Programming Assignment #1

CS146-08

Professor: Mike Wu

By: Eric Wu

Table of Contents

1)Explain/Illustrate your Google Search Engine Simulator design and implementation details.

To begin, I needed to get 30 URLs, so I downloaded the WebCrawler made by Stephen from netinstructions.com. I used the same URL he provided in his example, but the keyword is anything the user input. After that, the WebCrawler goes ahead and collect 30 URLs, which I then stored into an ArrayList. I then made a pair object which takes two parameters, a URL and a score. So back to the URLs I have collected, I iterated through the URLs and made a pair object for each URL. The score I assigned with the URL is the sum of 4 factors, a random frequency score with the range of 1-100, a random history score with the range of 1-100, a random relevancy score with the range of 1-100, and finally a random advertisement score with the range of 1-100. Also, at the same time when iterating through the URLs, I added the new pair object into a new ArrayList. After I am done iterating, I made a new heap object containing the ArrayList with pair objects. I then printed the URLs and score to check what we have. I then used the heapSort method on the array. Again, I print the URLs and score to see the change. After that I used the buildMaxHeap method on the array. And again, I print the URLs and score to see the change made to it. Following that, I ask the user if they want to change the score of any of the URLs? This would be if someone had paid google to increase their visibility of their website. If they do, they type in Y, but if they don't they can type in N. If they typed Y, the console prompts the user to enter the index of the URL where we want the score to chance. I then ask them to enter a new higher score than the previous because you cannot lower the score if you pay google. I repeat these steps until the user types N, which then takes then out of this loop. After that I use the heapExtractMax on the heap 10 times and I print out the associated Index and PageRank of them. This would be the final top ten search results. The program then asks the user if they want to search for any other keyword. If they type Y, the whole process happens again, but if they type N, the program wraps up and print the top 10 unique keyword searches. This top 10 mean that even if the user has searched the same keyword twice, only one of them will show up.

2)A list of classes/subroutines/function calls and explanations of each purpose.

Pair Class

```java
public class Pair {
    private String url;
    private int value;
    /**
     * This creates a Pair object which has a string and a value
     * @param url   The URL of the pair
     * @param value The value associated with the URL
     */
    public Pair(String url, int value) {
        this.url = url;     //store user inputted url into the object's url
        this.value = value; //store user inputted value into the object's value
    }
    /**
     * Gets the URL
     * @return the URL
     */
    public String getUrl() {
        return url;         //return the object's url
    }
    /**
     * Gets the value
     * @return the value
     */
    public int getValue() {
        return value;       //return the object's value
    }
    /**
     * Combine the URL and Value into a string
     * @return the combined URL and value string
     */
    public String toString() {
        return "Score: " + value + " URL: " + url;  //return a string contraining the url and value
    }
}
```

The purpose of this pair class is so that we can store each URL with a corresponding score. I could have used a map to store the data but by making a new object to store the URL and score, I know for certain that the data would be stable.

The constructor takes in a String (which is the URL) and a integer (which is the score of the URL). I have 3 methods in this class.

One of them is **getUrl()** which returns the URL that is associated with the object.

Another one is the **getValue()** which returns the score that is associated with the object.

And finally there is the **toString()** that just returns the score and URL in one line, which just makes it easier to print out the results.

Heap Class

```java
import java.util.ArrayList;

/**
 *  This is the Heap class
 * @author Eric
 */
public class Heap {
    private int heapSize = 0;
    private ArrayList<Pair> A;
    /**
     * This is constructor which takes in an array and makes a Heap object
     * @param array
     */
    public Heap(ArrayList array) {
        heapSize = array.size() - 1;    //makes the heapsize the size of the array
        A = array;  //make the object's arraylist A equal to the user input array
    }
    /**
     * This gets the index of the parent of a given node
     * @param i the index of a node
     * @return the index of the parent
     */
    public int getParent(int i) {
        return (i - 1) / 2;     //formula to get parent of a node in a binary tree
    }
    /**
     * This gets the index of the left child of a given node
     * @param i the index of a node
     * @return the index of the left child
     */
```

This heap class is the data structure which we would be using to store and organize our data.

I import the arraylist because that is what I am going to be using to store the nodes instead of an array because of the maxHeapInsert, we need to expand array size. It is just simpler to expand using an arraylist vs an array. I have an integer instance variable called heapsize which I initialized to be 0, and an arraylist with pair objects that I called A.

The constructor takes an arraylist (with pair objects) and I reinitialize the heapsize to the arraysize, and set A equal to the arraylist the user inputted.

The first method I have in this class is the **getParent()** which just returns the parent index of a given node index.

```java
public int getLeft(int i) {
    return (2 * i) + 1;     //formula to get left child of a node in a binary tree
}
/**
 * This gets the index of the right child of a given node
 * @param i the index of a node
 * @return the index of the right child
 */
public int getRight(int i) {
    return (2 * i) + 2;     //formula to get right child of a node in a binary tree
}
/**
 * This maxheapify the value at index i of an arraylist
 * @param i The index of the arraylist that we wish to perform maxheapify on
 */
public void maxHeapify(int i) {
    int largest;        //largest variable
    int l = getLeft(i); //l is the left child of the node i
    int r = getRight(i);//r is the right child of the node i
    if(l <= heapSize && A.get(l).getValue() > A.get(i).getValue()) {
        largest = l;    //if heapsize is >= l and left value is greater than i score
    } else {
        largest = i;    //if heapsize is < l and left value is less than i score
    }
    if(r <= heapSize && A.get(r).getValue() > A.get(largest).getValue()) {
       largest = r;     //if heapsize is >= r and r value is greater than largest value
    }
    if(largest != i) {  //if the largest variable is not i
        Pair temp = A.get(i);       //temp variable to store i
        A.set(i, A.get(largest));   //change position of i into the largest
        A.set(largest, temp);       //change position of largest into i
        maxHeapify(largest);        //repeat previous steps on the largest
    }
}
```

The next method I have is the **getLeft()** which returns the left child index of a given node index.

The next method I have is the **getRight()** which returns the right child index of a given node index.

Then there is the **maxHeapify()** method which takes an index of a node as an argument. We initialize a variable called largest, l which is the left child of the node, and r which is the right child of the node. If the l is less than or equal to heapSize and the score from the l node is greater than the score from the i node, we set largest to equal l. If that is not true, than we set largest to be equal to i. Then we have another if statement that is if r is less than heapSize and the score from node r is greater than the score from the node largest, we set largest equal to r. And finally the last if statement in this method checks if largest is not equal to i. If it is not equal, we initialize a temp variable equal to the node i. We then set the I index to the largest node. Then we set the largest index to largest. Then we would call maxHeapify() on the largest index. The purpose of this method is to check if a node is in the heap format. If it doesn't fit its criteria, we fix it by checking the the right and left child and swapping it with he bigger of the two. We repeat this process until that node is in its correct place in the heap.

```java
/**
 * This method rearranges the values of an arraylist into one that follows the maxheap format
 */
public void buildMaxHeap() {
    heapSize = A.size() - 1;    //heapsize is set to the size of the array
    for(int i = (A.size() - 1) / 2; i >= 0; i--) {
        maxHeapify(i);  //call the maxheapify method from half the size of the array to 0
    }
}
/**
 * Sorts the arraylist in ascending order
 */
public void heapSort() {
    buildMaxHeap(); //call the build max heap method
    for(int i = A.size() - 1; i >= 1; i--) {    //start from array size down to 1
        Pair temp = A.get(0);   //temp variable to store value at 0
        A.set(0, A.get(i));     //set the 0 index to the value at i
        A.set(i, temp);         //set the i index to the value at 0
        heapSize--;             //decrease heapsize by 1
        maxHeapify(0);          //call maxheapify method on the 0 index
    }
}
/**
 * gets the max value in the heap
 * @return the max value in the heap
 */
public Pair heapMaximum() {
    return A.get(0); //return the node with the max score
}
```

The next method we have is the **buildMaxHeap()** method. This method basically sets heapSize to the size of the array. It then iterates through half the array, from the middle down to 0, and each time it calls maxHeapify() on the current index. The purpose of this method is to rearrange the values of an arraylist into one that follows the maxheap format.

