Programming Assignment 1: Google Search Engine

Jen Luu

CS 146

March 24, 2019

1. **Design and Implementation**
    a. The primary data structure in the Google search engine is an ArrayList. ArrayLists are dynamic, which means that they can be altered as needed. Throughout the project, website entries may be added to an existing array during a search; therefore, it is important to have a flexible data structure that can expand to accommodate those changes. Additionally, the ArrayList can be used to hold integers or website objects, where a website consists of a Page Rank score and a URL. As such, they are declared as ArrayList<Integer> and ArrayList<Website>. Overall, using an ArrayList provides a versatile way to store a collection of data.
2. **Classes/subroutines/function calls:**
    a. **HeapFunctions**
        i. The purpose of HeapFunctions is to organize all of the methods that are related to heapsort and heap priority queue. Each method has a common purpose, which is to provide the foundation for sorting the Page Rank scores.
            1. **public void maxHeapify(ArrayList<Integer> A, int i)**
                a. Given an ArrayList of integers and an index i to specify which node to sort, this method checks to see if the node i obeys the max heap property: Parent(i) >= child(i). This method ensures that a single node i follows this rule and calls on itself recursively.
            2. **public void buildMaxHeap(ArrayList<Integer> A)**

   a. The input is an ArrayList of integers and turns it into a max heap. In the previous method, the max heap property is applied to a single node i recursively. However, in this method, the max heap property is applied to an entire array. Meaning, first entry or root of an array is the largest element.

3. **public void heapsort(ArrayList<Integer A)**
   a. Given an unsorted ArrayList of integers, heapsort will put the elements in order. This method uses buildMaxHeap to create a max heap first. Then, the sorting will take place as the root of the ArrayList will be moved to the last index. Each time this happens, the current heap will decrease by one node until a single node remains. That single node or root will become the first element in the sorted array.
4. **public void heapIncreaseKey(ArrayList<Integer> A, int i, int key)**
   a. This method allows the user to increase the content of some node i. The key is some value that is used for comparison purposes regarding the max heap property.
5. **public void maxHeapInsert(ArrayList<Integer> A, int key)**
   a. This method takes in a key value to insert a new element into the max heap. It calls heapIncreaseKey to maintain the max heap property after adding to the heap.

6. **public int heapMaximum(ArrayList<Integer> A)**
    a. This method returns the root, or the first element, in a max heap.
7. **public int heapExtract(ArrayList<Integer> A)**
    a. Similar to the previous method, this one returns the root of a max heap. However, it also removes the root by decreasing the heap size by 1. It calls upon maxHeapify to maintain the max heap property.

b. **WebCrawler**
    i. **public WebCrawler(String aKeyword)**
        1. A WebCrawler objected is created when it is given a keyword to search for.
    ii. **public void search()**
        1. This method prompts the WebCrawler to search the web for URLS that are relevant to the keyword.
    iii. **public Set<String> getUrls()**
        1. The WebCrawler will return a list of URLS from its search in a collection of strings.

c. **Website**
    i. **public Website(String url)**
        1. The constructor for the Website class will initialize a Website object given a URL. Additionally, the PageRank score and its four

factors will be randomly calculated and attached to the Website object as well.

ii. **public Website()**

    1. This is a second constructor that takes in nothing. It used to create objects of the Website class for testing purposes.

iii. **public String getUrl()**

    1. This method serves as a getter method to return the URL of a Website object.

iv. **public int getPageRankSum()**

    1. This method retrieves the PageRank score of a Website object.

v. **Public void heapPriorityQueue(ArrayList<Website> urlLinks)**

    1. Given an ArrayList of Website objects, this method will create an ArrayList<Integer> from the objects' PageRank scores. The method will call upon heapsort to sort the scores and create a priority queue based on the highest score. Afterwards, the ArrayList<Website> will be updated according to the indexes of the ArrayList<Integer> that holds the sorted scores.

vi. **Public void insertNewWebUrl(Website w, int newPageRank)**

    1. This method will let users insert a new web URL into the heap based on its Page Rank score. A new ArrayList<Integer> will be created and sorted based on the original Websites' and new website's PageRank scores. The function will call upon maxHeapInsert to insert the new PageRank score. Afterwards, the

original ArrayList<Website> will be updated based on the order of the PageRank scores.

vii. **Public String viewFirstWebUrl(ArrayList<Website w)**

1. Given an ArrayList<Website>, the user will be able to see the first ranked URL in the heap. To start, a new ArrayList<Integer> will created with the PageRank scores of the Website objects. The method will call upon heapExtractMax to locate the element with the first ranked URl, which will be the one with the highest PageRank score. Based on that, the method will locate the URL with the highest PageRank score to the one found from heapExtractMax and return it.

