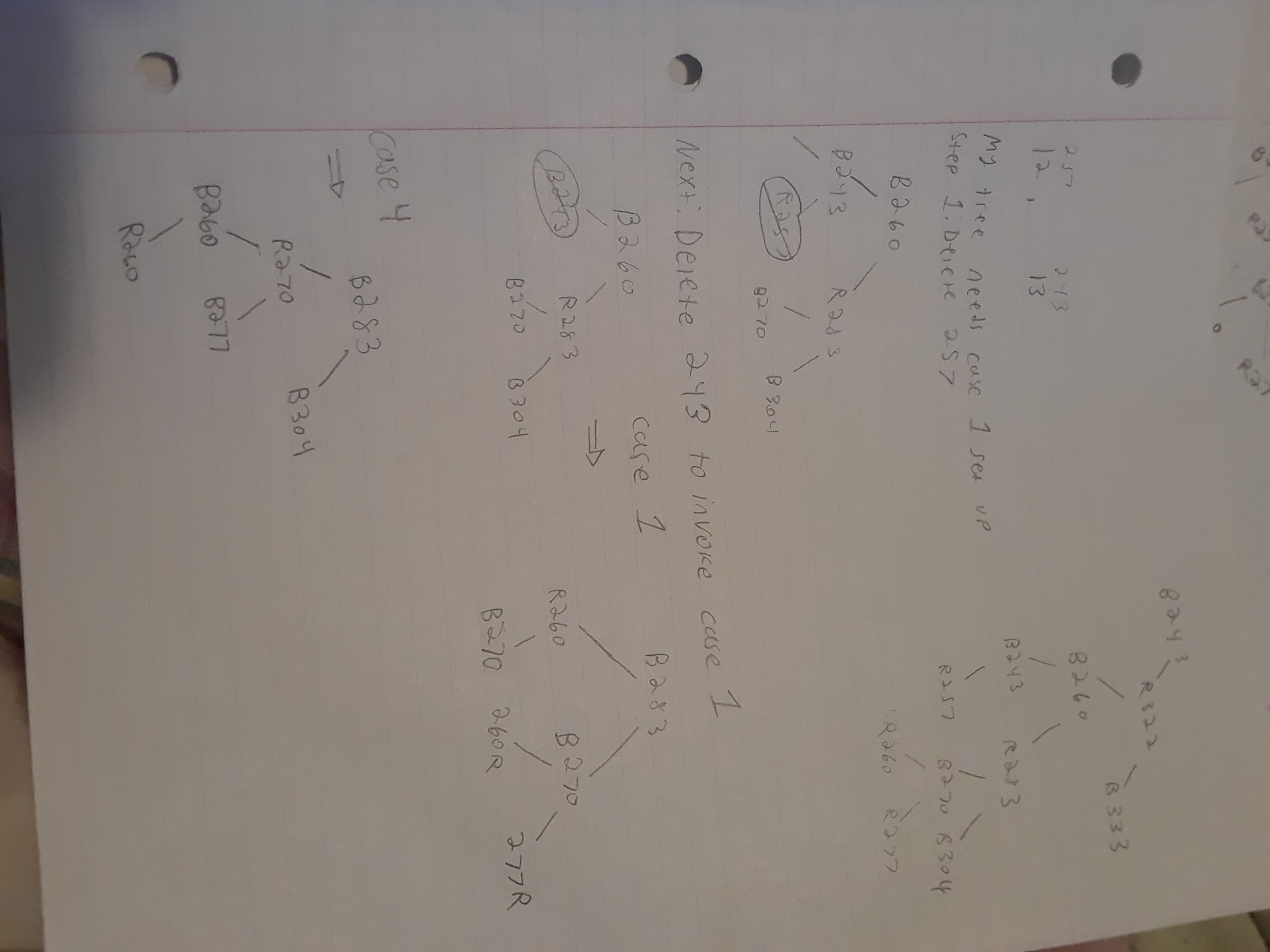
This was done by making a hidden command “!index”, which gives me the scores of all 30 urls after a search. I took those scores and made test classes called “program2” and “main2” for manipulating those inputs. Using the same inputs means I would get the same outputs every time, which made debugging and testing and drawing easier when dealing with 30 nodes.

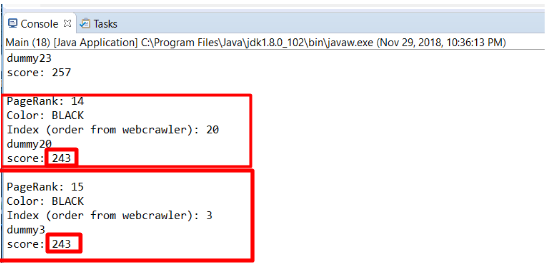
I called insert on the 30 scores and gave them all dummy names. The fact that it was able to insert 30 urls to the tree and still have RB tree property 5 maintained probably means my insert works. This is how the tree looks. I drew it with the help of the debugger, which let me peek at the variables within UrlNode objects inside the tree. So this doubles as testing.

**From here on out, this tree will be the tree I use to test add and delete. When drawing add and delete, I did not draw the entire tree of 30 nodes, only the subtree manipulated, to save time**

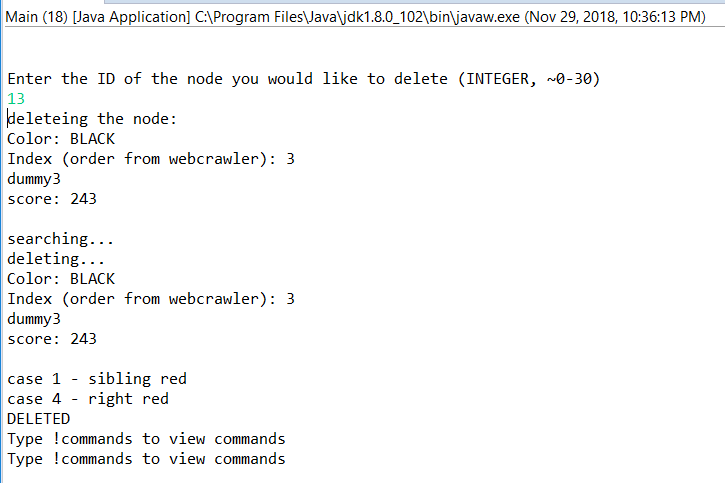


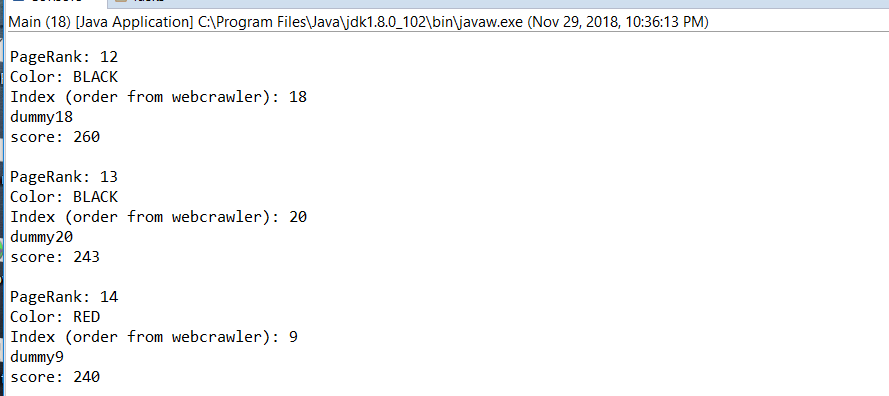
Set up case 1 in my tree by deleting 257. Then delete 243 to invoke case 1.