The next method we have is the **heapSort()** method. We begin by calling the buildMaxHeap() method so that it rearranges the values in the arraylist into a maxheap format. Then it iterates from the end of the array down to 1 and each time it sets a temp variable to equal node at index 0. Then we set the 0 index to be equal to the current index's node. We then set the current index to equal to temp. We decrement the heapSize by one then we call maxHeapify() on the 0 index. The purpose of this method is so we could sort the arraylist from ascending order.

The next method we have is **heapMaximum()**. This method returns the node with the maximum score and it does it by returning the 0 index node.

```
    * gets the max value in the heap and remove it, then perform maxheapify on the top node
    * @return the max value in the heap
    */
    public Pair heapExtractMax() {
        Pair max = A.get(0);        //set the max variable to be the the value at index 0
        A.set(0, A.get(heapSize));  //set the index 0 into the value at the index heapsize
        heapSize--;                 //decrease heapsize by 1
        maxHeapify(0);              //call maxheapify method on the 0 index
        return max;                 //return the max variable
    }
    /**
    * Increase the priority of a node in the heap
    * @param i the index of the node that we wish to increase the priority
    * @param key the new value of the priority
    */
    public void heapIncreaseKey(int i, int key) {
        Pair newPair = new Pair(A.get(i).getUrl(), key);    //new pair object with the i index url and new key
        A.set(i, newPair);  //replace the i index with the new pair
        while(i > 0 && A.get(getParent(i)).getValue() < A.get(i).getValue()) {
            Pair temp = A.get(i);   //temp variable is the i index
            A.set(i, A.get(getParent(i)));  //set the i index to the parent of i value
            A.set(getParent(i), temp);      //set the parent of i index to temp
            i = getParent(i);               //i is set to the value at parent of i
        }
    }

    /**
    * We can insert a new node into the heap and keep the heap property
    */
    public void maxHeapInsert(Pair newNode) {
        heapSize = A.size() - 1;    //heapsize is the size of array
        heapSize++;                 //increase the heapsize by 1
        Pair node = newNode;        //set node equal to the newNode input
        A.add(node);                //add the new node to the array
        heapIncreaseKey(heapSize, node.getValue()); //call the heapIncreaseKey method on the value of the node
```

The next method we have is the **heapExtractMax()**. It starts by setting a max variable to the 0 node. Then we set the 0 node to be equal to the last node. We decrease the heapSize by 1, because we are essentially popping a node out of the heap, so we need to account for the decrease in size. Then we call maxHeapify() on the 0 index. And finally we return the max variable. The purpose of this method is to get the max value node in the heap, and remove it. After that we rearrange the values in the arraylist so it follows the heap structure again.

The next method we have is **heapIncreaseKey()**. It takes a index and a key value. It starts by creating a new variable called newPair which is a Pair object that contains the URL from the i node, and the new score the user inputted. We replace the current i node with the newPair. Then we have a while loop that if i is greater than 0, and if the parent value of node i is less than node i value. If the conditions meet, we make a temp variable that stores the i node, set the I index to its parent node, set the parent index to be equal to temp and finally set i equal to its parent. The purpose of this method is to increase the priority of a node in the heap. Then it rearranges the heap again so it follows the heap structure again.

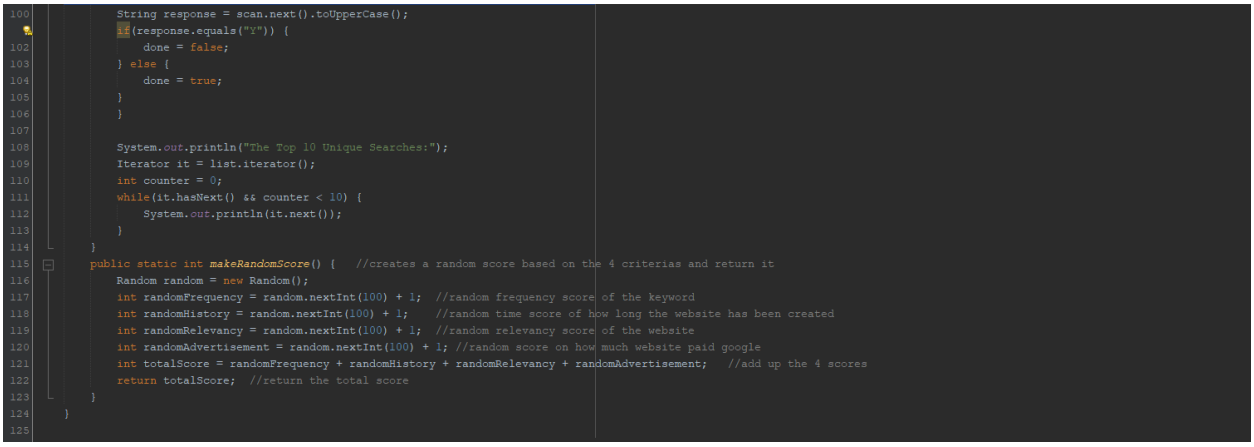
The last method we have in this class is the **maxHeapInsert()** method. It starts by setting heapSize equal to the size of the array. It then increases the heapSize by one to account for the new node we are adding. Then it adds the user inputted pair, into the arraylist. Then it calls the increaseKey() method on the new heapSize index and the new node value. The purpose of this method is so that we can insert new elements into the heap datastructure. Lets just say that we want to add a new URL into the heap, we can do so through this.

3) Provide a lot of screen shots of each simulation process/procedure including inputs and outputs for each of function listed in item 6 associated with the two major features listed in the end of the Problem Statements. This is to verify your codes and to check to see if your codes match the functional requirements and run correctly by yourself.