viii. **Public void increasePageRangeScore(int indexOfUrl, int money)**

1. This method will allow users to choose a single URL link and increase its priority by paying more money for more exposure. In other words, since the payment factor will be increased the overall PageRank score of that URL will increase as well. As a result, that URL will move up in the heap. The method will create an ArrayList<Integer> from the PageRank scores of a pre-existing ArrayList<Website>. That ArrayList<Integer>, the index of the desired URL, and the sum of money will be used in heapIncreaseKey. Afterwards, the pre-existing ArrayList<Website> will be updated based on the order of the PageRank scores.

d. **PageRank**

   i. **public PageRank()**

      1. There are four factors that make up a Website's PageRank score. These include the frequency of a keyword, which is how many times it appears within the web page, the age of the webpage, the number of other pages that reference the web page, and how much money an owner has paid for exposure. For simplicity purposes, each factor is created by Math.random() * 100 + 1. The original Math.random() only generates a random number from 0-1. It must be multiplied by 100 to increase the range from 0-99. It is also necessary to add 1 to shift the range from 1-100. Afterwards, the PageRank is created by adding the four factors.

   ii. **public int getPageRankSum()**

      1. This is a getter method that returns the current PageRank score of a website.

   iii. **public int getPayment()**

      1. This is a getter method that returns the current payment factor of a website.

3. **Self-testing screen shots**

   a. **heapsort**

```
public static void main(String[] args)
{
    ArrayList<Integer> heapTester = new ArrayList<Integer>();
    heapTester.add(14);
    heapTester.add(4);
    heapTester.add(2);
    heapTester.add(3);
    heapTester.add(9);
    heapTester.add(10);


    HeapFunctions example = new HeapFunctions();
    System.out.println("Heapsort");
    System.out.println("Before: " + heapTester);
    example.heapsort(heapTester);
    System.out.println("Expected: [2, 3, 4, 9, 10, 14]");
    System.out.println("Results: " + heapTester);
```

i.

```
Heapsort
Before: [14, 4, 2, 3, 9, 10]
Expected: [2, 3, 4, 9, 10, 14]
Results: [2, 3, 4, 9, 10, 14]
```

ii.

iii. I created a tester ArrayList<Integer>, a new HeapFunctions object, and called heapsort. I compared the results to the unsorted ArrayList<Integer> and the expectations. Since heapsort works, then it also indicates that maxHeapify and buildMaxHeap works as well.

b. extractMaxHeap

```
public static void main(String[] args)
{
    ArrayList<Integer> heapTester = new ArrayList<Integer>();
    heapTester.add(14);
    heapTester.add(4);
    heapTester.add(2);
    heapTester.add(3);
    heapTester.add(9);
    heapTester.add(10);


    HeapFunctions example = new HeapFunctions();

    example.buildMaxHeap(heapTester);
    System.out.println("Before: " + heapTester);

    System.out.println(example.heapExtractMax(heapTester));
    example.heapExtractMax(heapTester);
    System.out.println("After: " + heapTester);
```

i.

ii.

```
Before: [14, 9, 10, 3, 4, 2]
14
After: [9, 4, 2, 3, 4, 2]
```

iii. As shown in this snippet, I created a tester ArrayList<Integer>, called buildMaxHeap, and heapExractMax. The results showed that the max integer, which is 14, was removed from the heap and returned.

c. maxHeapInsert

```
public static void main(String[] args)
{
    ArrayList<Integer> heapTester = new ArrayList<Integer>();
    heapTester.add(14);
    heapTester.add(4);
    heapTester.add(2);
    heapTester.add(3);
    heapTester.add(9);
    heapTester.add(10);


    HeapFunctions example = new HeapFunctions();
    example.buildMaxHeap(heapTester);
    System.out.println("Before: " + heapTester);


    example.maxHeapInsert(heapTester, 8);
    System.out.println("After: " + heapTester);
    System.out.println("Expected: [14, 9, 10, 3, 4, 8, 2]");

}
```

i.

```
Before: [14, 9, 10, 3, 4, 2]
After: [14, 9, 10, 3, 4, 8, 2]
Expected: [14, 9, 10, 3, 4, 8, 2]
```

ii.

iii. To test maxheapInsert, I made an ArrayList<Integer> and called buildMaxHeap to apply the max heap property to it. Afterwards, I used that ArrayList<Integer> to put into maxHeapInsert and set the key = 8. In the result, I drew out a max heap tree of the results and expectations and saw that they matched each other.

