**Programming Assignment 2**

**Simulator of Google Search**

**Engine Results Page (SERP) using Quick Sort and Binary Search Tree**

CS 146 – Intro to Algorithm and Data Structure

Design by Linjun Cao

**Table of Contents**

1. Explain Google Search Engine Simulator design and implementation details --------------------------------- 3
2. Explanation of the Classes/Functions
3. Classes ---------------------------------------- 5
4. Functions --------------------------------------- 17
5. The procedure of how to unzip files, install application, and run/test codes --------------------------------------- 23
6. Problems encountered during the implementation ---- 25
7. Lessons Learned --------------------------------------- 26

**Part I: Explain Google Search Engine Simulator design and implementation details**

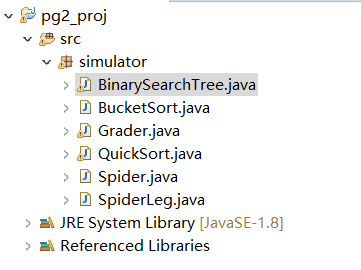
Programming-2 is based on Programming-1. So, the part of web crawler will remain the same as the previous one. The most challenging part is how to build a Binary Search Tree so that we can manipulate the data from the URLs by the tree. First, I work on the part of Quick Sort. All the work is from previous homework, so I can use them directly. One thing that should take care is the quick sort function is quicksort(array, index 1, index 2). While the heap sort function is heapsort(array) directly.

Then I work on the part of Binary Search Tree. First, I create Node Class for all the details. It need to contain index, total score, page rank and URL. I choose total score as the key, because if we are sorting by the total score, we can also get the page rank easily. One thing that we should take care is class Node cannot be public. It is ‘inside’ the class Binary Search Tree. Next, I work on the part which is simple and definite. It is implementing the pseudocodes from the textbook to the program. Just read them, understand them and implement them directly. But notice the page rank. It cannot be implemented directly because it is the result after we sort all the nodes(URLs) by their score. A nice way to deal with it is implement all 0 as their page rank first. After all nodes are inserted, we update their page rank. Then we meet with the second problem which node do we start to update. We can start from the first node (with the highest total score) or start from the last node (with the lowest total score). It is not good to start from the last node because we do not know how many nodes are there in the tree are. Maybe we know it is 30 at the beginning. But after we insert or delete more nodes in the tree, it will be changed. The best way is to start from the first node (with the highest total score). Because the page rank of the first node in the tree is always 1. And the second one is 2, etc. When we go through the entire tree from the first node (with the highest total score) to the last node (with the lowest total score), we can use the function Inorder-Tree-Walk(x) and make a little reverse. Inorder-Tree-Walk(x) go through the entire tree from the extreme left node (with the lowest total score) to the extreme right node (with the highest total score). We reverse this function to achieve our goal. Now we have all the functions of the Binary Search Tree. Just put the 30 URLs with their index, total score and string into the tree and we are all set.

**Part II: 2. Explanation of the Classes/Functions**

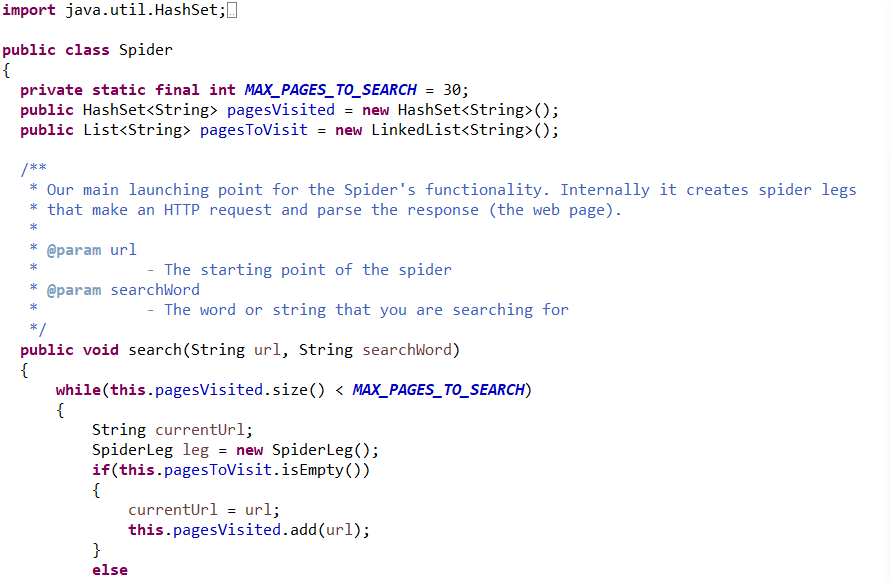
**Classes:**

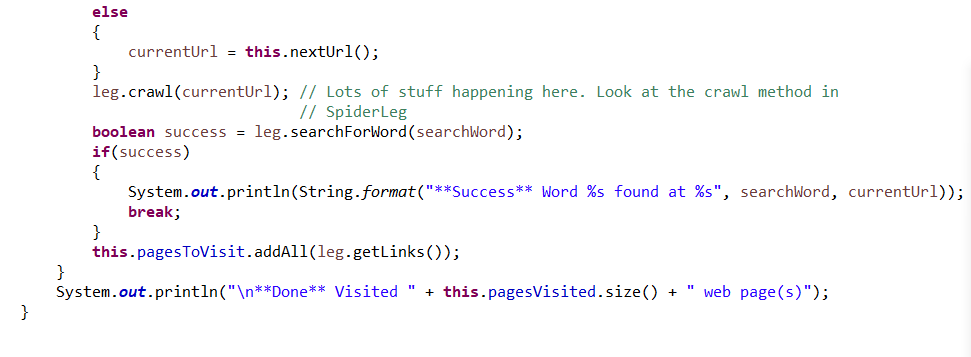
We have the following classes.