PageRank Class

```
import java.util.*;

public class Pagerank {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        boolean done = false;
        LinkedHashSet<String>list = new LinkedHashSet<String>();
        while(!done) {
        System.out.println("What keyword do you want to search?: ");
        String keyWord = scan.next();
        list.add(keyWord);
        Spider crawler = new Spider();      //new spider object
        crawler.search("https://arstechnica.com/", keyWord);     //website we are getting URLs from, and keyword is science
        ArrayList<String> array = crawler.getPageList();    //set the string arraylist equal to the crawler's list of URLs
        ArrayList<Pair> A = new ArrayList<>();

        for(int i = 0; i < array.size(); i++) {
            Pair pair = new Pair(array.get(i), makeRandomScore());  //assign each url a random score
            A.add(pair);    //store each url and ranodm score into the arraylist
        }

        Heap heap = new Heap(A);    //Making a new heap object with the A arraylist as the argument
        System.out.println("The 30 URLs we have collected!");
        for(int i = 0; i < A.size(); i++) {
            System.out.println("Index: " + i + " " + A.get(i).toString());  //prints the URLs unmodified
        }   System.out.println();

        heap.maxHeapify(0);  //tests the maxHeapify method on the 0th index of the arraylist
        System.out.println("The 30 URLs after maxHeapify is ran on the 0th index!");
        for(int i = 0; i < A.size(); i++) {
            System.out.println("Index: " + i + " " + A.get(i).toString());  //prints the URLs
        }   System.out.println();

        heap.heapSort();    //tests the heapSort method, it should sort the URLs based on the scores in ascending order
        System.out.println("After we run the heapSort method, the 30 URLs are now sorted in ascending order!");
        for(int i = 0; i < A.size(); i++) {
            System.out.println("Index: " + i + " " + A.get(i).toString());  //prints the URLs sorted
        }   System.out.println();

        heap.buildMaxHeap();    //test the buildMaxHeap method, it should rearrange arraylist so it follows heap structure
        System.out.println("After we run the buildMaxHeap method, the 30 URLS are now rearranged so it follows the heap property!");
        for(int i = 0; i < A.size(); i++) {
            System.out.println("Index: " + i + " " + A.get(i).toString());  //print the URLs so that it follows the heap structure
```

```
        }   System.out.println();

        System.out.println("Do you want to change the score of any URL? (Y/N): ");  //prompts user to change any score if needed
        String result = scan.next().toUpperCase();
        while(result.equals("Y")) {
            System.out.println("Enter the Index of the URL that you want to change the score: ");
            int index = scan.nextInt();
            int rank = A.get(index).getValue();
            System.out.println("Enter a new score higher than " + rank + " : ");    //we are increasing key, not decreasing
            int newRank = scan.nextInt();
            heap.heapIncreaseKey(index, newRank); //tests the heapIncreaseKey method, the new arraylist should also follow the heap structure
            System.out.println("Do you want to change the score of any other URL? (Y/N): ");    //ask user if they want to continue changing scores
            result = scan.next().toUpperCase();
        }
        System.out.println("The heap after changing scores of any URL!:");
        for(int i = 0; i < A.size(); i++) {
            System.out.println("Index: " + i + " " + A.get(i).toString());  //print the URLs so that it follows the heap structure
        }   System.out.println();

        System.out.println("We are testing the maxHeapInsert method!");
        System.out.println("The website name is MikeWu.com and the score is the max integer!");
        Pair mike = new Pair("https://www.MikeWu.com/", Integer.MAX_VALUE);
        heap.maxHeapInsert(mike);   //test out the maxHeapInsert method, it should place this node to the top of the priority queue
        System.out.println("The 31 URLs after we add a new URL!:");
        for(int i = 0; i < A.size(); i++) {
            System.out.println("Index: " + i + " " + A.get(i).toString());  //print the URLs so that it follows the heap structure
        }   System.out.println();

        System.out.println("We are testing the heapMaximum method which should return MikeWu's website and score!");
        System.out.println("The max node is: " + heap.heapMaximum().toString()); //test the heapMaximum method
        System.out.println();                                                    //it should print out MikeWu's score b/c it is highest value

        ArrayList<Pair> priorityQueue = new ArrayList<>();  //make a new priority queue arraylist
        Heap priority = new Heap(priorityQueue);    //make a new priority heap that takes the previous arraylist as an argument
        for(int i = 0; i < 10; i++) {
            priority.maxHeapInsert(heap.heapExtractMax());  //test the heapExtractMax method and also the maxHeapInsert method
        }                                                   //the extracted node is not inserted into the new heap

        System.out.println("The Top 10 Search Results:");
        for(int i = 0; i < 10; i++) {
            System.out.println("Index: " + i + " PageRank: " + (i + 1) + " " + priority.heapExtractMax().toString());
        }   System.out.println();   //the line above extracts the max node from the priority heap and call the toString() method on it

        System.out.println("Do you want to search for any other word? (Y/N): ");
```



```
        String response = scan.next().toUpperCase();
        if(response.equals("Y")) {
            done = false;
        } else {
            done = true;
        }
        }

        System.out.println("The Top 10 Unique Searches:");
        Iterator it = list.iterator();
        int counter = 0;
        while(it.hasNext() && counter < 10) {
            System.out.println(it.next());
        }
    }
    public static int makeRandomScore() {   //creates a random score based on the 4 criterias and return it
        Random random = new Random();
        int randomFrequency = random.nextInt(100) + 1;  //random frequency score of the keyword
        int randomHistory = random.nextInt(100) + 1;    //random time score of how long the website has been created
        int randomRelevancy = random.nextInt(100) + 1;  //random relevancy score of the website
        int randomAdvertisement = random.nextInt(100) + 1; //random score on how much website paid google
        int totalScore = randomFrequency + randomHistory + randomRelevancy + randomAdvertisement;   //add up the 4 scores
        return totalScore;  //return the total score
    }
}
```

First I will explain how the WebCrawler works. The WebCrawler application I am using also uses jsoup and it has two classes, a Spider class and a SpiderLeg class. The spider class basically uses the spiderleg class to “crawl” on a website, trying to find if it matches with the keyword we provide. The spider class takes two arguments to begin, a website url and a keyword string from which it is going to search the website for. The spider class has three instance variables, one called MAX_PAGES_TO_SEARCH which I set to 30 because we are looking for 30 websites. The next variable is pagesVisted which is a stored in a set. And finally the last variable is called pagesToVisit which keeps track of the remaining pages needed to check. It is stored with a list. This program gets 30 URLs by navigating from the main page. It then looks around for any links on the website, and it adds them to pagesToVisit. Then it traverse down the list of pagesToVisit and if check if the URL page contains the keyword and if it does, it tells us by printing out it found it. Then it adds the URL to the pagesVisited set. In this case, I used the made a spider object with the URL as https://arstechnica.com/ and user input as the keyword. I made this program repeat asking user for keywords until they type N and they will exit out of the program. After that, the program will print out the top 10 unique keywords that the user have inputted.

Time Complexity:

In this program I have 11 loops but looking at it asymptotically, it is still O(n). We used a WebCrawler but I am just copying the result (which is stored as a set) so that is O(n). As we learned in class, the heap function calls take O(nlgn). Since we are adding up 11(O(n)) + O(n) + O(nlgn), the big O is still O(nlgn) because it is still bigger than the other factors.

The following is the result from running the program with further explanation.

```
run:
What keyword do you want to search?:
science
```

```
**Visiting** Received web page at https://arstechnica.com/
Found (213) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/

**Visiting** Received web page at https://arstechnica.com/#main
Found (213) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/#main

**Visiting** Received web page at https://arstechnica.com
Found (213) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com

**Visiting** Received web page at https://arstechnica.com/information-technology/
Found (211) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/information-technology/

**Visiting** Received web page at https://arstechnica.com/gadgets/
Found (210) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/gadgets/

**Visiting** Received web page at https://arstechnica.com/science/
Found (208) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/science/

**Visiting** Received web page at https://arstechnica.com/tech-policy/
Found (211) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/tech-policy/

**Visiting** Received web page at https://arstechnica.com/cars/
Found (209) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/cars/

**Visiting** Received web page at https://arstechnica.com/gaming/
Found (212) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/gaming/

**Visiting** Received web page at https://arstechnica.com/civis/
Found (89) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/civis/

**Visiting** Received web page at https://arstechnica.com/store/product/subscriptions/
Found (25) links
Searching for the word science...
```

```
**Visiting** Received web page at https://arstechnica.com/search/
Found (52) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/search/

**Visiting** Received web page at https://arstechnica.com/#site-menu
Found (213) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/#site-menu

**Visiting** Received web page at http://video.arstechnica.com/
Found (155) links
Searching for the word science...
**Success** Word science found at http://video.arstechnica.com/

**Visiting** Received web page at https://arstechnica.com/features/
Found (203) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/features/

**Visiting** Received web page at https://arstechnica.com/reviews/
Found (211) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/reviews/

**Visiting** Received web page at https://arstechnica.com/tag/ars-approved/
Found (204) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/tag/ars-approved/

**Visiting** Received web page at https://arstechnica.com/rss-feeds/
Found (71) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/rss-feeds/

**Visiting** Received web page at https://arstechnica.com/?view=mobile
Found (140) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/?view=mobile

**Visiting** Received web page at https://arstechnica.com/about-us/
Found (58) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/about-us/

**Visiting** Received web page at https://arstechnica.com/staff-directory/
Found (119) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/staff-directory/

**Visiting** Received web page at https://arstechnica.com/contact-us/
Found (56) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/contact-us/
```

```
**Visiting** Received web page at https://arstechnica.com/advertise-with-us/
Found (54) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/advertise-with-us/

**Visiting** Received web page at https://arstechnica.com/reprints/
Found (53) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/reprints/

**Visiting** Received web page at http://arstechnica.com/?view=grid
Found (213) links
Searching for the word science...
**Success** Word science found at http://arstechnica.com/?view=grid

**Visiting** Received web page at http://arstechnica.com/?view=archive
Found (209) links
Searching for the word science...
**Success** Word science found at http://arstechnica.com/?view=archive

**Visiting** Received web page at http://arstechnica.com/?theme=light
Found (213) links
Searching for the word science...
**Success** Word science found at http://arstechnica.com/?theme=light

**Visiting** Received web page at http://arstechnica.com/?theme=dark
Found (213) links
Searching for the word science...
**Success** Word science found at http://arstechnica.com/?theme=dark

**Visiting** Received web page at https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Found (22) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F

**Visiting** Received web page at https://arstechnica.com/civis/ucp.php?mode=sendpassword
Found (18) links
Searching for the word science...
**Success** Word science found at https://arstechnica.com/civis/ucp.php?mode=sendpassword

**Done** Visited 30 web page(s)
```

This is the result from using the WebCrawler, we have gone through the 30 URLs.