d. WebCrawler

i.

```
Enter keyword
dogs

**Visiting** Received web page at https://google.com/search?q=dogs&num=80
Found (268) links
Searching for the word dogs...
**Success** Word dogs found at https://google.com/search?q=dogs&num=80
**Done** Visited 86 web page(s)
```

ii. I made a Scanner object to prompt the user to enter in a keyword and used that as input for the WebCrawler.

e. 30 URLs with PageRank scores

i.

```
Here are the first 30 URL links:

1.www.fox4news.com/news/police, PageRank score: 310
2.www.awrestaurants.com/food/, PageRank score: 293
3.www.nydailynews.com/news/na, PageRank score: 285
4.tricountyhumanesociety.org/a, PageRank score: 265
5.www.acctphilly.org/adopt/ava, PageRank score: 263
6.www.hamiltonhumane.com/adopt, PageRank score: 254
7.www.peggyadams.org/adopt-a-, PageRank score: 254
8.www.hamiltonhumane.com/adopt, PageRank score: 254
9.www.peggyadams.org/adopt-a-, PageRank score: 254
10.www.healthypawspetinsurance, PageRank score: 250
11.nevadahumanesociety.org/ado, PageRank score: 248
12.www.citruscritters.com/dog-, PageRank score: 243
13.www.austinhumanesociety.org/, PageRank score: 230
14.www.banfield.com/pet-health, PageRank score: 228
15.people.com/pets/golden-retr, PageRank score: 217
16.www.cdc.gov/healthypets/pet, PageRank score: 213
17.www.smbc-comics.com/comic/d, PageRank score: 209
18.www.petsmart.com/dog/dental, PageRank score: 206
19.indyhumane.org/adoptable-do, PageRank score: 203
20.www.purina.com/dogs&sa=U&ve, PageRank score: 202
21.en.wikipedia.org/wiki/Dog%2, PageRank score: 198
22.www.humanerescuealliance.org, PageRank score: 193
23.www.webmd.com/pets/dogs/def, PageRank score: 191
24.en.wikipedia.org/wiki/Dog&s, PageRank score: 169
25.www.akc.org/&sa=U&ved=0ahUK, PageRank score: 167
26.multcopets.org/adoptable/do, PageRank score: 164
27.berkeleyhumane.org/adopt-a-, PageRank score: 155
28.www.slsc.org/omnimax-films/, PageRank score: 136
29.learningenglish.voanews.com, PageRank score: 133
30.www.tailshumanesociety.org/, PageRank score: 100
```

ii. After retrieving the URLs from the WebCrawler, I created website objects while also generating a PageRank score to each one. Afterwards, I

gathered the PageRank scores and put them into an ArrayList<Integer> to call heapsort on.

f. heapPriorityQueue

i.

```
//Store the first sorted 20 out of 30 web url links into the heap
public ArrayList<String> heapPriorityQueue(ArrayList<Website> urlLinks)
{
    //Create an integer to store 20 page rank scores
    ArrayList<Integer> sorted = new ArrayList<Integer>();
    for (int i = 0; i < 20; i++)
    {
        sorted.add(urlLinks.get(i).getPageRankSum());
    }

    //Create a heapfunctions object
    HeapFunctions heap = new HeapFunctions();
    //Sort the page rank scores
    heap.heapsort(sorted);

    //Create a new ArrayList<String> to hold the updated urls
    ArrayList<String> updated = new ArrayList<String>();

    for (int i = 0; i < 20; i++)
    {
        //Use the index of the sorted pageranks to match with the index of the original website's pagerank
        if (sorted.get(i) != urlLinks.get(i).getPageRankSum())
        {
            //if the current pagerank does not match with the original, traverse the arraylist until it is found
            for (int j = 0; j < 20; j++)
            {
                if (sorted.get(i) == urlLinks.get(j).getPageRankSum())
                {
                    updated.add(urlLinks.get(j).getUrl());
                }
            }
        }
        else
        {
            updated.add(urlLinks.get(i).getUrl());
        }
    }
    return updated;
```