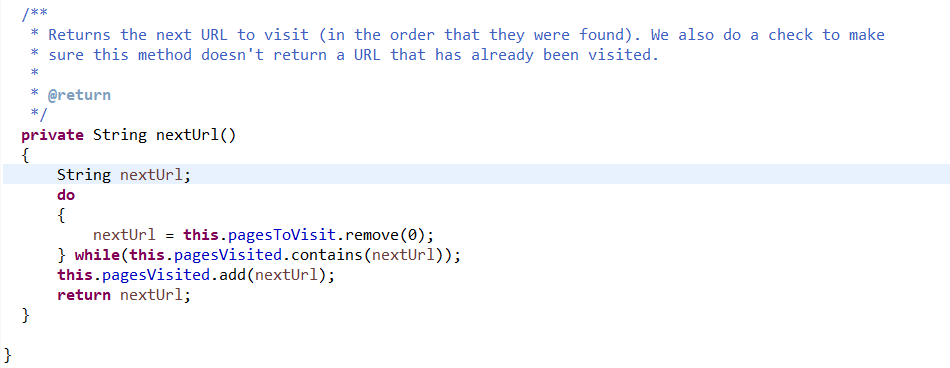


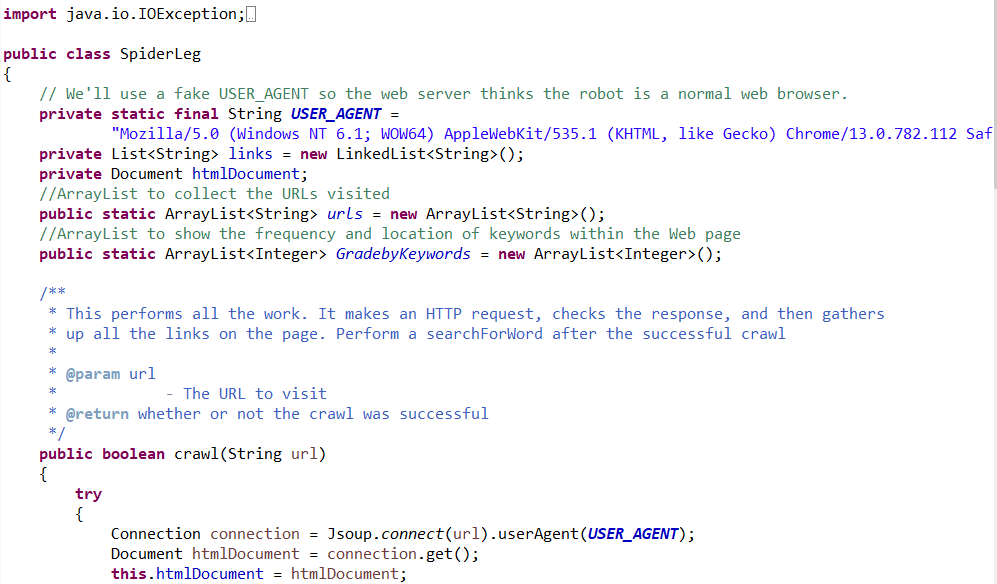
Class Spider and Class SpiderLeg are the part of web crawler. It ‘eats’ a link and a key word and give URLs which we need. An array is created to collect those URLs until it has already got 30 URLs.

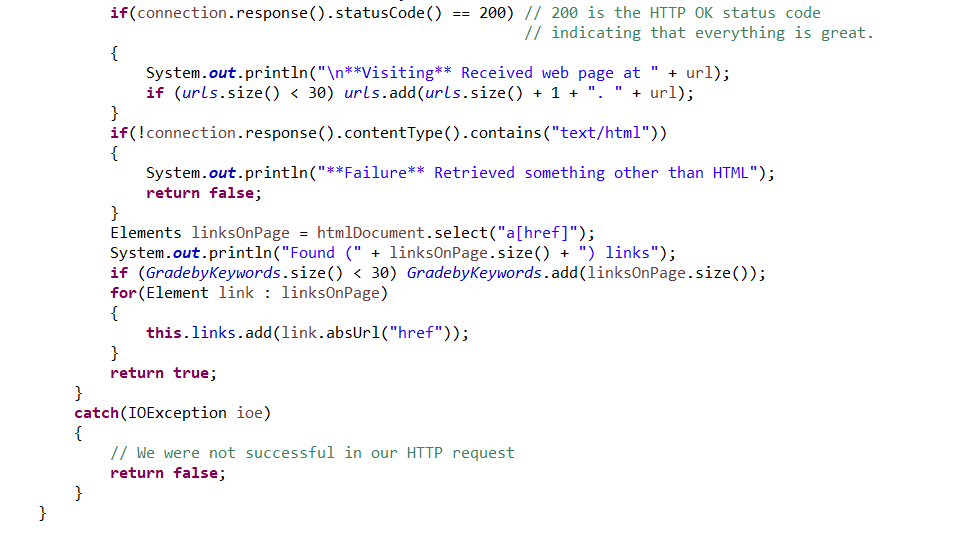
Here are some screen shots of these two classes:

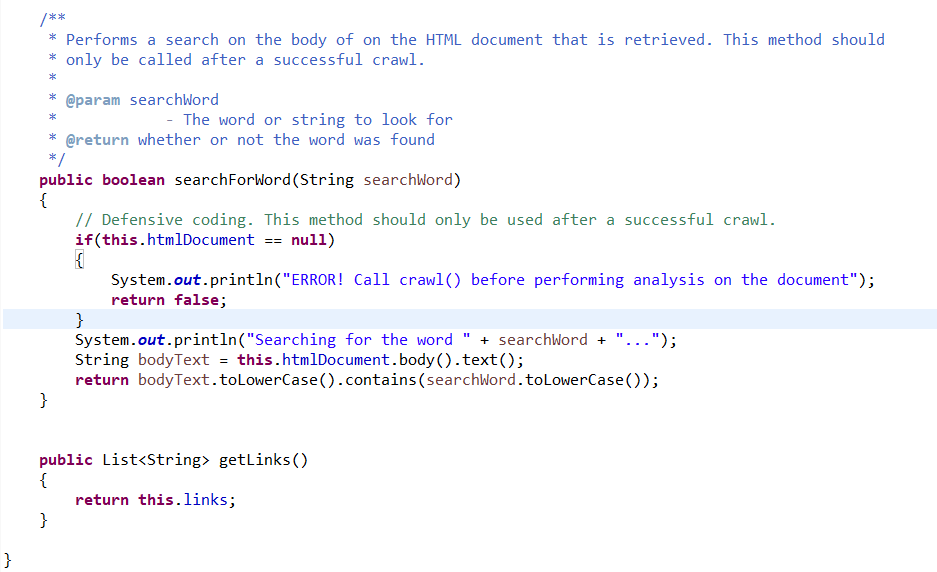




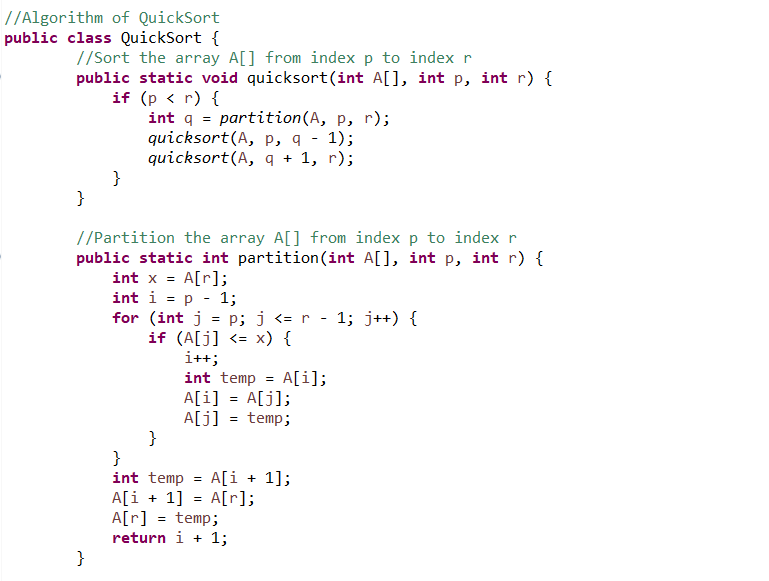


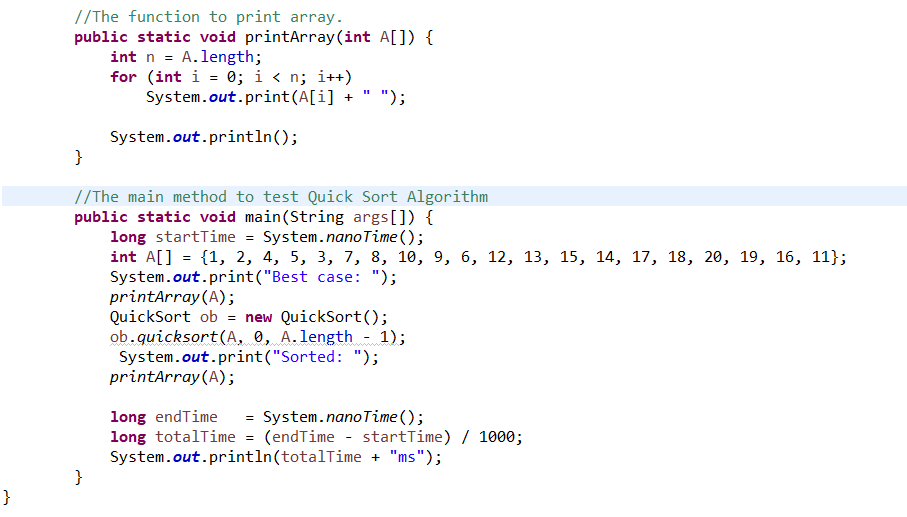




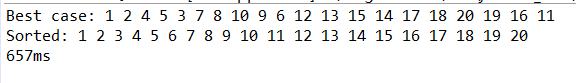


Class QuickSort is the part to achieve quicksort. Here are some screen shots of it:

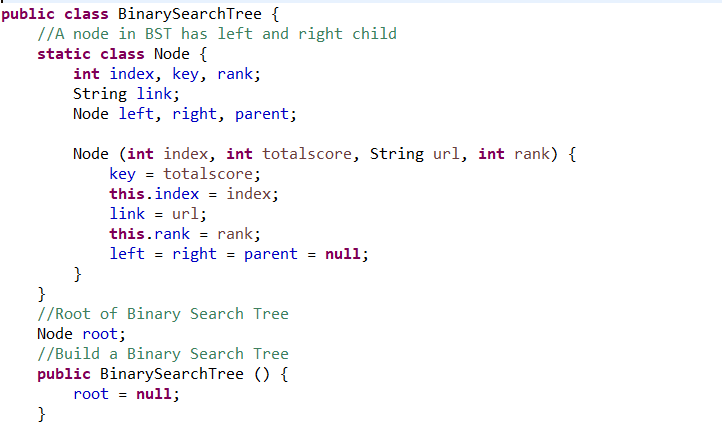


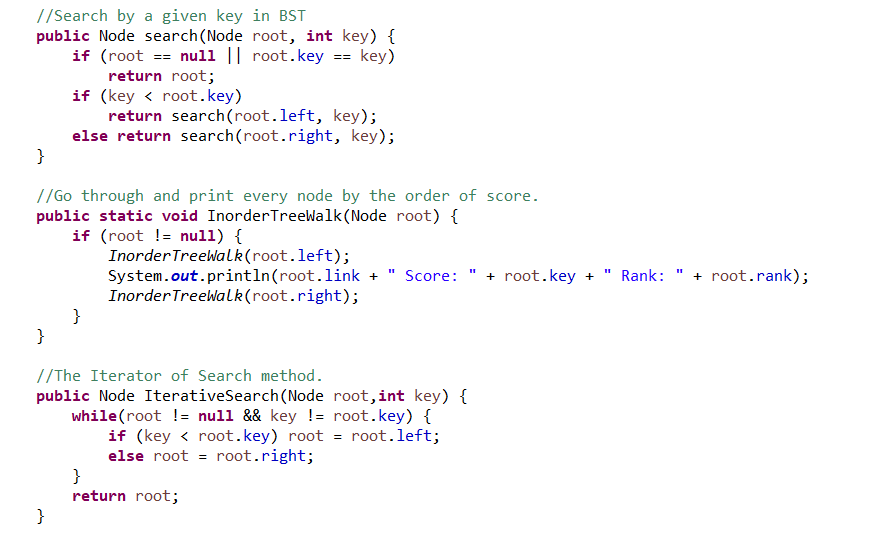


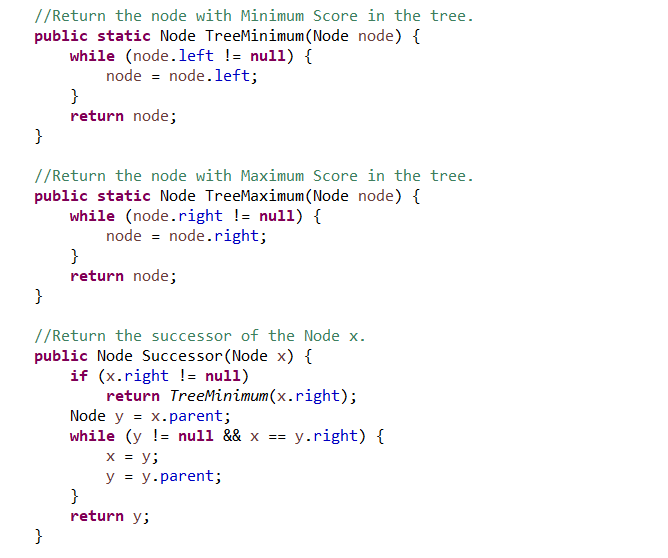
Here is the result of the test:

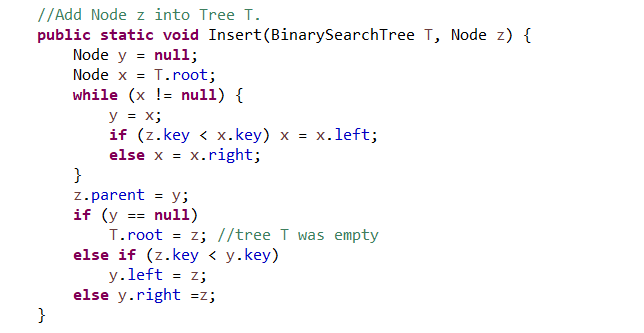


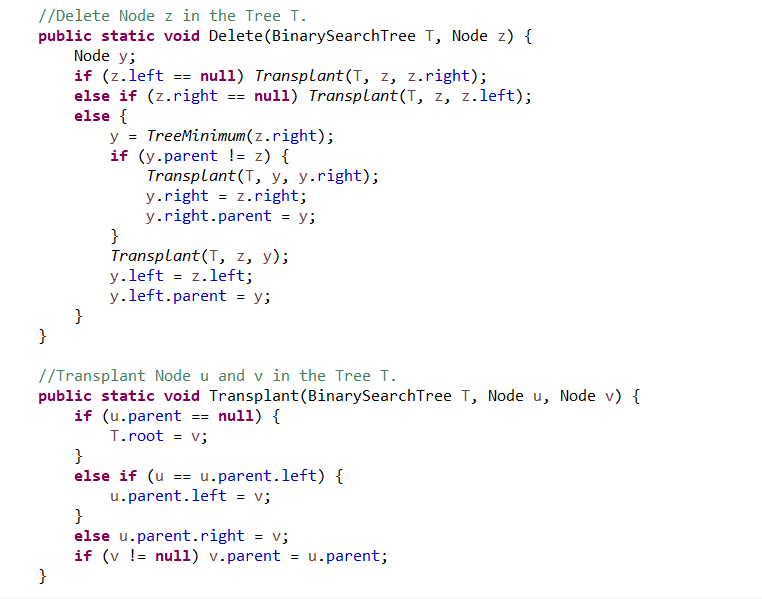
Class BinarySearchTree is the part to build Binary Search Tree. It included the function both from text book and my customer made. Here are some screen shots of it:







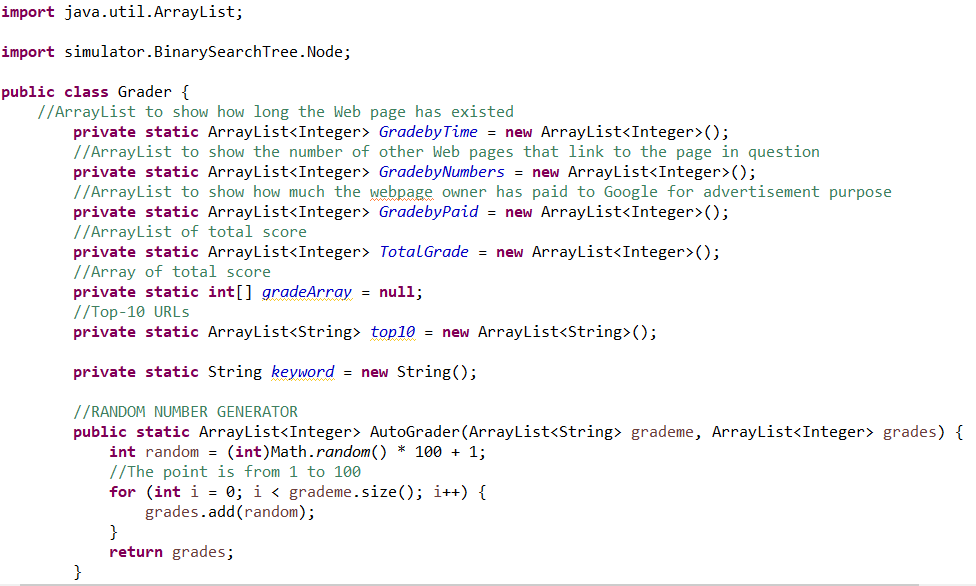


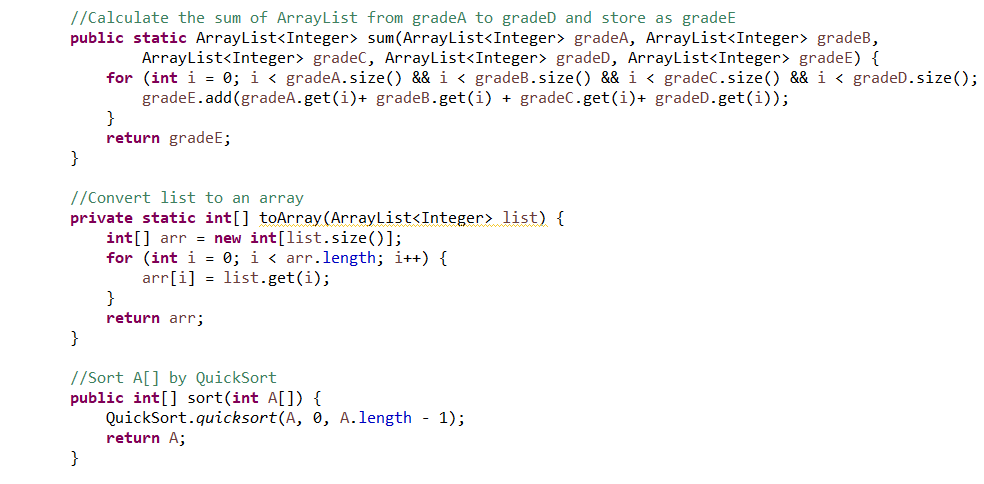


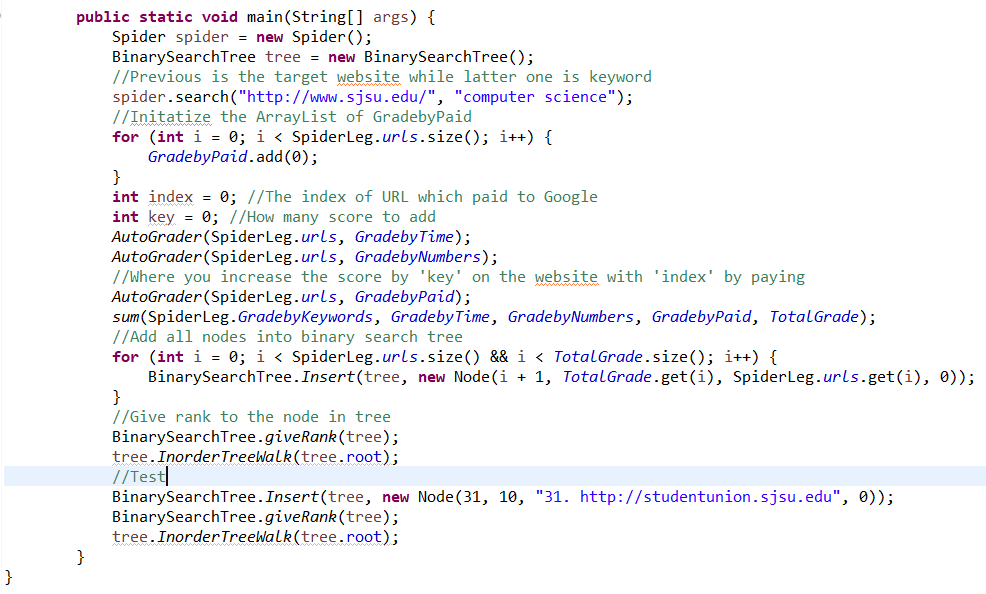


Class Grader is the executive class of the entire programming. In this class I collect 30 URLs from the web crawler, analyze and put them into the Binary Search Tree. In the main method we can print all 30 URLs and their index, total score and page rank. Moreover, we can do insert and delete using the function in Class Binary Search Tree. Which will change the result of printing.

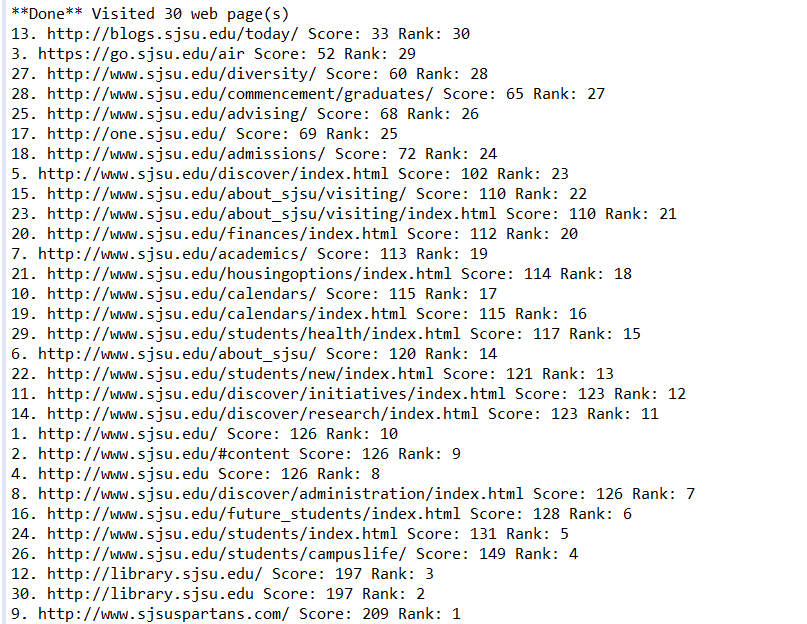
The screen shots of this class are the following:



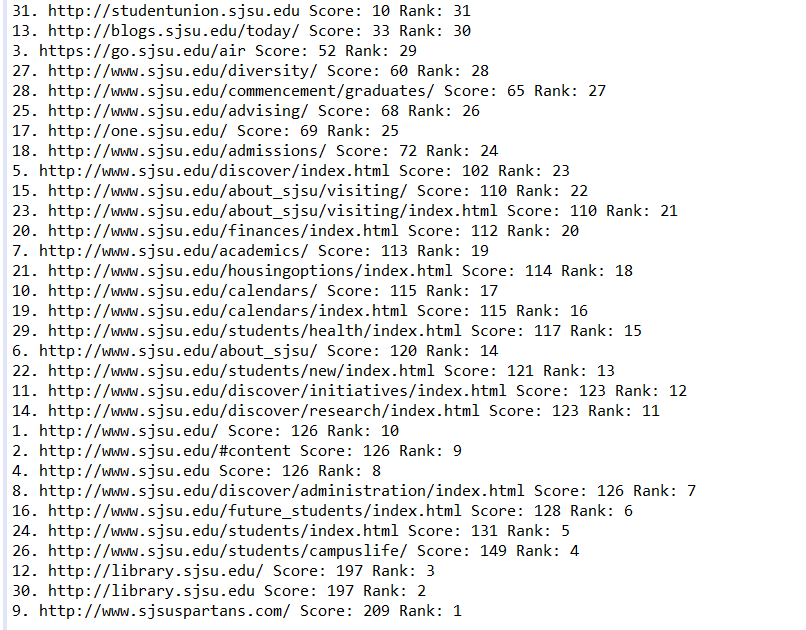




Here is the result of 30 URLs:

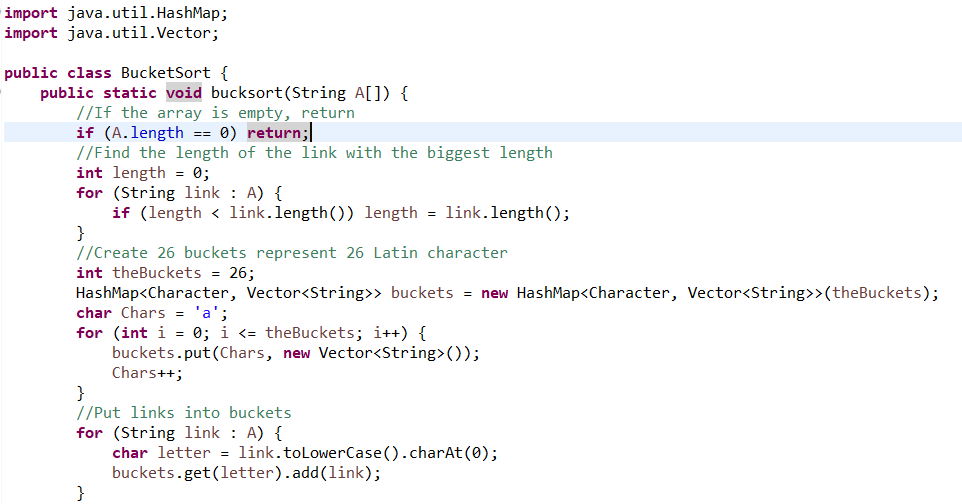


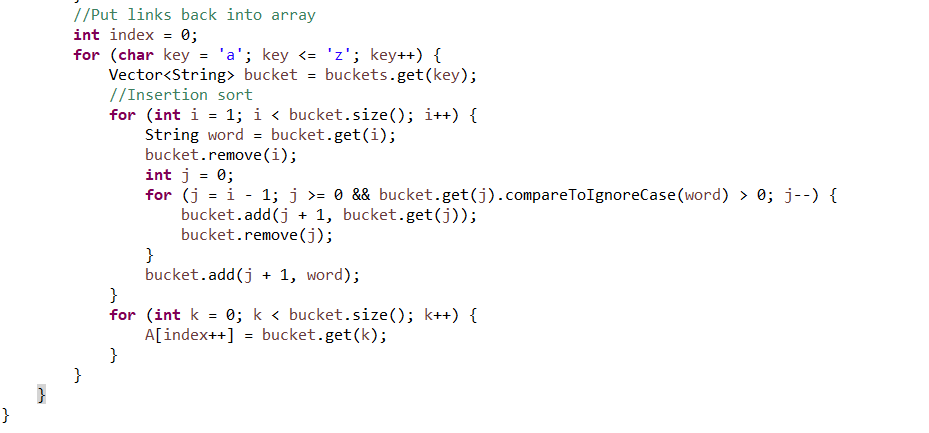
Here is the result after insertion, we can find a new URL in the list:



Class BucketSort is use bucket sort to sort URLs (String). Just as previous homework I create 26 buckets and put the URLs into it. And I use insertion sort to sort the URLs with the same initial character.