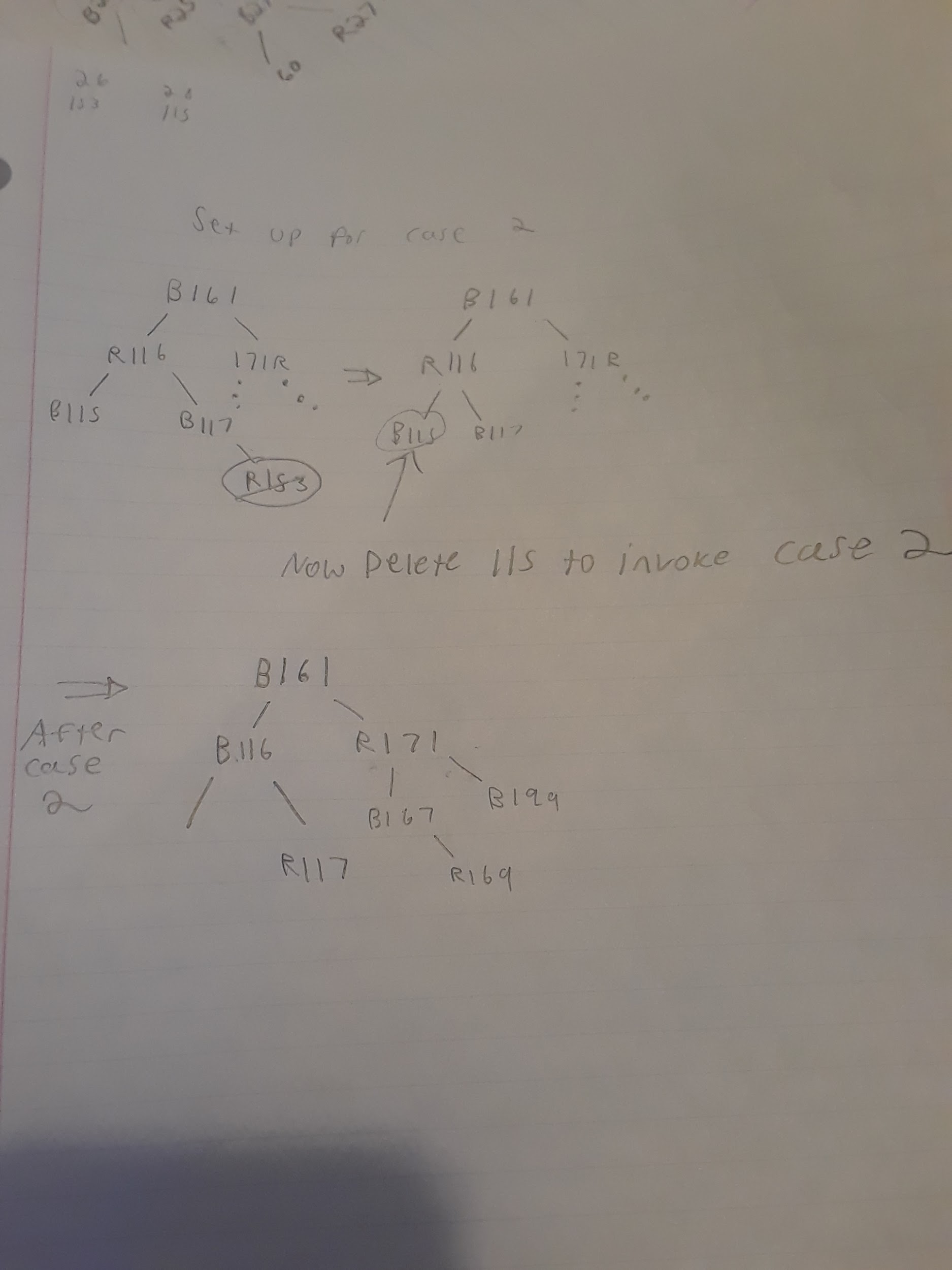


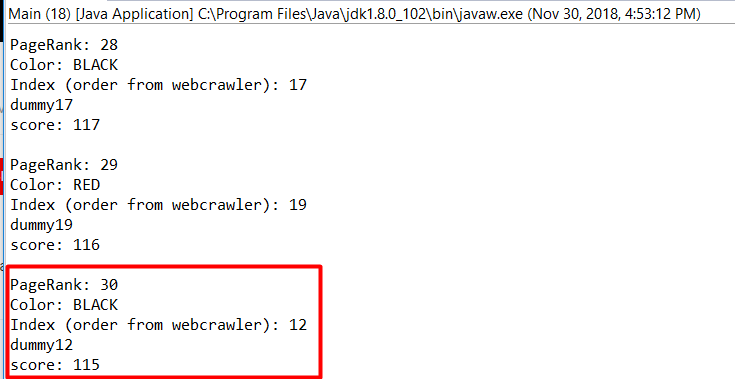
Delete 257, then delete the correct 243

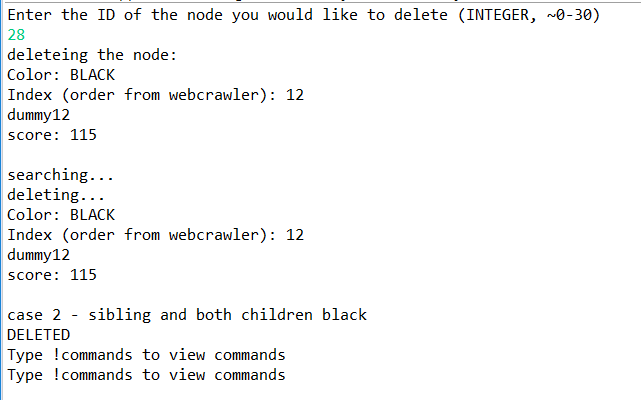


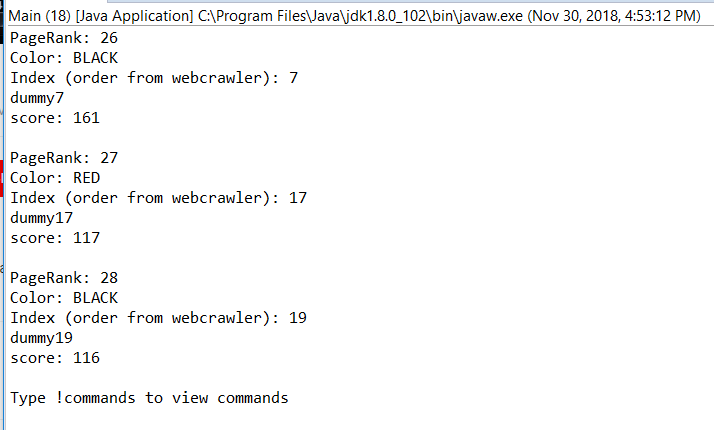


To invoke case 2, first we delete 153, then we delete 115.

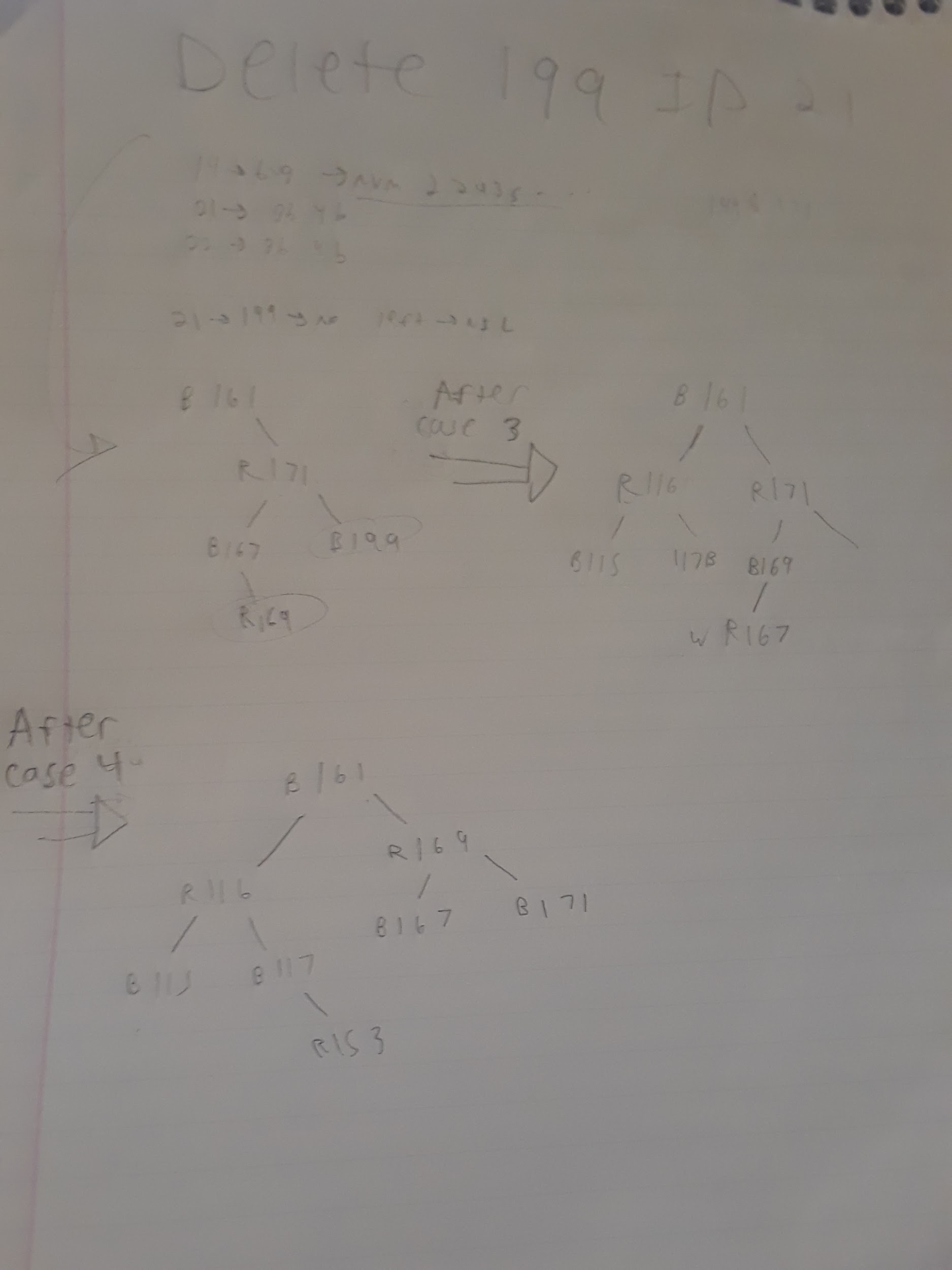




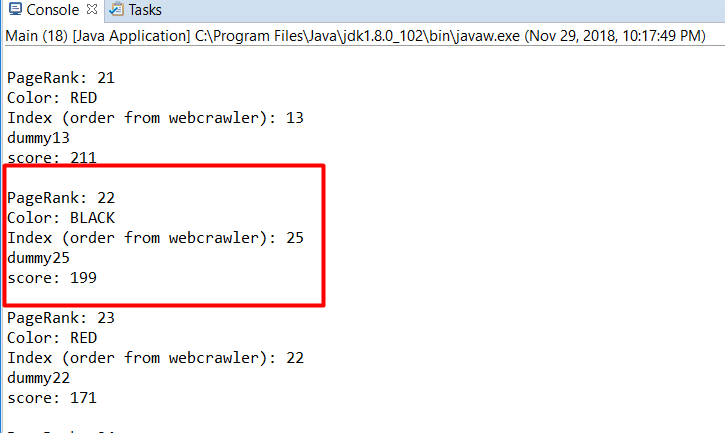


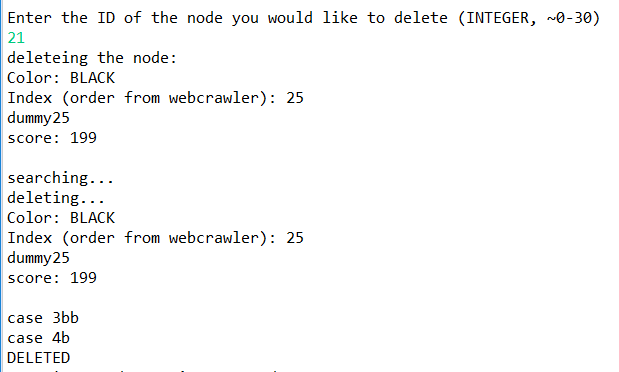


Deleting 199 to invoke case 3 and then case 4.