ii.

```
Website example = new Website();
ArrayList<String> twentyURLS = new ArrayList<String>(example.heapPriorityQueue(websites));
System.out.println("Here are the 20 URLS in the priority queue: ");
for (int i = 0; i <20; i++)
{
    System.out.println(i+1 + ". " + twentyURLS.get(i));
}
```

```
Here are the 20 URLS in the priority queue:
1. www.cmtsj.org/&sa=U&ved=0ah
2. weather.com/weather/tenday/
3. www.operasj.org/&sa=U&ved=0
4. www.applevacations.com/san-j
5. www.sjmuni.com/&sa=U&ved=0a
6. knightfoundation.org/commun
7. www.sjpl.org/&sa=U&ved=0ahU
8. sjsuspartans.com/&sa=U&ved=
9. en.wikipedia.org/wiki/Downt
10. www.stubhub.com/ncaa-tourna
11. realestate.usnews.com/place
12. www.sanjoseca.gov/&sa=U&ved=
13. www.creatvsj.org/&sa=U&ved=
14. sanjosetheaters.org/&sa=U&v
15. www.eventbrite.com/e/2019-s
16. www.wunderground.com/weathe
17. www.sanjoseca.gov/index.aspx
18. sanjosejazz.org/&sa=U&ved=0
19. www.sanjose.org/&sa=U&ved=0
20. www.japantownsanjose.org/&s
```

iii.

iv. To test this method, I used another WebCrawler and keyword to search for URLS and created an ArrayList<Website> based off of them. In addition, made another ArrayList<Integer> to store 20 of the PageRank scores and called heapsort to categorize them based on the priority of the scores.

g. viewFirstWebUrl

i.

```
1.southwesthumane.org/adopt/c, PageRank score: 296
2.www.buzzfeed.com/katangus/t, PageRank score: 284
3.www.michiganhumane.org/cats, PageRank score: 280
4.www.i-cat.com/&sa=U&ved=0ah, PageRank score: 277
5.www.cat.com/&sa=U&ved=0ahUK, PageRank score: 255
6.www.oregonhumane.org/adopt/, PageRank score: 248
7.www.seattlehumane.org/adopt, PageRank score: 242
8.en.wikipedia.org/wiki/Cat&s, PageRank score: 235
9.www.chewy.com/b/cat-325&sa=, PageRank score: 217
10.www.animalhumanesociety.org, PageRank score: 212
11.www.walmart.com/cp/cat-supp, PageRank score: 210
12.anticruelty.org/adopt-a-new, PageRank score: 209
13.www.washingtonpost.com/scie, PageRank score: 206
14.www.reddit.com/r/cats/&sa=U, PageRank score: 195
15.www.catphones.com/&sa=U&ved, PageRank score: 177
16.twitter.com/cats&sa=U&ved=0, PageRank score: 172
17.www.cat.com/en_ZA.html&sa=U, PageRank score: 168
18.www.humanesociety.org/anima, PageRank score: 168
19.www.cat.com/en_ZA.html&sa=U, PageRank score: 168
20.www.humanesociety.org/anima, PageRank score: 168
21.www.marketwatch.com/investi, PageRank score: 162
22.en.wiktionary.org/wiki/cat&, PageRank score: 157
23.www.caterpillar.com/&sa=U&v, PageRank score: 141
24.tenor.com/search/cat-gifs&s, PageRank score: 140
25.icraeastbay.org/adoption/gar, PageRank score: 131
26.www.thesprucepets.com/cats-, PageRank score: 123
27.www.wihumane.org/adopt&sa=U, PageRank score: 118
28.www.theguardian.com/lifeand, PageRank score: 117
29.mashable.com/category/cats/, PageRank score: 92
30.www.nj.com/south/2019/03/ca, PageRank score: 78
Root: www.nj.com/south/2019/03/ca
```

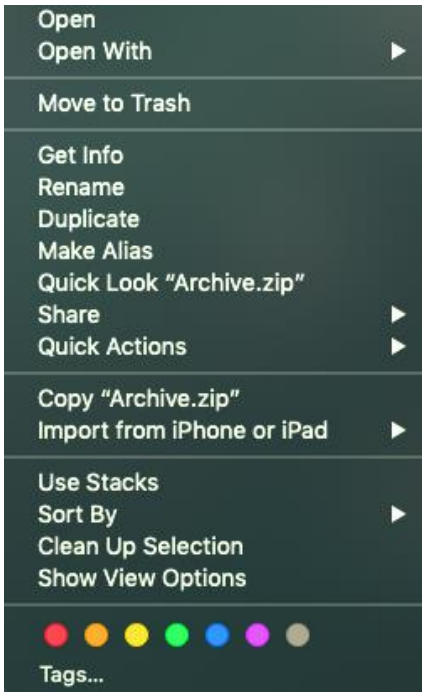
ii. To test this method, I used the ArrayList<Website> that has already been sorted as input. If this was only the max heap, then the max element would be at the root, where the PageRank score is the highest. Conversely, this is ArrayList<Website> is already sorted then the max element or root would be where the PageRank score is at the lowest.