The screen shots are the following:

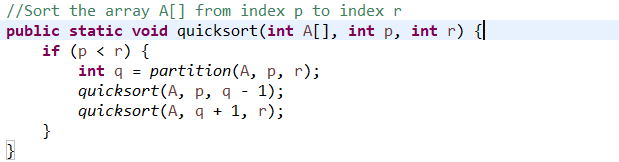




Functions:

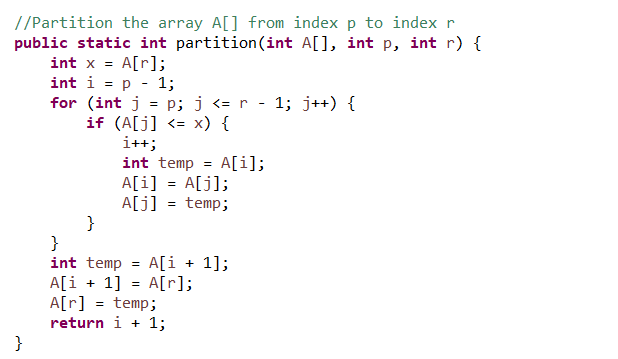
Class QuickSort:

Function quicksort:



This function recursively sorts the two subarrays.

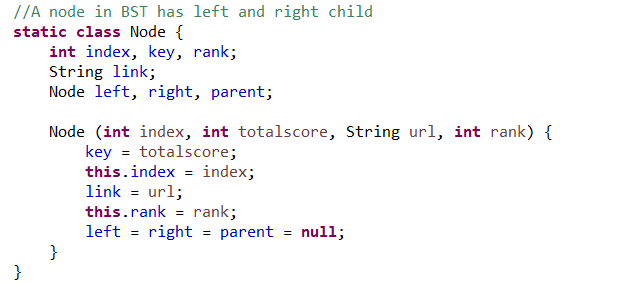
Function partition:



This function Partition the array into two subarrays around a pivot x such that elements in lower subarray≤ x ≤ elements in upper subarray.

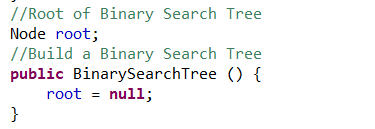
Class Binary Search Tree:

Class Node:



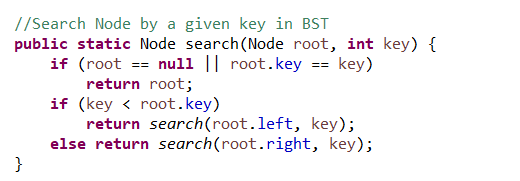
Create Node contains the index, total score, string and rank of a URL. It cannot be public because it is based on Binary Search Tree.

Constructor Binary Search Tree:



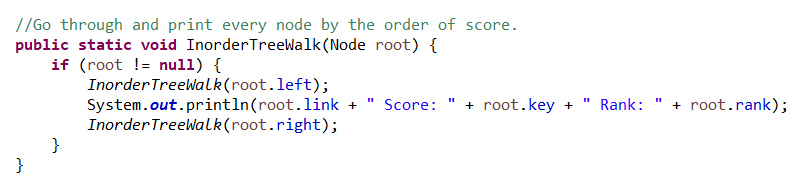
The constructor of Binary Search Tree initializes the tree.

Function search:



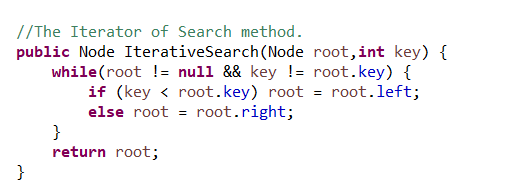
Search the certain node in the tree by its key.

Function Inorder-Tree-Walk:



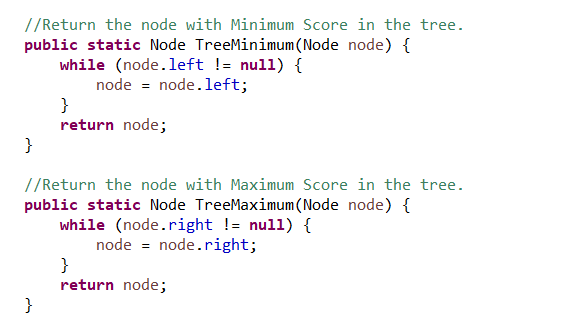
Go through the entire tree and print the detail of each node.

Function Iterative Search:



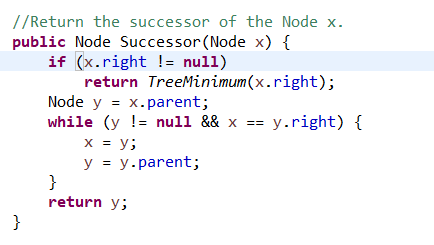
The Iterator of Search.

Function TreeMinimum and TreeMaximum:



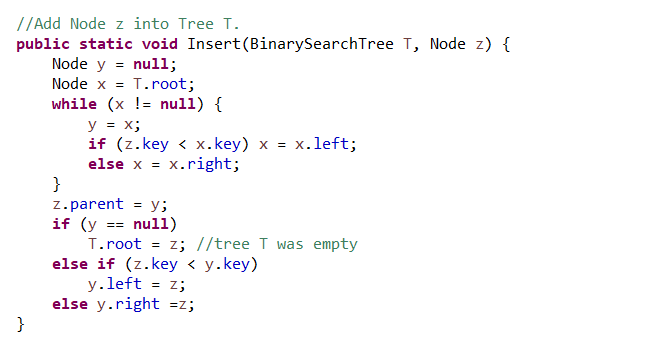
Return the node with the minimum or maximum total score in the tree.