Notice that the program ask the user what keyword they want to search for.

```
The 30 URLs we have collected!
Index: 0 Score: 133 URL: http://arstechnica.com/?theme=dark
Index: 1 Score: 168 URL: https://arstechnica.com/search/
Index: 2 Score: 213 URL: http://arstechnica.com/?view=archive
Index: 3 Score: 299 URL: https://arstechnica.com/tag/ars-approved/
Index: 4 Score: 294 URL: https://arstechnica.com/reviews/
Index: 5 Score: 195 URL: http://arstechnica.com/?theme=light
Index: 6 Score: 292 URL: https://arstechnica.com/about-us/
Index: 7 Score: 201 URL: https://arstechnica.com/staff-directory/
Index: 8 Score: 90 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index: 9 Score: 253 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index: 10 Score: 192 URL: https://arstechnica.com
Index: 11 Score: 264 URL: https://arstechnica.com/rss-feeds/
Index: 12 Score: 182 URL: https://arstechnica.com/science/
Index: 13 Score: 200 URL: https://arstechnica.com/advertise-with-us/
Index: 14 Score: 209 URL: https://arstechnica.com/?view=mobile
Index: 15 Score: 167 URL: https://arstechnica.com/contact-us/
Index: 16 Score: 180 URL: https://arstechnica.com/
Index: 17 Score: 273 URL: http://arstechnica.com/?view=grid
Index: 18 Score: 226 URL: https://arstechnica.com/tech-policy/
Index: 19 Score: 130 URL: https://arstechnica.com/gadgets/
Index: 20 Score: 231 URL: https://arstechnica.com/civis/
Index: 21 Score: 274 URL: https://arstechnica.com/#site-menu
Index: 22 Score: 208 URL: https://arstechnica.com/gaming/
Index: 23 Score: 217 URL: https://arstechnica.com/store/product/subscriptions/
Index: 24 Score: 189 URL: https://arstechnica.com/features/
Index: 25 Score: 168 URL: https://arstechnica.com/information-technology/
Index: 26 Score: 70 URL: https://arstechnica.com/reprints/
Index: 27 Score: 294 URL: https://arstechnica.com/cars/
Index: 28 Score: 231 URL: https://arstechnica.com/#main
Index: 29 Score: 315 URL: http://video.arstechnica.com/
```

This is the URLs we have collected printed out. Note that nothing has been done to this arraylist.

```
The 30 URLs after maxHeapify is ran on the 0th index!
Index: 0 Score: 213 URL: http://arstechnica.com/?view=archive
Index: 1 Score: 168 URL: https://arstechnica.com/search/
Index: 2 Score: 292 URL: https://arstechnica.com/about-us/
Index: 3 Score: 299 URL: https://arstechnica.com/tag/ars-approved/
Index: 4 Score: 294 URL: https://arstechnica.com/reviews/
Index: 5 Score: 195 URL: http://arstechnica.com/?theme=light
Index: 6 Score: 209 URL: https://arstechnica.com/?view=mobile
Index: 7 Score: 201 URL: https://arstechnica.com/staff-directory/
Index: 8 Score: 90 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index: 9 Score: 253 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index: 10 Score: 192 URL: https://arstechnica.com
Index: 11 Score: 264 URL: https://arstechnica.com/rss-feeds/
Index: 12 Score: 182 URL: https://arstechnica.com/science/
Index: 13 Score: 200 URL: https://arstechnica.com/advertise-with-us/
Index: 14 Score: 315 URL: http://video.arstechnica.com/
Index: 15 Score: 167 URL: https://arstechnica.com/contact-us/
Index: 16 Score: 180 URL: https://arstechnica.com/
Index: 17 Score: 273 URL: http://arstechnica.com/?view=grid
Index: 18 Score: 226 URL: https://arstechnica.com/tech-policy/
Index: 19 Score: 130 URL: https://arstechnica.com/gadgets/
Index: 20 Score: 231 URL: https://arstechnica.com/civis/
Index: 21 Score: 274 URL: https://arstechnica.com/#site-menu
Index: 22 Score: 208 URL: https://arstechnica.com/gaming/
Index: 23 Score: 217 URL: https://arstechnica.com/store/product/subscriptions/
Index: 24 Score: 189 URL: https://arstechnica.com/features/
Index: 25 Score: 168 URL: https://arstechnica.com/information-technology/
Index: 26 Score: 70 URL: https://arstechnica.com/reprints/
Index: 27 Score: 294 URL: https://arstechnica.com/cars/
Index: 28 Score: 231 URL: https://arstechnica.com/#main
Index: 29 Score: 133 URL: http://arstechnica.com/?theme=dark
```

This is the URLs printed after we perform the maxHeapify operation on the 0th index. In this case it would be the “Index: 0 Score: 133 URL: http://arstechnica.com/?theme=dark” that is moved down to index 29. We can check if this is true. Previously the 0th index compare itself to its left (index 1) and right (index 2) child. The right child is bigger so it swaps. Now that it is at index 2, it compares itself to its left (index 5) and right (index 6) child again. The right child is bigger so it swaps. Now that it is at index 6, it compares itself to its left (index 13) and right (index 14) child. The right child is bigger so we swap. Now that it is at index 14, it compares itself to its left (index 29) and right child (does not exists). The left child is bigger so it swaps. Finally, we are done with maxHeapify on the index 0.

```
After we run the heapSort method, the 30 URLs are now sorted in ascending order!
Index: 0 Score: 70 URL: https://arstechnica.com/reprints/
Index: 1 Score: 90 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index: 2 Score: 130 URL: https://arstechnica.com/gadgets/
Index: 3 Score: 133 URL: http://arstechnica.com/?theme=dark
Index: 4 Score: 167 URL: https://arstechnica.com/contact-us/
Index: 5 Score: 168 URL: https://arstechnica.com/search/
Index: 6 Score: 168 URL: https://arstechnica.com/information-technology/
Index: 7 Score: 180 URL: https://arstechnica.com/
Index: 8 Score: 182 URL: https://arstechnica.com/science/
Index: 9 Score: 189 URL: https://arstechnica.com/features/
Index: 10 Score: 192 URL: https://arstechnica.com
Index: 11 Score: 195 URL: http://arstechnica.com/?theme=light
Index: 12 Score: 200 URL: https://arstechnica.com/advertise-with-us/
Index: 13 Score: 201 URL: https://arstechnica.com/staff-directory/
Index: 14 Score: 208 URL: https://arstechnica.com/gaming/
Index: 15 Score: 209 URL: https://arstechnica.com/?view=mobile
Index: 16 Score: 213 URL: http://arstechnica.com/?view=archive
Index: 17 Score: 217 URL: https://arstechnica.com/store/product/subscriptions/
Index: 18 Score: 226 URL: https://arstechnica.com/tech-policy/
Index: 19 Score: 231 URL: https://arstechnica.com/#main
Index: 20 Score: 231 URL: https://arstechnica.com/civis/
Index: 21 Score: 253 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index: 22 Score: 264 URL: https://arstechnica.com/rss-feeds/
Index: 23 Score: 273 URL: http://arstechnica.com/?view=grid
Index: 24 Score: 274 URL: https://arstechnica.com/#site-menu
Index: 25 Score: 292 URL: https://arstechnica.com/about-us/
Index: 26 Score: 294 URL: https://arstechnica.com/cars/
Index: 27 Score: 294 URL: https://arstechnica.com/reviews/
Index: 28 Score: 299 URL: https://arstechnica.com/tag/ars-approved/
Index: 29 Score: 315 URL: http://video.arstechnica.com/
```