****

Before delete

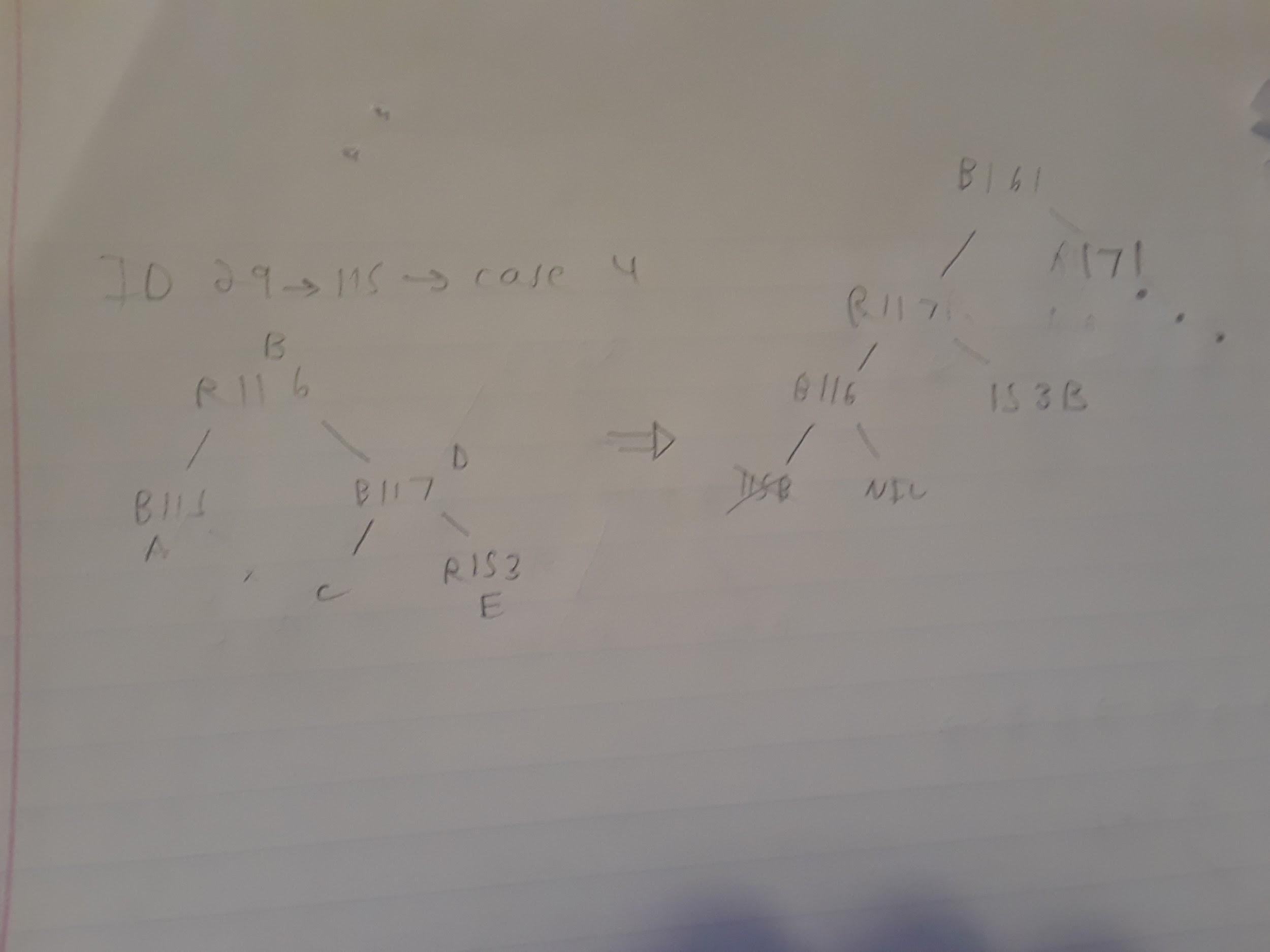
****

****

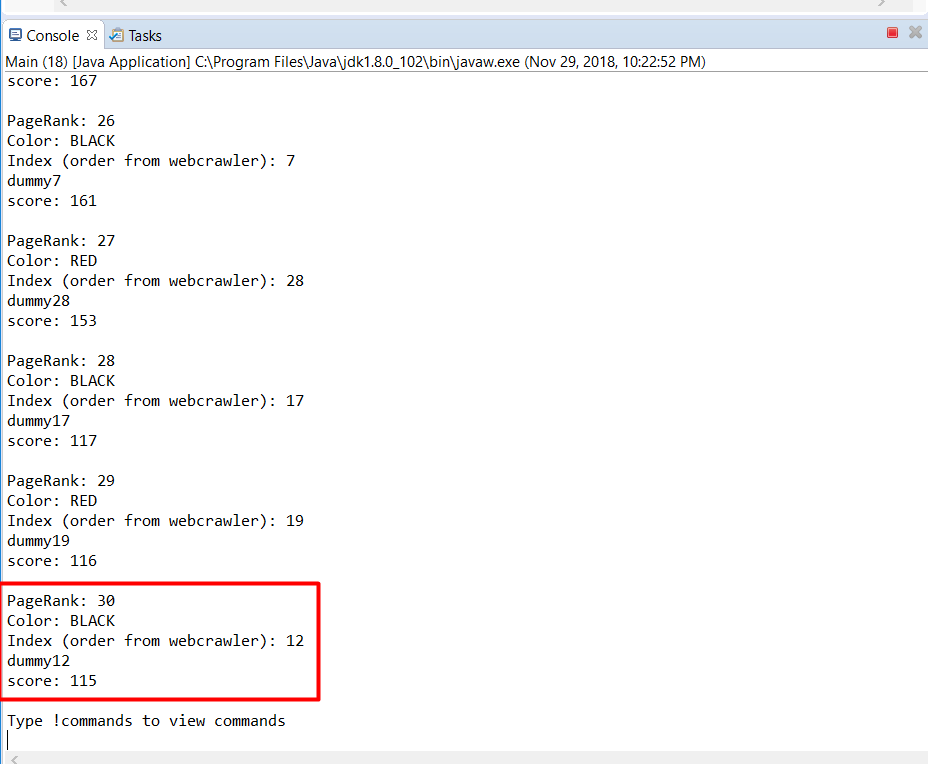
After delete

****

Deleting 115 to invoke case 4

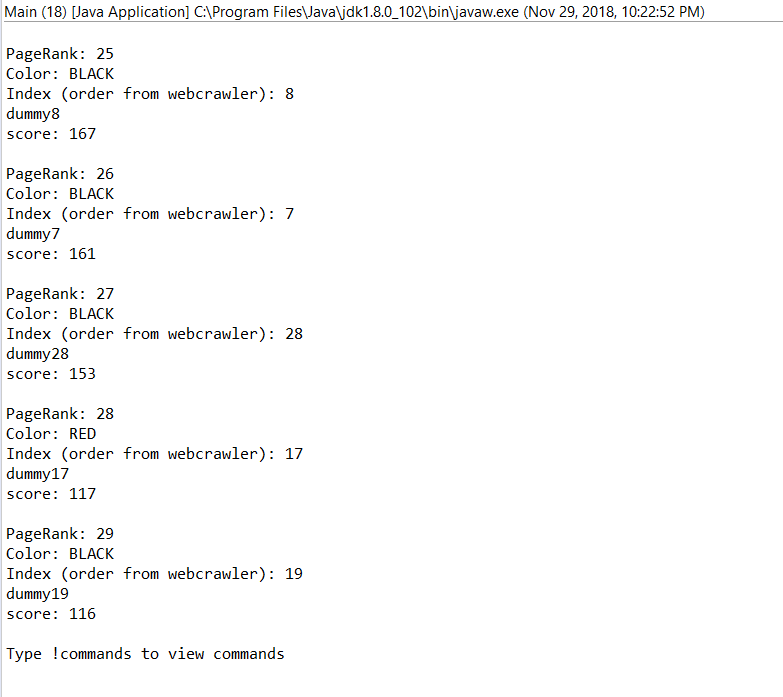
****

Before delete

****

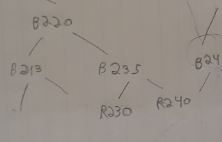
****

After delete

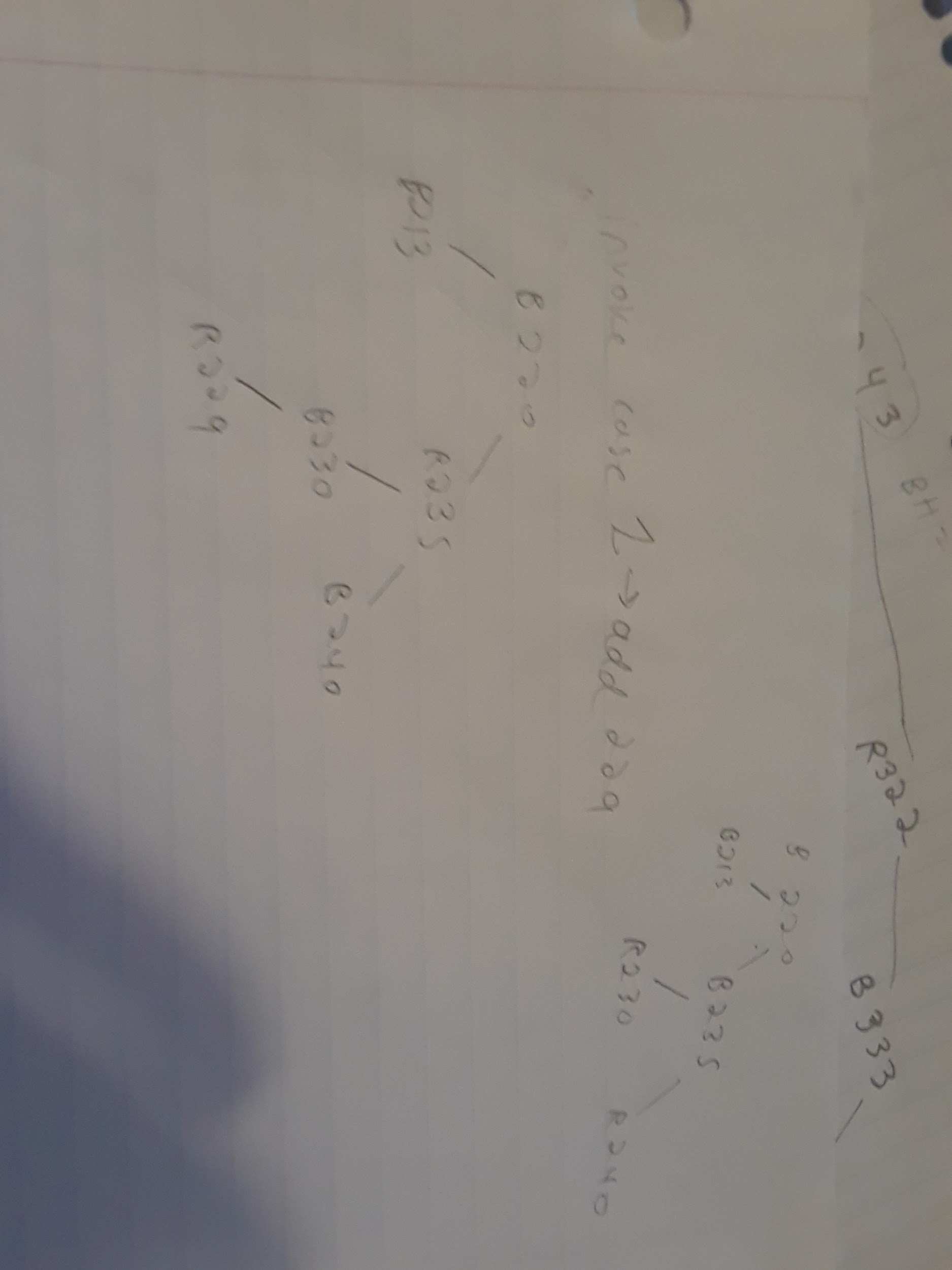
****

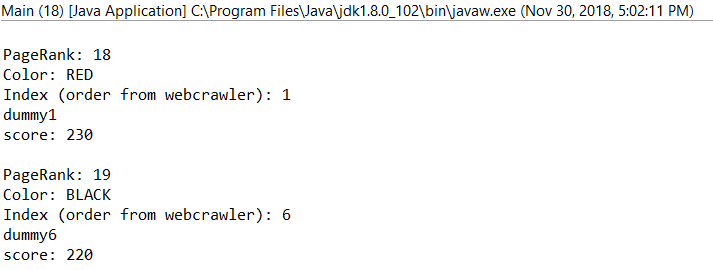