Function Successor:



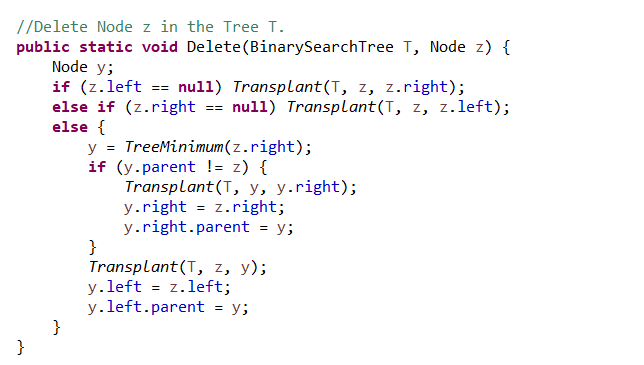
Return the successor of the target node. It will be used in delete algorithm.

Function Insert:



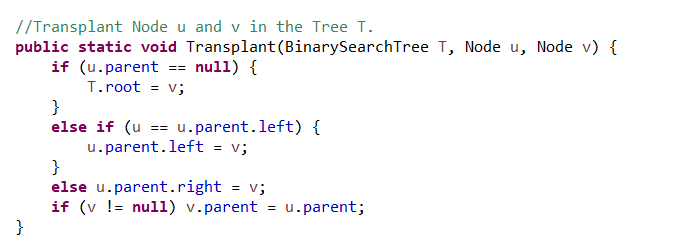
Add a new node into the tree.

Function Delete:



Delete the target node form the tree.

Function Transplant:



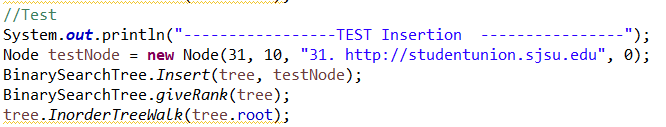
Transplant node u and node v. It will be used in Delete algorithm.

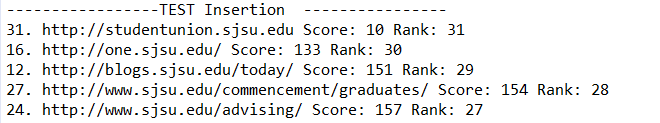
Function giveRank:



My customer function. It can update the rank of all the node in tree.

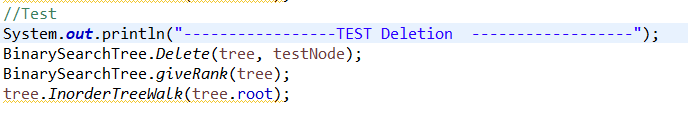
Test of Insertion:

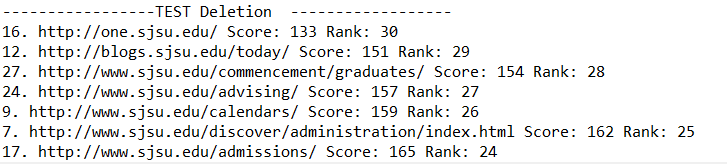




As the screen shots, a new URL has been added with its index, link, total score and rank.

Test of Deletion:





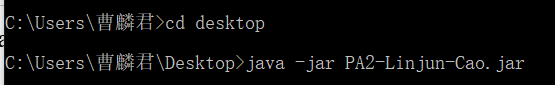
After deletion, the new added URL has been removed.

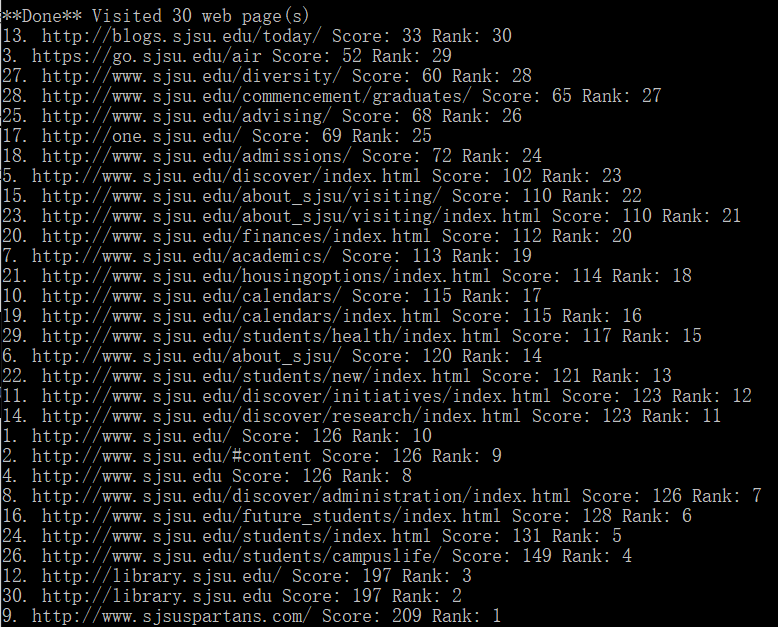
Part III. The procedure of how to unzip files, install application, and run/test codes:

First, unzip the zip and you will see a jar file called: PA2-Linjun-Cao.jar. Then run the command and code in:

java -jar PA2-Linjun-Cao.jar

And the jar file will run directly.





Part IV. Problems encountered during the implementation

I meet several problems when I implement and test program.

First, when I start to implement the pseudo code from the textbook, there are always bugs. Since the pseudo code just teaches an idea about how this program runs, in the real java code, it needs to be fixed.

Also, since the total score of each URL is from RANDOM NUMBER GENERATOR, so the result is quite different from every time I test. And it will take more time for me to check.

Part V. Lessons Learned：

I learn two things from this assignment. First, choose data structure carefully. There are many different types can be chosen. But not everyone is work. This time I choose Vector when I do bucket sort, and it is successful.

Next, I can’t just enter the code in book into the program directly. It need to be reversed by debugging.