We run the heapSort method on the 30 URLs which sorts the arraylist into an ascending order and then we print out the results.

```
After we run the buildMaxHeap method, the 30 URLS are now rearranged so it follows the heap property!
Index: 0 Score: 315 URL: http://video.arstechnica.com/
Index: 1 Score: 264 URL: https://arstechnica.com/rss-feeds/
Index: 2 Score: 299 URL: https://arstechnica.com/tag/ars-approved/
Index: 3 Score: 226 URL: https://arstechnica.com/tech-policy/
Index: 4 Score: 253 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index: 5 Score: 294 URL: https://arstechnica.com/cars/
Index: 6 Score: 294 URL: https://arstechnica.com/reviews/
Index: 7 Score: 213 URL: http://arstechnica.com/?view=archive
Index: 8 Score: 217 URL: https://arstechnica.com/store/product/subscriptions/
Index: 9 Score: 231 URL: https://arstechnica.com/#main
Index: 10 Score: 192 URL: https://arstechnica.com
Index: 11 Score: 274 URL: https://arstechnica.com/#site-menu
Index: 12 Score: 292 URL: https://arstechnica.com/about-us/
Index: 13 Score: 201 URL: https://arstechnica.com/staff-directory/
Index: 14 Score: 208 URL: https://arstechnica.com/gaming/
Index: 15 Score: 209 URL: https://arstechnica.com/?view=mobile
Index: 16 Score: 180 URL: https://arstechnica.com/
Index: 17 Score: 133 URL: http://arstechnica.com/?theme=dark
Index: 18 Score: 182 URL: https://arstechnica.com/science/
Index: 19 Score: 189 URL: https://arstechnica.com/features/
Index: 20 Score: 231 URL: https://arstechnica.com/civis/
Index: 21 Score: 167 URL: https://arstechnica.com/contact-us/
Index: 22 Score: 90 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index: 23 Score: 273 URL: http://arstechnica.com/?view=grid
Index: 24 Score: 195 URL: http://arstechnica.com/?theme=light
Index: 25 Score: 168 URL: https://arstechnica.com/search/
Index: 26 Score: 200 URL: https://arstechnica.com/advertise-with-us/
Index: 27 Score: 130 URL: https://arstechnica.com/gadgets/
Index: 28 Score: 70 URL: https://arstechnica.com/reprints/
Index: 29 Score: 168 URL: https://arstechnica.com/information-technology/
```

We run the buildMaxHeap method which then rearranges the arraylist so that it follows the heap property. We can also see that the arraylist now follow the heap structure.

```
Do you want to change the score of any URL? (Y/N):
Y
Enter the Index of the URL that you want to change the score:
29
Enter a new score higher than 168 :
325
Do you want to change the score of any other URL? (Y/N):
N
The heap after changing scores of any URL!:
Index: 0 Score: 325 URL: https://arstechnica.com/information-technology/
Index: 1 Score: 264 URL: https://arstechnica.com/rss-feeds/
Index: 2 Score: 315 URL: http://video.arstechnica.com/
Index: 3 Score: 226 URL: https://arstechnica.com/tech-policy/
Index: 4 Score: 253 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index: 5 Score: 294 URL: https://arstechnica.com/cars/
Index: 6 Score: 299 URL: https://arstechnica.com/tag/ars-approved/
Index: 7 Score: 213 URL: http://arstechnica.com/?view=archive
Index: 8 Score: 217 URL: https://arstechnica.com/store/product/subscriptions/
Index: 9 Score: 231 URL: https://arstechnica.com/#main
Index: 10 Score: 192 URL: https://arstechnica.com
Index: 11 Score: 274 URL: https://arstechnica.com/#site-menu
Index: 12 Score: 292 URL: https://arstechnica.com/about-us/
Index: 13 Score: 201 URL: https://arstechnica.com/staff-directory/
Index: 14 Score: 294 URL: https://arstechnica.com/reviews/
Index: 15 Score: 209 URL: https://arstechnica.com/?view=mobile
Index: 16 Score: 180 URL: https://arstechnica.com/
Index: 17 Score: 133 URL: http://arstechnica.com/?theme=dark
Index: 18 Score: 182 URL: https://arstechnica.com/science/
Index: 19 Score: 189 URL: https://arstechnica.com/features/
Index: 20 Score: 231 URL: https://arstechnica.com/civis/
Index: 21 Score: 167 URL: https://arstechnica.com/contact-us/
Index: 22 Score: 90 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index: 23 Score: 273 URL: http://arstechnica.com/?view=grid
Index: 24 Score: 195 URL: http://arstechnica.com/?theme=light
Index: 25 Score: 168 URL: https://arstechnica.com/search/
Index: 26 Score: 200 URL: https://arstechnica.com/advertise-with-us/
Index: 27 Score: 130 URL: https://arstechnica.com/gadgets/
Index: 28 Score: 70 URL: https://arstechnica.com/reprints/
Index: 29 Score: 208 URL: https://arstechnica.com/gaming/
```

We ask the user if they want to change the score of any URL. The user wants to change the score of the last index (last URL) to have a score of 325. The score 325 is the new highest score out of all the previous scores, so it should be on the top. If we look at the new arraylist, we can see that is true. This new arraylist also follows the heap structure.

```
We are testing the maxHeapInsert method!
The website name is MikeWu.com and the score is the max integer!
The 31 URLs after we add a new URL!:
Index: 0 Score: 2147483647 URL: https://www.MikeWu.com/
Index: 1 Score: 264 URL: https://arstechnica.com/rss-feeds/
Index: 2 Score: 325 URL: https://arstechnica.com/information-technology/
Index: 3 Score: 226 URL: https://arstechnica.com/tech-policy/
Index: 4 Score: 253 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index: 5 Score: 294 URL: https://arstechnica.com/cars/
Index: 6 Score: 315 URL: http://video.arstechnica.com/
Index: 7 Score: 213 URL: http://arstechnica.com/?view=archive
Index: 8 Score: 217 URL: https://arstechnica.com/store/product/subscriptions/
Index: 9 Score: 231 URL: https://arstechnica.com/#main
Index: 10 Score: 192 URL: https://arstechnica.com
Index: 11 Score: 274 URL: https://arstechnica.com/#site-menu
Index: 12 Score: 292 URL: https://arstechnica.com/about-us/
Index: 13 Score: 201 URL: https://arstechnica.com/staff-directory/
Index: 14 Score: 299 URL: https://arstechnica.com/tag/ars-approved/
Index: 15 Score: 209 URL: https://arstechnica.com/?view=mobile
Index: 16 Score: 180 URL: https://arstechnica.com/
Index: 17 Score: 133 URL: http://arstechnica.com/?theme=dark
Index: 18 Score: 182 URL: https://arstechnica.com/science/
Index: 19 Score: 189 URL: https://arstechnica.com/features/
Index: 20 Score: 231 URL: https://arstechnica.com/civis/
Index: 21 Score: 167 URL: https://arstechnica.com/contact-us/
Index: 22 Score: 90 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index: 23 Score: 273 URL: http://arstechnica.com/?view=grid
Index: 24 Score: 195 URL: http://arstechnica.com/?theme=light
Index: 25 Score: 168 URL: https://arstechnica.com/search/
Index: 26 Score: 200 URL: https://arstechnica.com/advertise-with-us/
Index: 27 Score: 130 URL: https://arstechnica.com/gadgets/
Index: 28 Score: 70 URL: https://arstechnica.com/reprints/
Index: 29 Score: 208 URL: https://arstechnica.com/gaming/
Index: 30 Score: 294 URL: https://arstechnica.com/reviews/
```