Adding 229 to invoke case 1

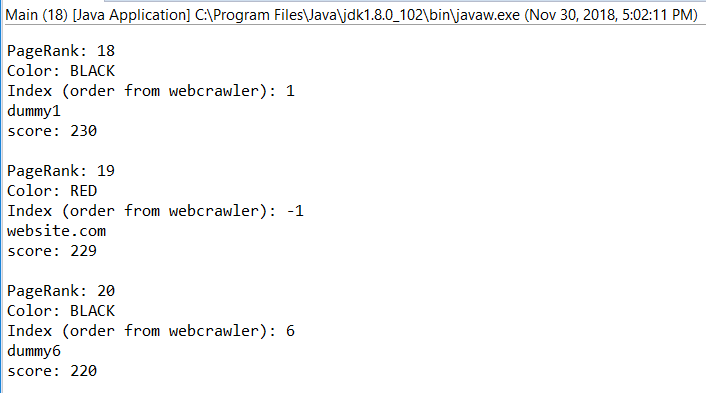
Before:

: 

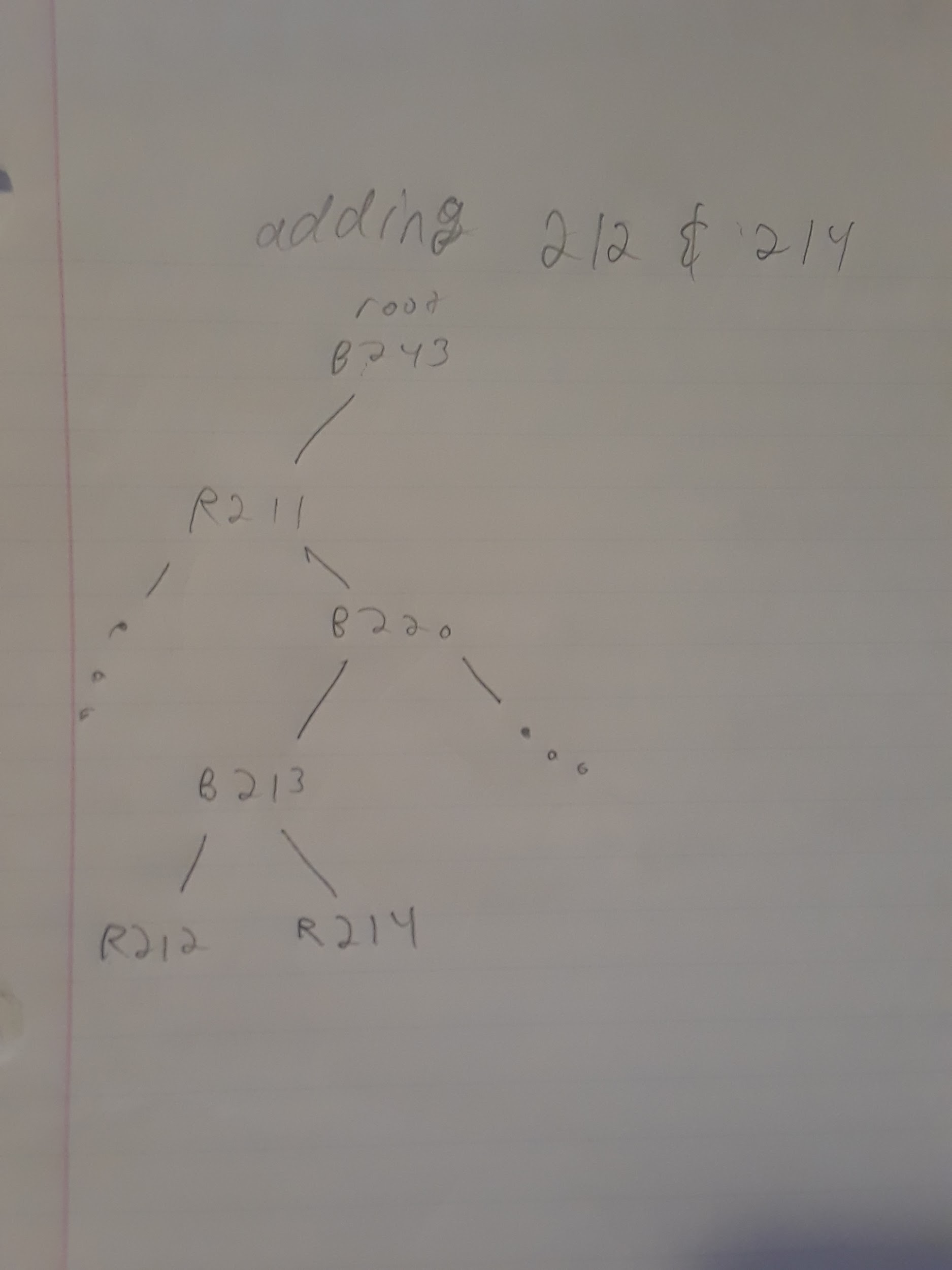
After:

****

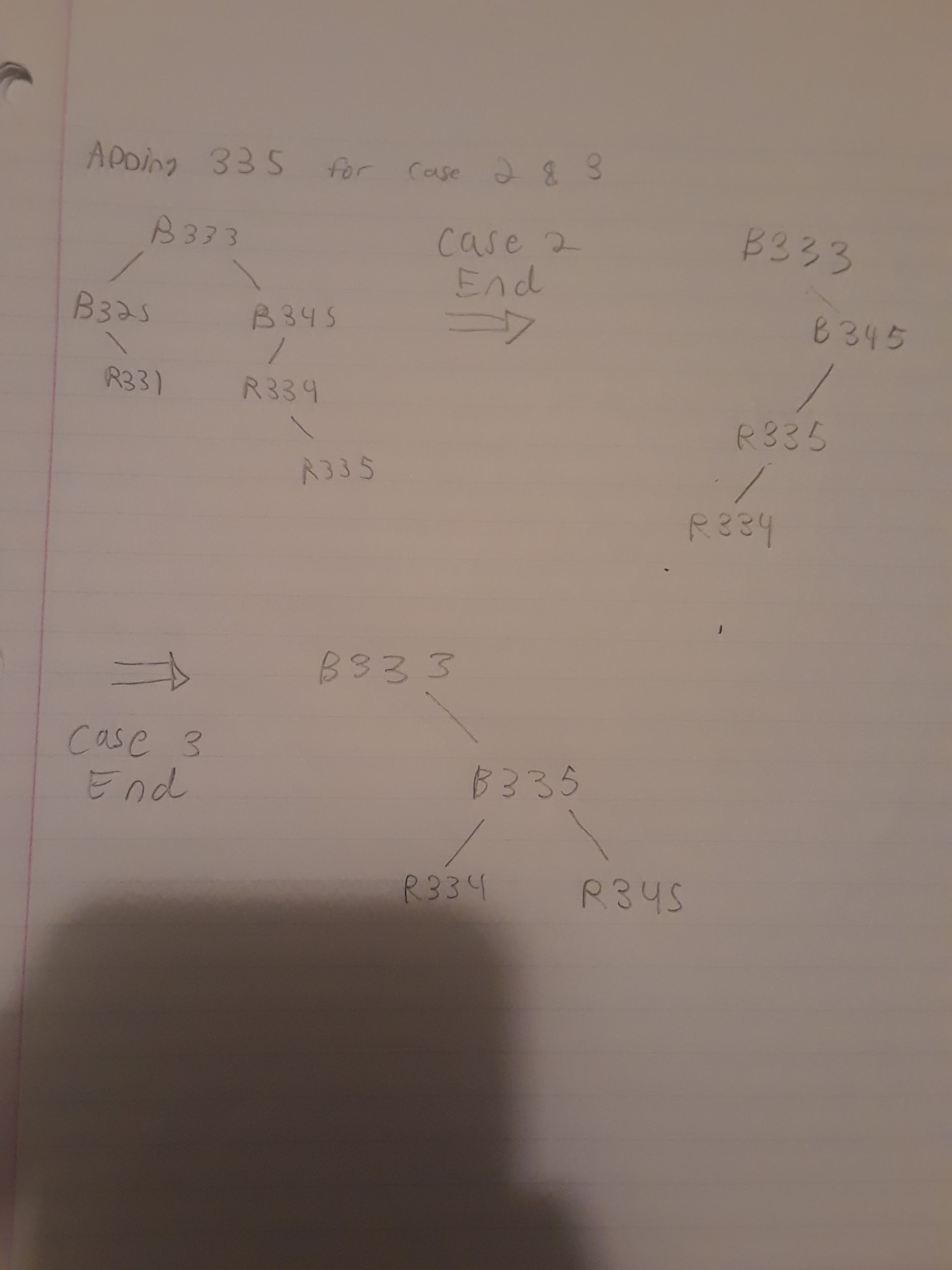
****

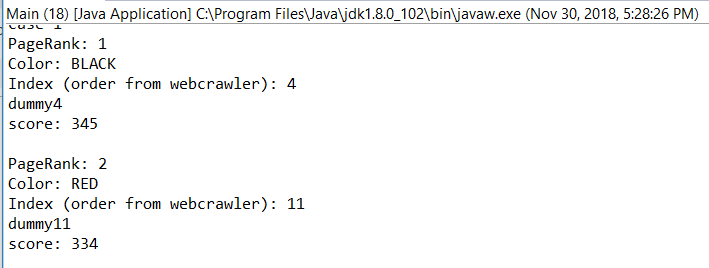
****

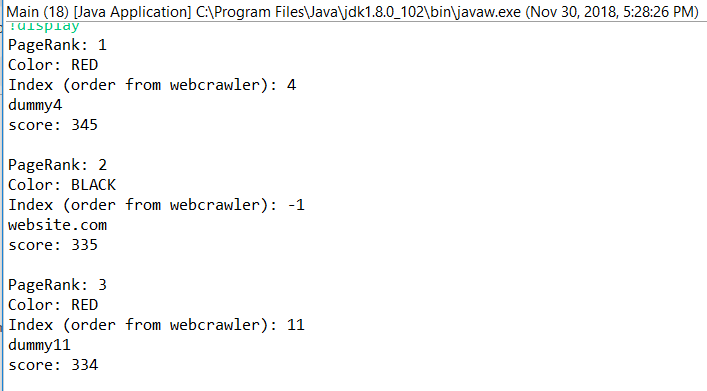
I don’t think much happens if you add a 212 and 214, because the parent is black and the chlidren are red