4. **How to launch the program**

    a. Begin with the zip file labeled: “PA1-Section 7-Luu.zip”

    b. Right click it and this menu should show:

    i.



    ii. Click “open”

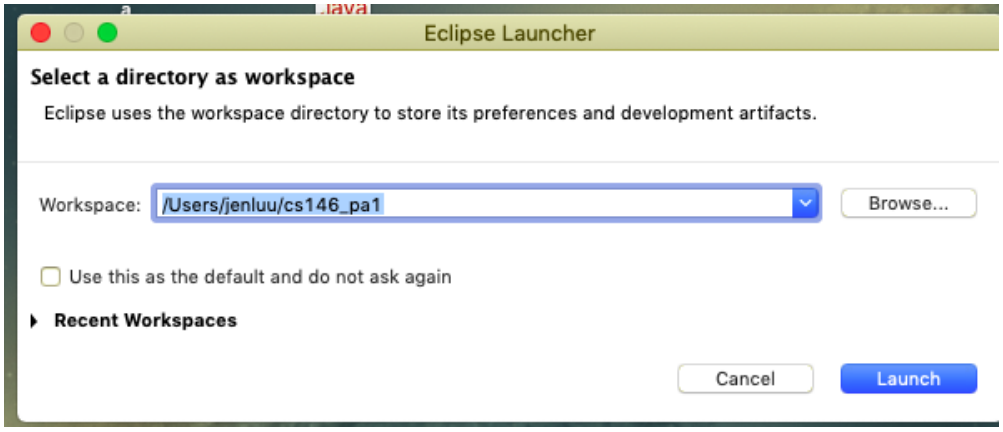
    iii. It should contain the following:

        1. PA1_JenLuu.jar
        2. HeapFunctions.java
        3. Website.java
        4. PageRank.java
        5. WebCrawler.java
        6. Testing.java

    c. Open the program Eclipse

d. Select a workspace

i.



e. Select the top left-hand corner icon of a webpage with a plus

f. Select “Project”

g. Select “Java Project”

h. Give it any name

i. Select “Open Perspective”

j. Right click on the project

k. Click “Build Path”

l. Go to “Configure Build Path”

m. Go to “Libraries” and click on “Modulepath”

n. Click “Add External JARs…”

o. Locate the jar file and add it

p. Go to the Website class and click run

5. **Problems encountered**

a. One of the most difficult struggles was accommodating for incompatible argument input and return types. For instance, a Website object must consist of a URL and a PageRank score. It is crucial to keep them together for organization

purposes. However, the heapsort and heap priority queue methods take in ArrayList<Integer> as their arguments, not ArrayList<Website>. This was challenging because the only way to sort the websites was by their PageRank scores. Therefore, the PageRank score and the URL of a website had to be separated during sorting.

   i. I came up with a solution by taking a break from coding and thinking about it conceptually. To start, I wrote down the argument type for the heapsort and heap priority queue methods to understand what data type my input had to be, which was an ArrayList<Integer>. Afterwards, I wrote down what my current data types, which was an ArrayList<Website>. I had to think about how what common factors these two types of ArrayList had and how to bridge them together. I realized that I could extract the PageRank scores from the website objects and put them into a separate ArrayList<Integer> for sorting.
   ii. Once the sorting was done, I knew that the indexes of the PageRank scores correlated with the correct order. I created another ArrayList<Website> and used the order of the PageRank scores to fill it with the website objects from the original ArrayList<Website>.

b. Another common issue I faced was an array index out of bounds exceptions, especially for heapsort. To solve this, I created print statements throughout the body of my methods to pinpoint where the problem occurred. When I was able to locate a specific area, I drew out the heap to have a visual representation of what I was working with. Additionally, I drew out the steps of my expected outcome and

compared that to what was happening in the code. Once I realized which part of my code did not match up with my expectations, I was able to exactly locate the error and fix the problem.

6. **Lessons learned**

   a. The hardest part about coding is not actually writing code, it is about fixing bugs. There were many times when I thought I had a clear understanding of a method. I transferred my thoughts into code, saw that it compiled, but the results were far from my expectations. As such, the majority of my time was spent fixing bugs and it is crucial not to underestimate the time it takes for that.

   b. Additionally, creating successful projects requires a complete understanding of what is happening. Writing code with no plan or structure is meaningless. Code is a direct translation of one's though; therefore, if the thought process is unclear, then it will manifest in the program.

   c. Moreover, a short method may seem simple but there are multiple concepts working at once. For instance, maxHeapInsert has very few lines but it requires a solid understanding of how an ArrayList works, the index of it, heapsize, and how the heap changes as input is added.