We are testing the maxHeapInsert method on the website MikeWu.com with a score of the max integer. We then print out the new arraylist. We should expect the MikeWu.com website to be on the top of this heap because it has the largest score, we should also see that there are now 31 websites, and we should also see that this arraylist is still in heap structure. This is true, so the maxHeapInsert method worked.

```
We are testing the heapMaximum method which should return MikeWu's website and score!
The max node is: Score: 2147483647 URL: https://www.MikeWu.com/
```

We are testing the heapMaximum method and we know from our previous testing that MikeWu's website should be at the top. We can see this is true therefore the method works.

```
The Top 10 Search Results:
Index: 0 PageRank: 1 Score: 2147483647 URL: https://www.MikeWu.com/
Index: 1 PageRank: 2 Score: 325 URL: https://arstechnica.com/information-technology/
Index: 2 PageRank: 3 Score: 315 URL: http://video.arstechnica.com/
Index: 3 PageRank: 4 Score: 299 URL: https://arstechnica.com/tag/ars-approved/
Index: 4 PageRank: 5 Score: 294 URL: https://arstechnica.com/cars/
Index: 5 PageRank: 6 Score: 294 URL: https://arstechnica.com/reviews/
Index: 6 PageRank: 7 Score: 292 URL: https://arstechnica.com/about-us/
Index: 7 PageRank: 8 Score: 274 URL: https://arstechnica.com/#site-menu
Index: 8 PageRank: 9 Score: 273 URL: http://arstechnica.com/?view=grid
Index: 9 PageRank: 10 Score: 264 URL: https://arstechnica.com/rss-feeds/
```

We are displaying the top 10 results.

```
Do you want to search for any other word? (Y/N):
Y
What keyword do you want to search?:
technology

**Visiting** Received web page at https://arstechnica.com/
Found (212) links
Searching for the word technology...

**Visiting** Received web page at https://arstechnica.com/#main
Found (212) links
Searching for the word technology...
```

We ask user if they want to search for any other word. If they type Y, we continue. In this case they typed technology as new keyword. The rest of the program runs the same as before so I will just leave that part out.

```
Do you want to search for any other word? (Y/N):
Y
What keyword do you want to search?:
technology
```

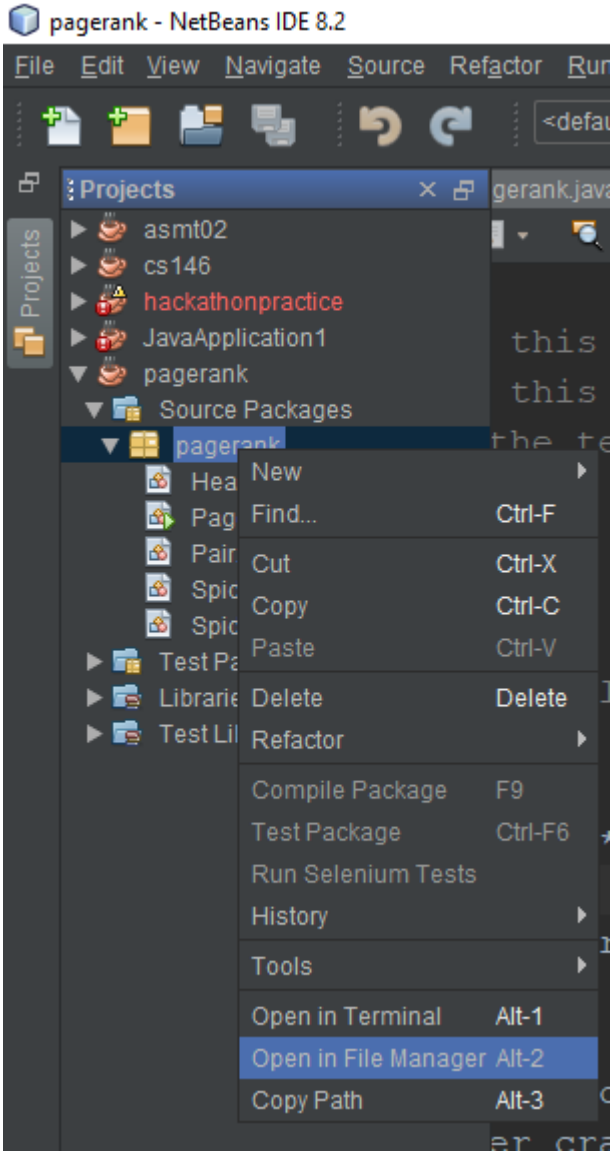
We test the program when it has the same search word. There are now 2 same keywords, and when it prints out the top 10 unique search words, notice how it doesn't have any words that repeat because they are unique.

```
The Top 10 Search Results:
Index: 0 PageRank: 1 Score: 2147483647 URL: https://www.MikeWu.com/
Index: 1 PageRank: 2 Score: 304 URL: https://arstechnica.com/about-us/
Index: 2 PageRank: 3 Score: 301 URL: https://arstechnica.com/tag/ars-approved/
Index: 3 PageRank: 4 Score: 285 URL: https://arstechnica.com/cars/
Index: 4 PageRank: 5 Score: 285 URL: https://arstechnica.com/advertise-with-us/
Index: 5 PageRank: 6 Score: 266 URL: https://arstechnica.com/search/
Index: 6 PageRank: 7 Score: 262 URL: https://arstechnica.com/#main
Index: 7 PageRank: 8 Score: 239 URL: https://arstechnica.com/#site-menu
Index: 8 PageRank: 9 Score: 239 URL: https://arstechnica.com
Index: 9 PageRank: 10 Score: 235 URL: https://arstechnica.com/science/

Do you want to search for any other word? (Y/N):
N
The Top 10 Unique Searches:
science
technology
```

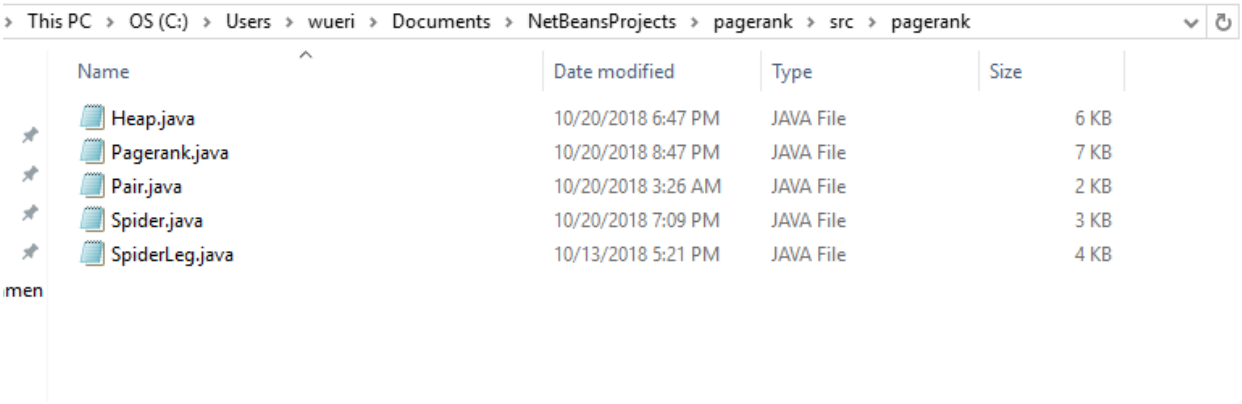
As we can see, this search keyword (technology) displays the top 10 as this. We again ask user if they want to search for another word. In this case, user types N. We finish the program and wrap it up by printing out the top 10 unique searches. Since the user only input 2 unique keywords, we only have 2 unique keywords to show for it which is science and technology.