****

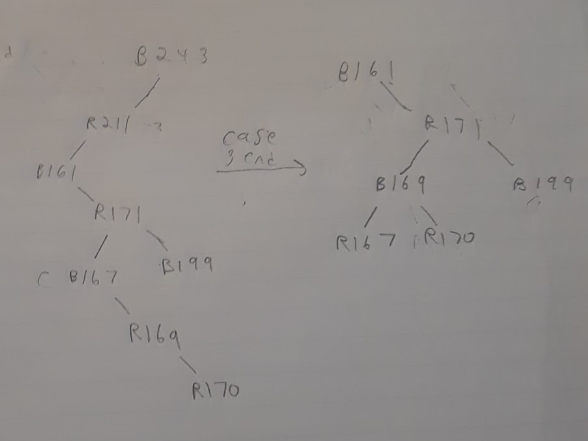
Invoking case 2 then 3 by adding 335

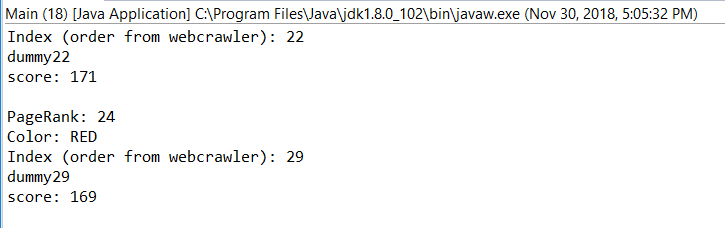


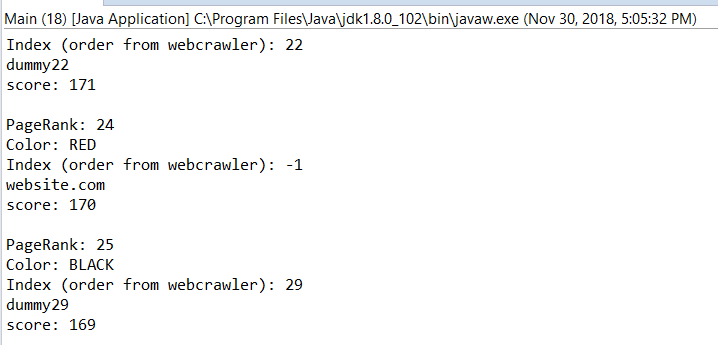




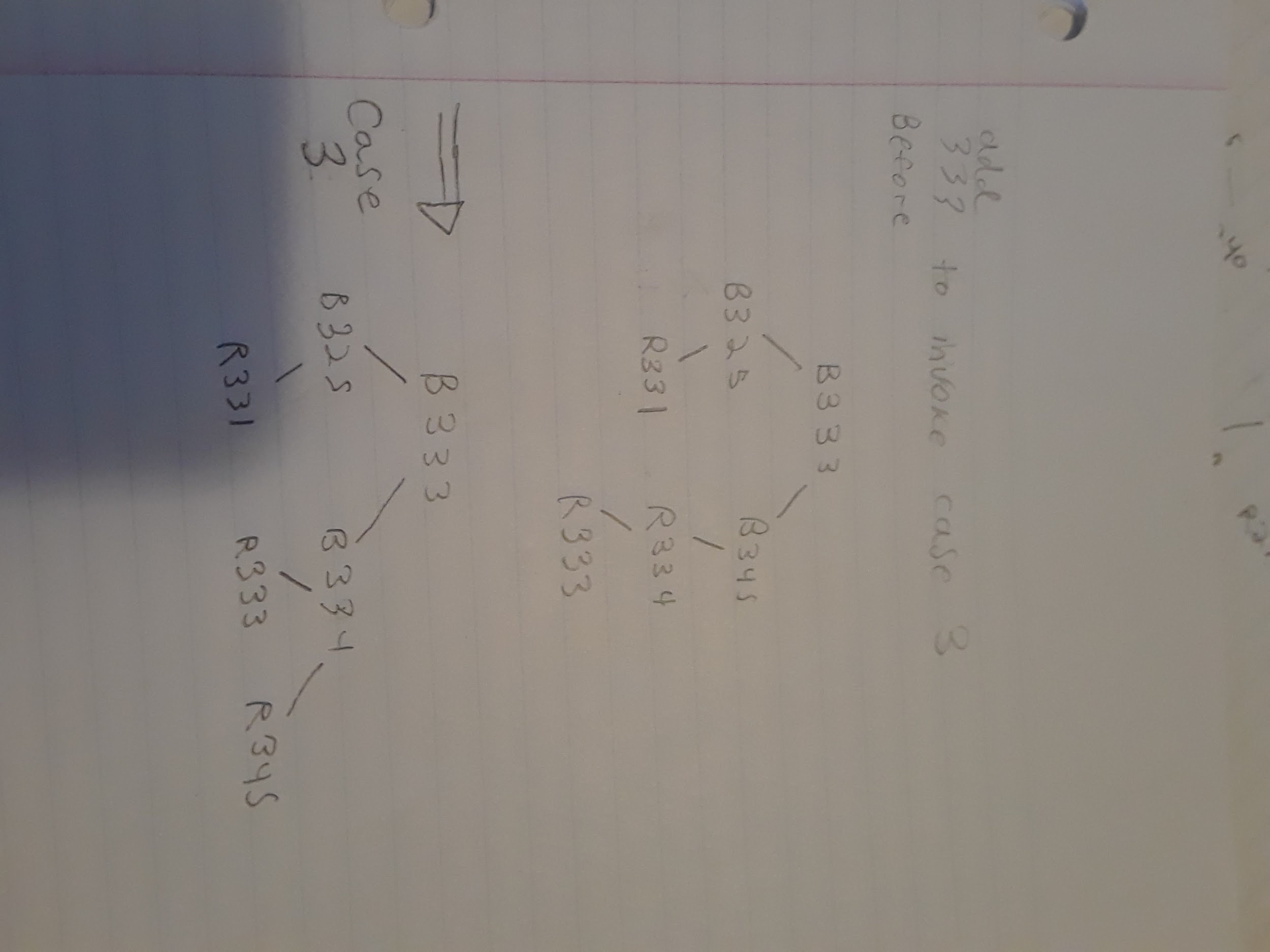
Adding 170 to invoke case 3

****

****

****

Another case 3 by adding 333

****

**HOW IT WORKS**

Spider and SpiderLegs are the webcrawler. They return a list of URLs. Program class takes that list of URLs , constructs node objects of them with scores, and adds them to an instance of tree. Program class also handles all the user input and manipulates the tree based off of the user input.

**Url class**

I needed a way to store urls and their scores together. So I put it into one object. The 4 variables that are added up to calculate the score are random.

**UrlNode class**

A normal node with left, right, and parent, with a url object as the key. I made this class because trees can only manipulate nodes that have left, right, and parent pointers. It also has a String color to identify the color.

**UrlTree class**

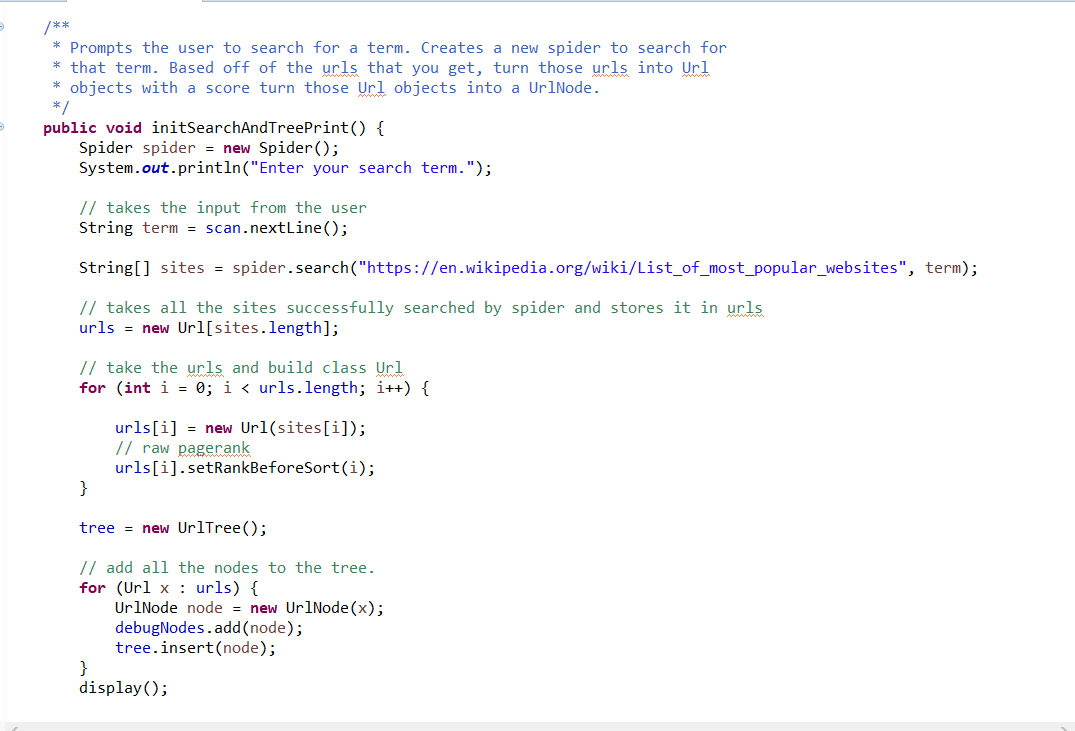
It Manipulates UrlNode objects, which hold Urls, which hold the website url its score. Therefore, this is my way of manipulating the results and changing their scores / deleting / adding them. The only notable change I made was by adding searchByNode and deleteByName. These two methods exist because the tree is sorted based off of score, but if two websites had the same score, I need a way to delete them based off of name.