4)The procedure (step by step) of how to unzip your files, install your application, and run/test your codes.

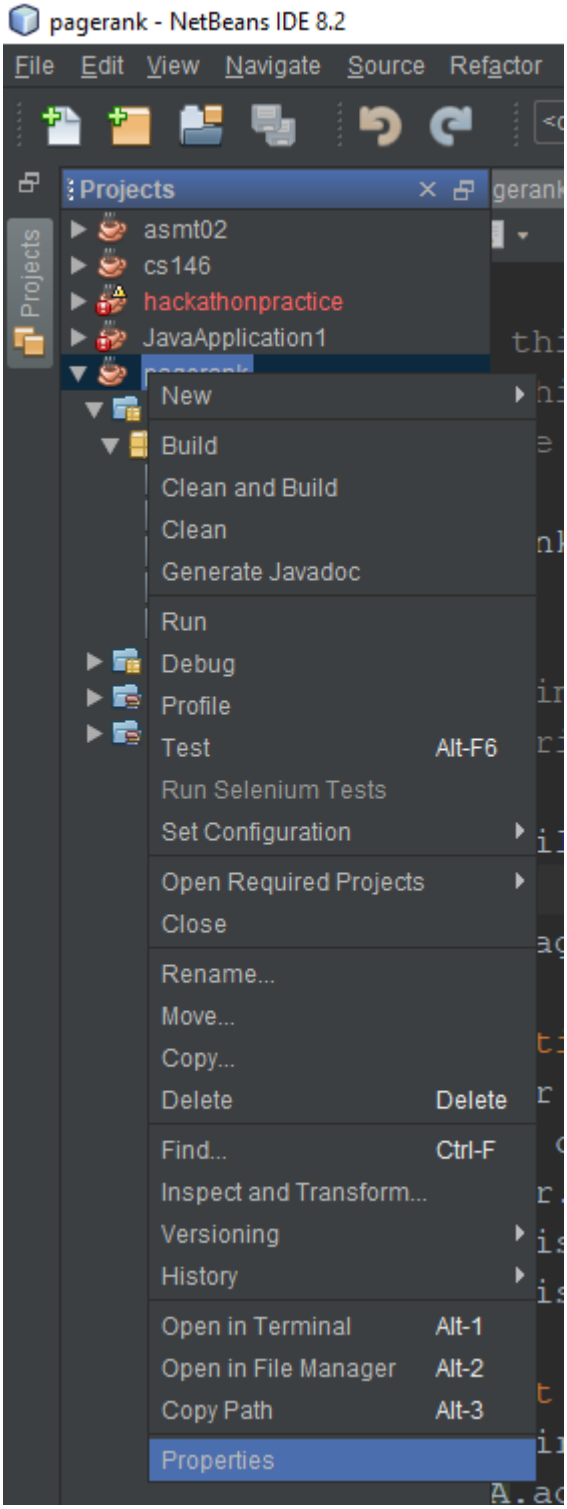
I am using netbeans to write my code. First begin by unzipping the files. Drag out everything from the folder and drag it into your desktop. Then open up your IDE's file path.



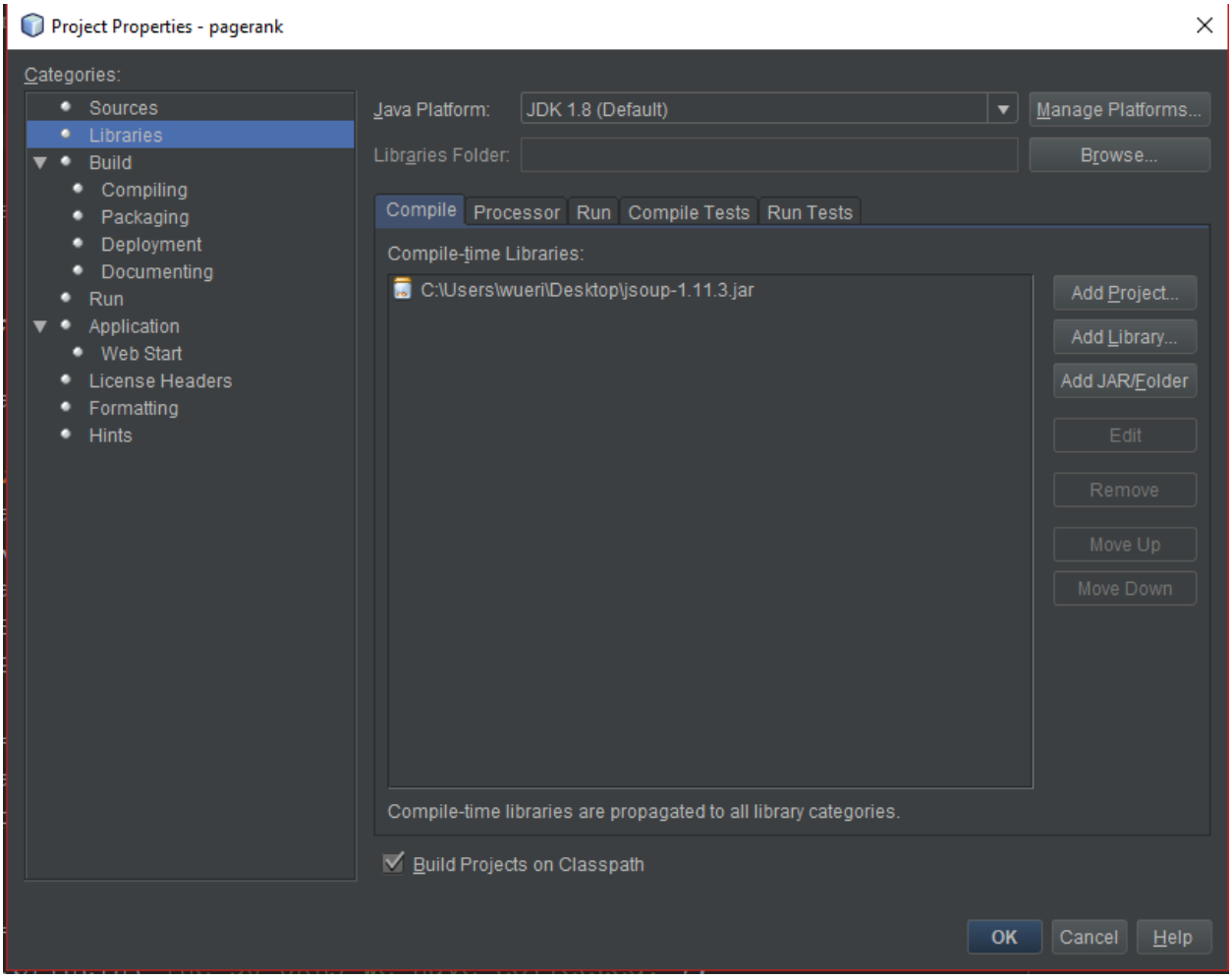
Then drag everything from the code folder into the file path.