**Program2 and main2 are for debugging.** I used main2 and program2 to manipulate the nodes I used to draw the tree.

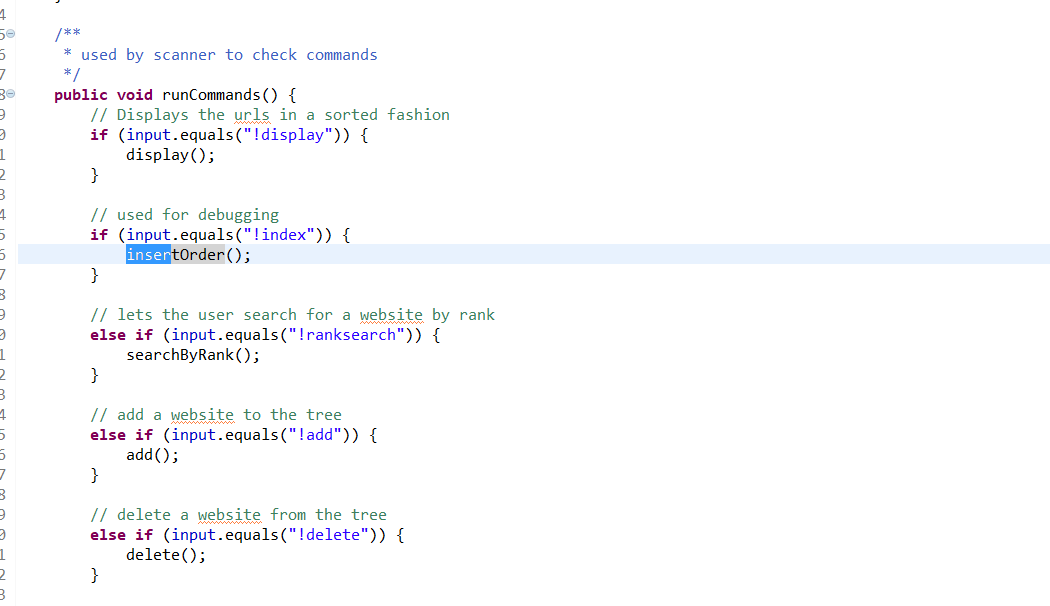
**Program class** → Explained in previous report



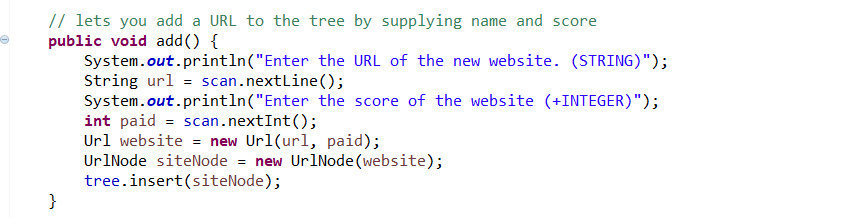
^I use while(true) loops so that the user can run several commands in succession without breaking out of the program



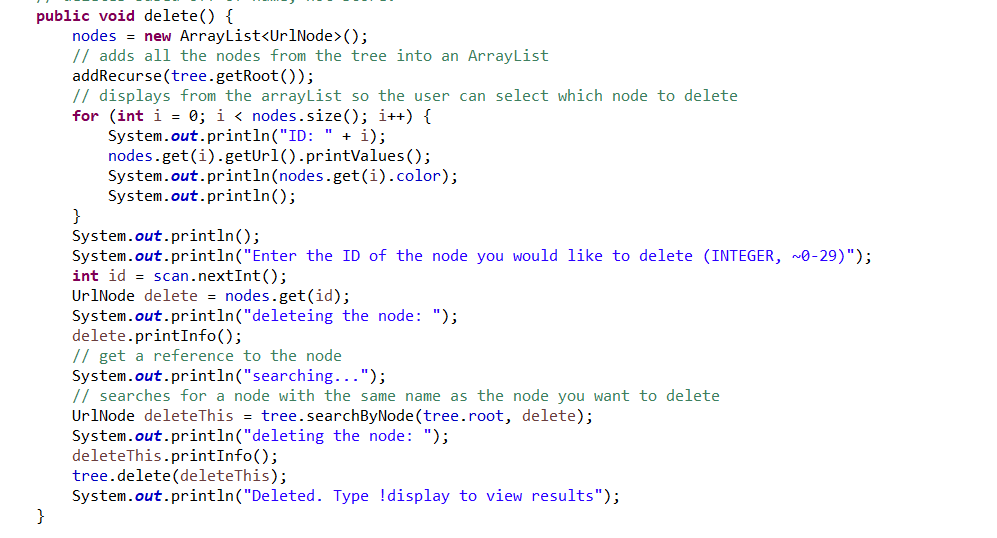
^Prompts the user to search when the program starts. From all the searched terms, creates Url objects with a score randomly assigned. With the Url Objects, creates UrlNode with a color variable and adds that node to the tree with the tree.insert() function.



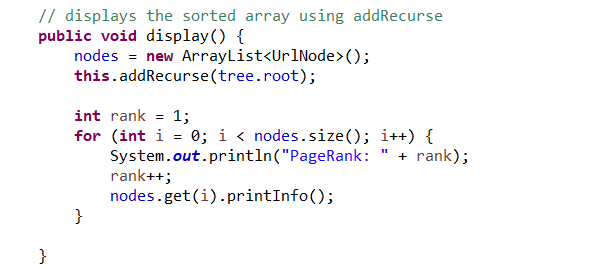
^This large block of text checks what the user entered into the console. Depending on their input, will run a function tied to that input



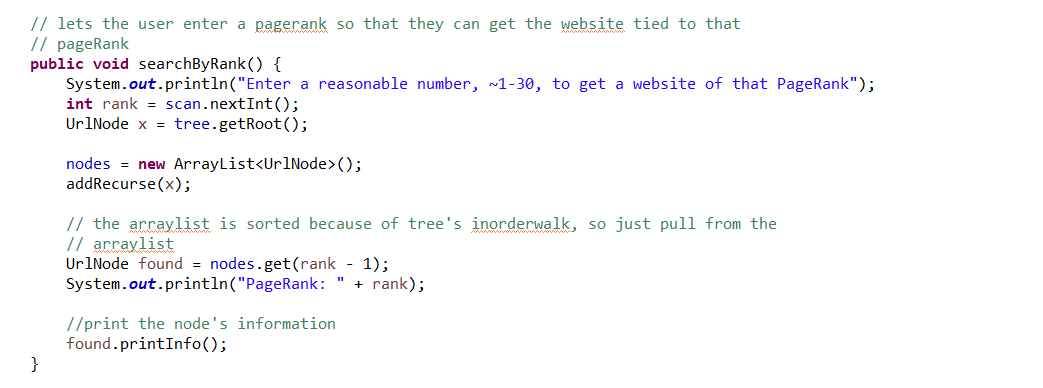
^ Speaks for itself, asks the user to enter the name and score of a website they want to add, creates a node from that information and inserts that node into the tree.



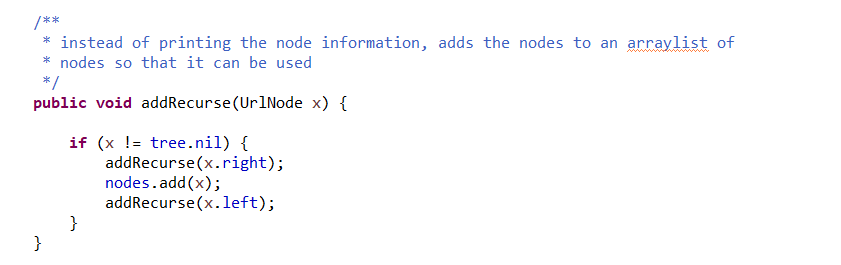
^A bit more complicated. For all the nodes in the tree, add that node to an arraylist with addRecurse() function. Before calling addRecurse, clear the arraylist of nodes that addRecurse() deals with. Print all the nodes from that arraylist so that the user can select which node they want to delete. Search from the node they pull for the arraylist in the tree, it has to exist in the tree because all the nodes from the arraylist were just from the tree. After it finds that node, delete that node.



^shows all the nodes in the tree in order with their PageRanks. Again, addRecurse adds all the nodes from the tree to an arraylist that is outside the scope of the method we are calling



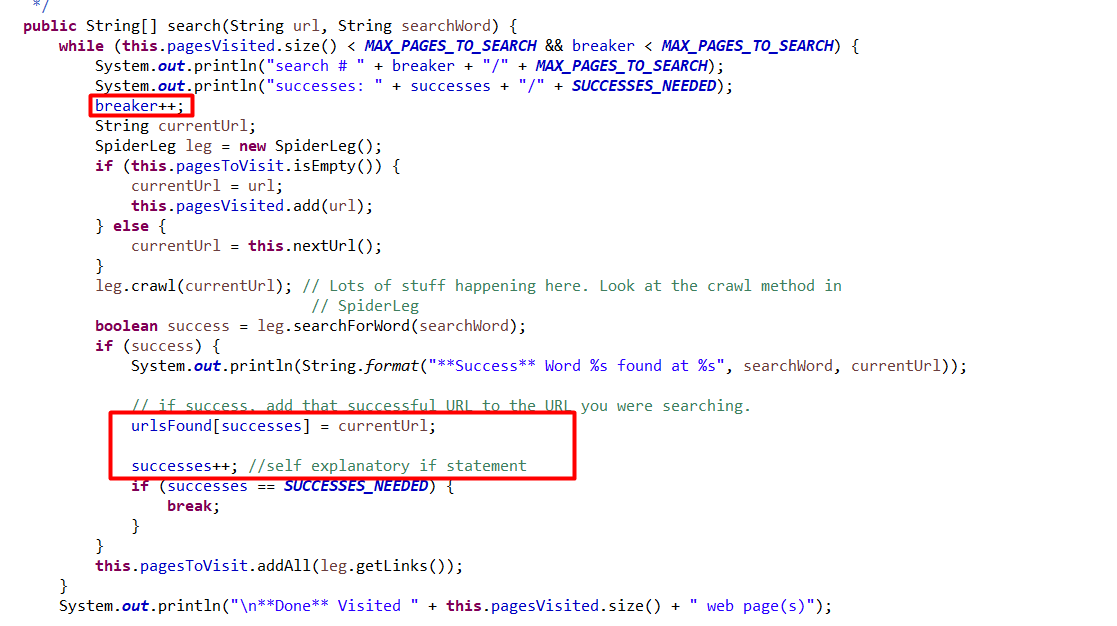
^Allows the user to search for a node based off of its pagerank. Addrecurse adds the nodes to the ArrayList of nodes in their pagerank order, so we can just pull from that ArrayList index - 1, since arrays start at 0.



^There is an arraylist of nodes outside of the scope of this method. Everytime this method is called, caller has to remember to clear that ArrayList of nodes. Essentially helps me get all the nodes in the tree so I can manipulate them.

**Spider & Spiderleg class (Taken from my previous reports)**

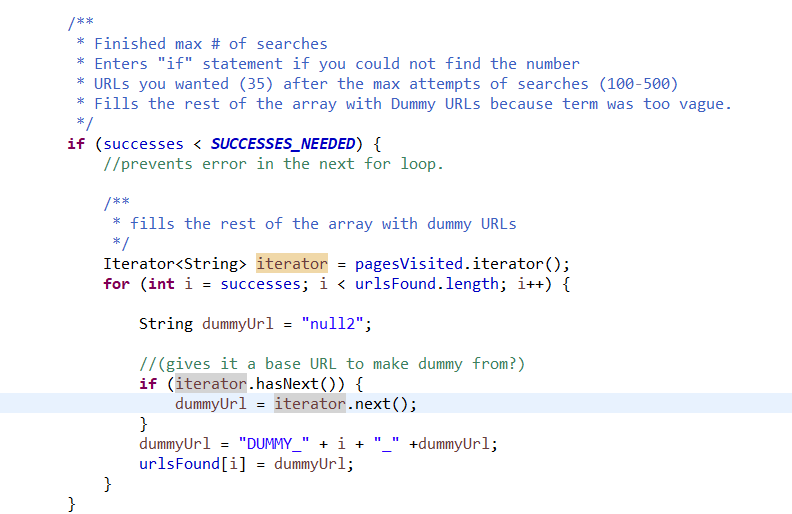
Imagine you entered a term. For the root website, it searches a wikipedia page and all the websites linked by it for that term. If the term is found on the website it is on, the website’s URL is added to an array of Strings[]. If there are 100 searches (value can be changed by going into the code) and the array of successful Urls is not full, then it fills the remaining slots in the array with dummy Urls, pulled from the list of unsuccessful searches. I chose to do this instead of having it run forever because it saves time.



^ I don’t know what most of the Spider code does, but I realized that the person who coded it has an if(success) statement that applies when the term is found on the website. So I decided to take advantage of that and instead of just printing the currentUrl, add it to a list of successes and increment a “success” variable to keep track of the number of the number of successes. Once it reaches the number of successes it needs, it leaves the loop.



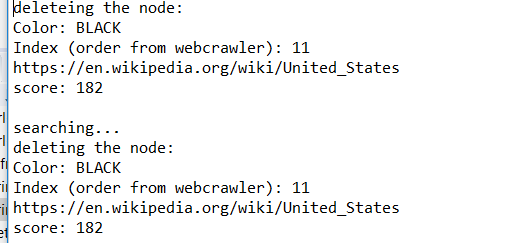
^ The “breaker” variable is to keep track of the number of websites searched. That’s why it ends at 100 and not into infinity.



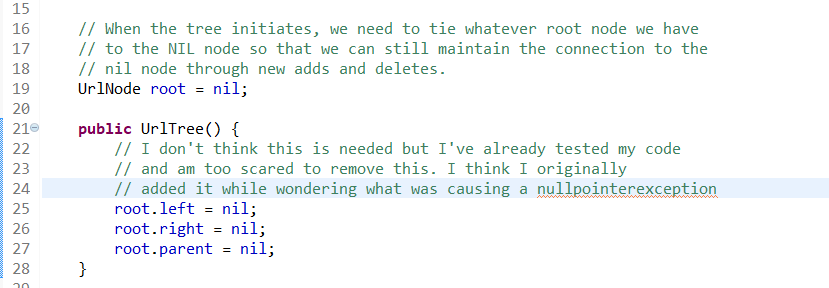
^ If the array of successes is not full, add dummy URLs to fill up the rest of the array. I get the dummy URLs from the list of unsuccessful websites. Dummyurl is initiated as “null2” so that if a website URL is named null2, I have an idea of where the error is from.

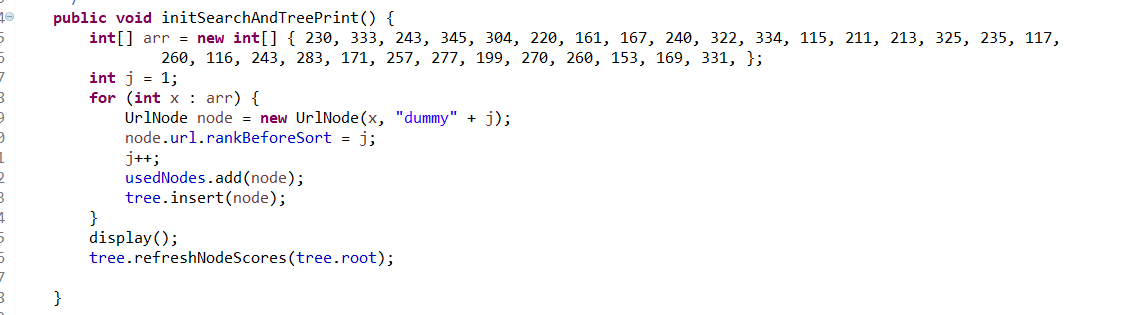
**PROBLEMS ENCOUNTERED**

* The tree searches for which node to delete by name because the probability that two nodes have the name name is less than the probability that two nodes have the same score. I made the program print out which node you want to delete and which node is found just in case two nodes happen to have the same name, you can realize why it deleted the wrong node.



* I had to write my own right rotate because I could not find it in the book. That led to a bug that took me a while to debug.
* I needed a way to implement the NIL node and tie that NIL node to the root so that insert could work properly. That led to this

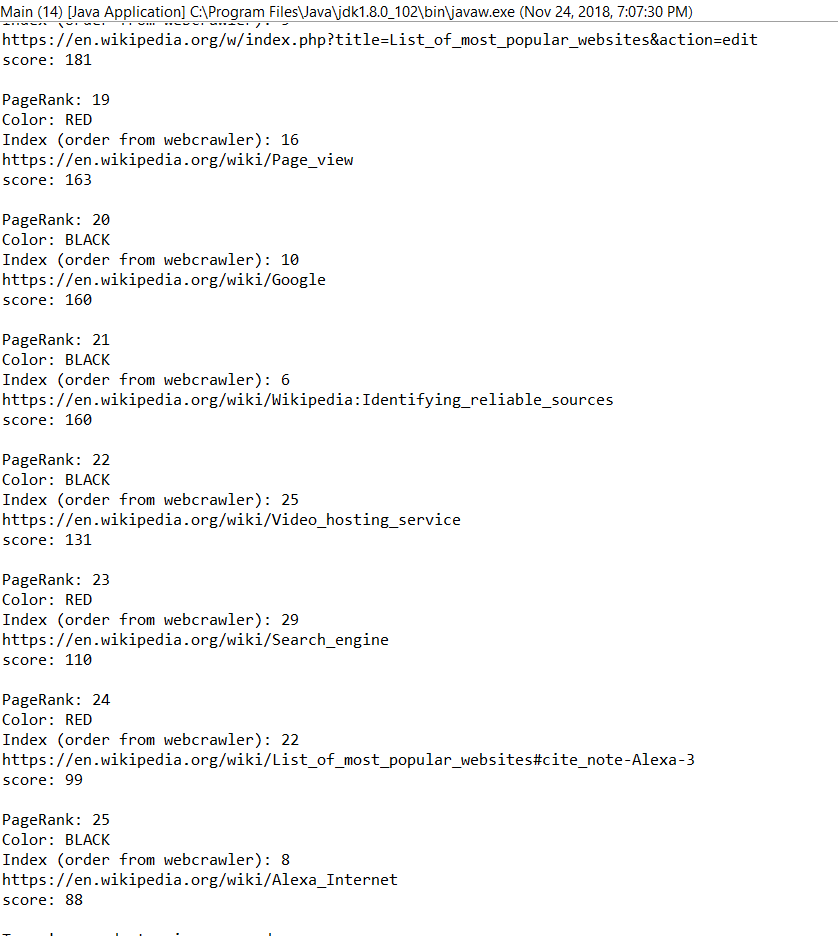


* After more debugging, I realized that my infinite loops were happening because I had to change all “null” variables to “nil”
* In the textbook’s psuedocode, I thought else if meant else if(), not else{ if() }. It took a lot of debugging while I was wondering if jumping from case 1 to 3 was really necessary
* It was hard to debug straight from my web crawler because the scores are random every search. This is why I made the “program2” and “main2” class. To get random inputs for that array, I added the command “!index”, which lets me pull the scores so I can insert them into an array. 