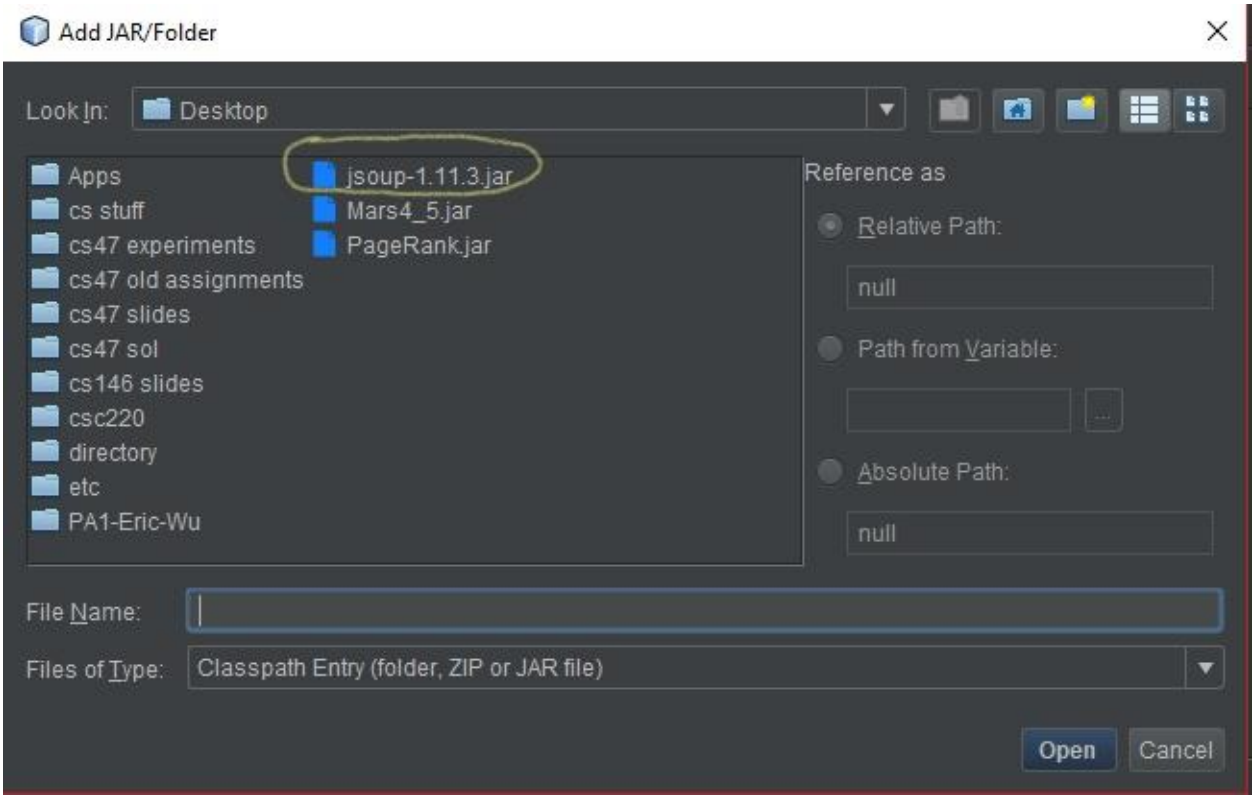
Right click on the project and click on properties.



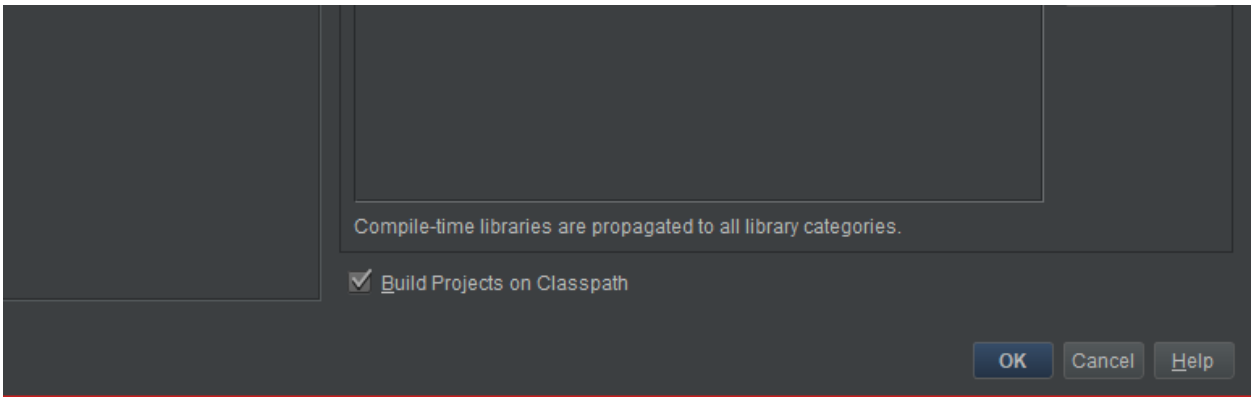
Click on libraries on the left hand side and then click on the Add JAR/Folder on the right side.



Select the jsoup-1.11.3.jar file that you have unzipped along with the code. It can be found in the desktop if unzipped the files to the desktop.



Then click on the Ok button.



You are done with setting up the program! Now proceed to run the main class, Pagerank.java.

Overview on how to use the program

The program begins by asking user to input a keyword.

```
run:
What keyword do you want to search?:
science
```

The program then ask user if they want to change the score of any URL. If you decide to, you can type Y and then proceed to enter the index of the URL you want to change and the new score. This step will repeat until you type N.

```
Do you want to change the score of any URL? (Y/N):
Y
Enter the Index of the URL that you want to change the score:
29
Enter a new score higher than 168 :
325
Do you want to change the score of any other URL? (Y/N):
N
```

The program then ask you if you want to search for any other word. If you type Y, it will repeat the steps over again, but if you type N, the program finishes up, by printing out the top 10 unique searches.

```
Do you want to search for any other word? (Y/N):
Y
What keyword do you want to search?:
technology
```

```
Do you want to search for any other word? (Y/N):
N
The Top 10 Unique Searches:
science
technology
```

5) Problems encountered during the implementation

I have encountered many problems while trying to write this program. The first problem I encountered was trying to make the WebCrawler work. It turned out that the WebCrawler had an excess line that wasn't needed, which was a break statement. This made it so that it would end the program if we find a keyword on the URL page. I went to a tutor to ask why the WebCrawler doesn't work and he went over the program line by line until we found this part. Another problem I have consistently encountered is when I converted the Heap algorithm from the pseudo code from the book, I always had index out of bounds error. To fix this, I had to meticulously run the code through my head until I find out that I only needed to either increase or decrease a number by 1. Another problem I had was about structure of this assignment. I had to email my professor many times and asked in class so that he could clarify the problem to me. Another problem I had was about how to store the URLs and scores together. At first I decided to try using a hash map but I decided that was too complicated for me because there were several problems I couldn't solve. The data when put into a heap wouldn't be stable and we might also encounter some problems if there was two URLs with the same score. Then I decided to create a new object that would store the URL and score as its properties. In this case the data would always be stable because each object is unique and there won't be any problem if there is a duplicate score in the URLs. I have encountered many problems while trying to finish the programming assignment but I hope that I can learn from the mistakes I have made and use that experience and apply it to other projects I will be working on in the future.

6) Lessons Learned

I have learned many things from doing this programming assignment. First it would be time management. I have procrastinated on completing this assignment and that turned out to be a bad idea. For the past few days, I have been getting little sleep because I am trying to finish the project and also juggle with work from other classes. The next time I am given an assignment, I will start it as soon as possible, so if I have any major bugs with the program I will have time to fix it and also ask my professor for help. I would also be less stressed out because I should be doing a decent amount of work on the assignment each day over the course of several week and not all in several days. I also learned that there are many things on the internet that we could take advantage of like the WebCrawler. I do not need to make my own WebCrawler because it has been made already by some other person. This teaches me about the power of collaboration. If I had been working on this programming from scratch, including making my own WebCrawler, I would probably not finish this until the end of the year. But because I am using the WebCrawler that someone else has made, I can finish this programming assignment on time. Another thing I learned was that I need to practice my programming skills. It has been a long time since I have touched java, so this assignment helped me refresh my java skills. I guess I could work on projects alongside with classes so that I could always keep my programming knowledge sharp.