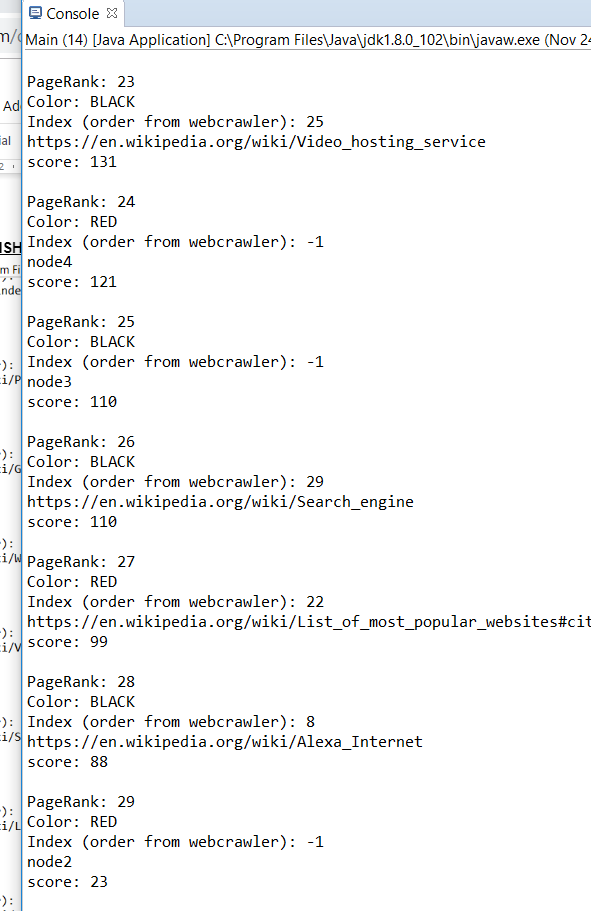
**LESSONS LEARNED**

* I gained some experience in debugging RB trees. Although I did not fully understand the theory behind the code, I was able to get by by asking myself what the code should do and look like before and after each case, as well as which casees should be applied.
* Left-rotate rotates strictly by rotating the pointers, not the information inside the nodes. I only realized that after writing right-rotate
* While testing the code, I kind of saw how the cases flowed. I think, for example, in insertFixup, if case 1 happens, then case 2 and 3 will not. But if case 2 happens, case 3 will happen.
* One way of implementing nil into a RB tree is by initializing the root as nil and have its left, right, and parent pointers all point to itself. The insert and delete methods will still work in building the tree.
* When I first saw the slides, I was wondering what would happen if in the fixups, z or z’s parent was a right child instead of a left. I realized while programming that the entire second else{} statement is for those cases.

**MISCELLANEOUS SCREENSHOTS**

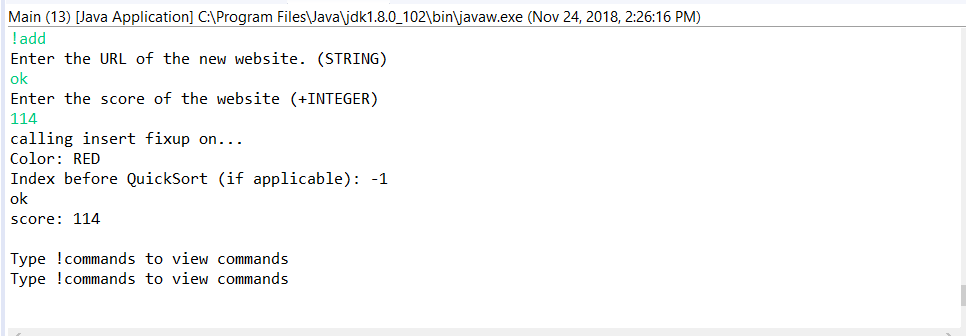
****

^deleting 5 nodes from the tree



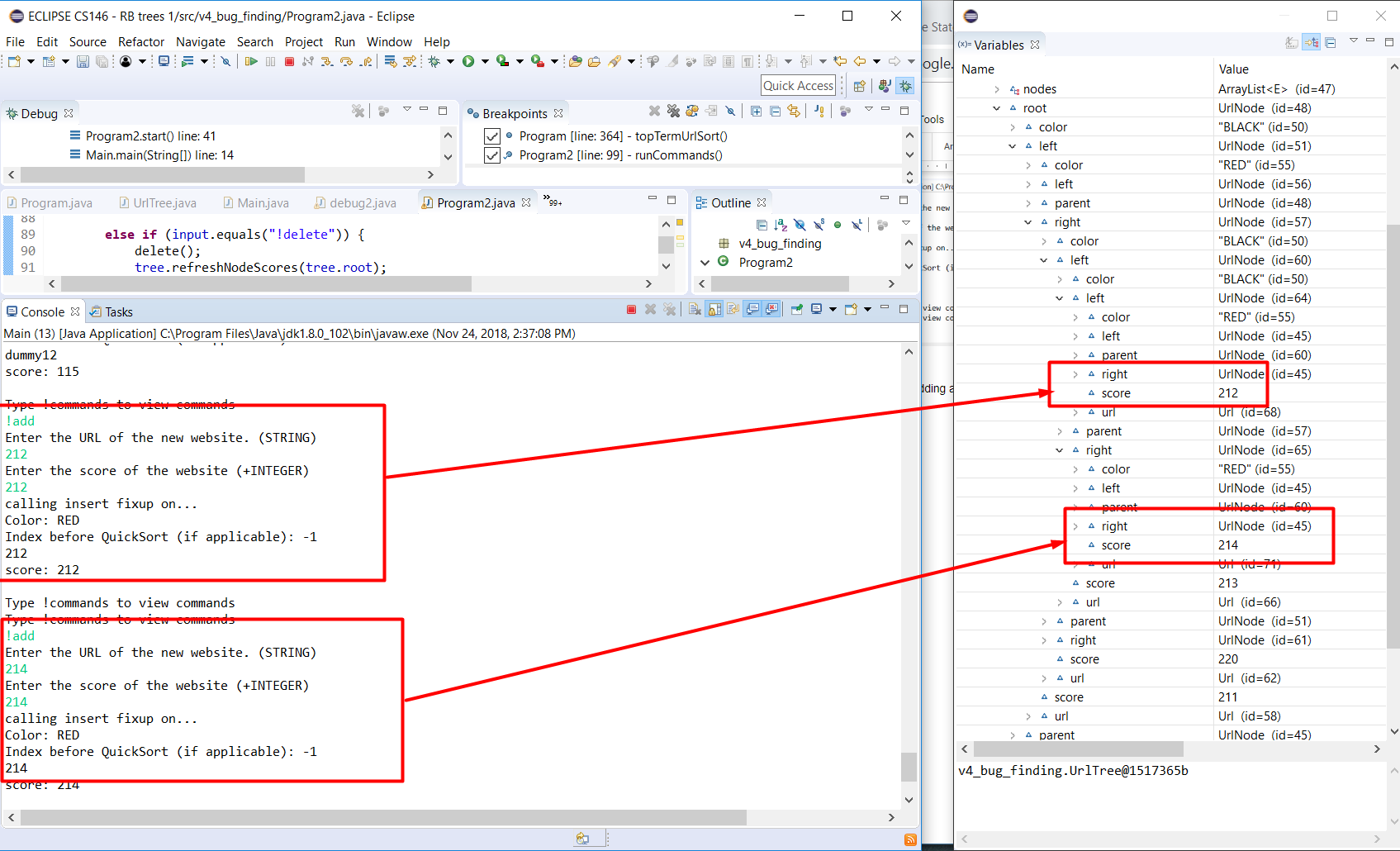
^adding nodes back to the tree

Adding a red node to a black parent will not do much in invoking cases

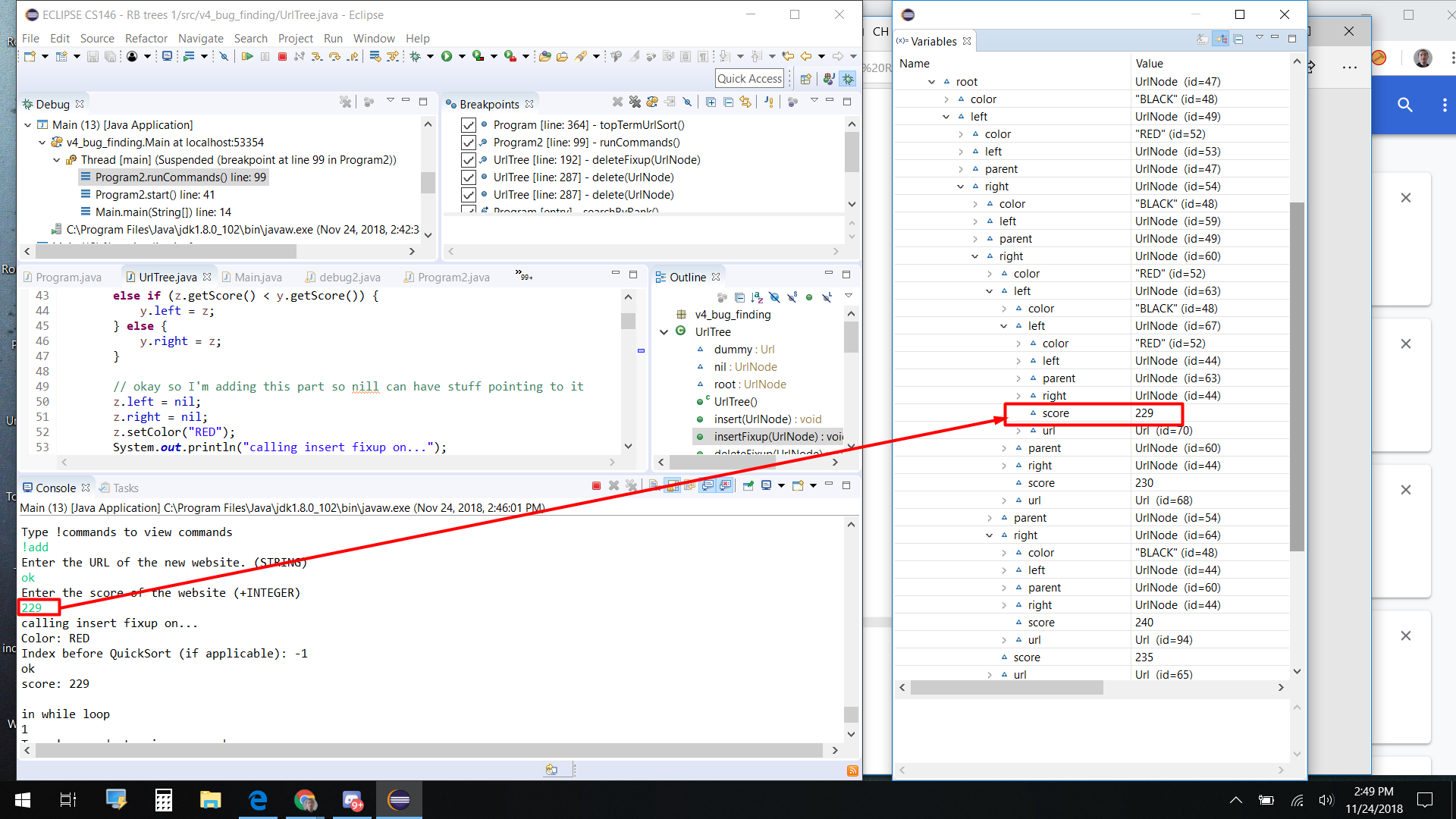


I think if you’re adding a red node and parent is black the while loop will do nothing

Simple add 212 and 214 under 213. None of cases invoked.



Adding 229 and invoking case 1



Adding 170 to invoke case 2 and 3

