CMPT 435 - Fall 2021 - Dr. Labouseur

Assignment Three – LaTeX Searching and Hash Table

Tyler Vultaggio

October 2021

1 Searches

For this assignment we did searching, where you only need to get one item from a long list of things. To do this two different searching algorithms: linear search and binary search.

2 Comparisons

Search Name — Comparisons — Asymptotic Running time:
Linear Search — 334.67 — 0(n)
Binary Search — 8.24 — 0(log n)

3 Linear Search

Linear search is just about the most basic and straight forward way to search. You simply iterate through a list until you find the item you are looking for. This understandably makes the best case scenario a constant time if the first item is what you are looking for, while the average and worst case are both n.

3.1 Linear Search

```
1 public class LinearSearch
2 {
3         public static int linSearch(String SortedItems[], String target)
4         {
```

```
5                   int comparisons = 0;
6
7                   for(int i = 0; i < SortedItems.length; i++)
8                   {
9                           comparisons++;
10                          if(SortedItems[i].equalsIgnoreCase(target))
11                          {
12                                  //System.out.println(comparisons);
13                                  return comparisons;
14                          }
15
16                  }
17                  return comparisons;
18          }
19
20
21          public static void printArray(String[] printableArray)
22          {
23                  for(int i = 0; i <      printableArray.length; i++)
24                  {
25                          System.out.println(     printableArray[i]);
26                  }
27          }
28
29          public static void main(String[] args) throws FileNotFoundException
30          {
31                  double totalCompares = 0;
32                  double averageCompares = 0.00;
33                  String[] TempArray = new String[666];
34                  TempArray = myFileReader.fileArray();
35                  String[] TargetArray = new String[42];
36                  for(int i = 0; i < TargetArray.length; i++)
37                  {
38                          TargetArray[i] = TempArray[i];
39                  }
40
41                  //printArray(TargetArray);
42                  TempArray = MergeSort.mSort(TempArray, 0, TempArray.length-1);
43
44                  for(int i = 0; i < TargetArray.length; i++)
45                  {
46                          totalCompares = totalCompares + linSearch(TempArray, TargetArray[i]);
47                  }
48
49                  averageCompares = totalCompares/TargetArray.length;
50                  double roundedAverageCompares = (double) (Math.round(averageCompares*100.0)/10
51                  System.out.println("This is the number of total comparisons: " + totalCompares
52                  System.out.println("This is the average number of comparisons: " + roundedAver
53
54
55          }
56
57  }
```

4 Binary Search

Divide and conquer once again takes the cake in terms of quickly doing things and doing them especially efficiently. In this particular case though, there is a prerequisite: the list must be sorted before it can be searched. This causes a bit more time than a basic linear search as it requires an additional step, but on a pre-sorted list will leave linear search in the dust with a worst case of log n which also happens to be its average case as well. With a best case of constant as well, it makes it the faster sort.

4.1 Binary Search

```
1  public class BinarySearch
2  {
3
4          public static int binSearch(String sortedArray[], String target)
5          {
6                   int start = 0;
7                   int end = sortedArray.length - 1;
8                   int comparisons = 0;
9                  while (start <= end)
10                 {
11                         comparisons++;
12                         //System.out.println(comparisons);
13                     int position = start + (end - start) / 2;
14
15                     int newtarget = target.compareTo(sortedArray[position]);
16
17                     // Check if newtarget is present at mid
18                     if (newtarget == 0)
19                         return comparisons;
20
21                     // If newtarget greater, ignore left half
22                     if (newtarget > 0)
23                         start = position + 1;
24
25                     // If newtarget is smaller, ignore right half
26                     else
27                         end = position - 1;
28                 }
29
30                 return -1;
31         }
32
33
34         public static void main(String[] args) throws FileNotFoundException
35         {
36                 double totalCompares = 0;
37                 double averageCompares = 0;
38                 String[] TempArray = new String[666];
39                 TempArray = myFileReader.fileArray();
40                 String[] TargetArray = new String[42];
41                 for(int i = 0; i < TargetArray.length; i++)
42                 {
```

```
43                          TargetArray[i] = TempArray[i];
44                  }
45
46                  TempArray = MergeSort.mSort(TempArray, 0, TempArray.length-1);
47
48                  for(int i = 0; i < TargetArray.length; i++)
49                  {
50                          totalCompares = totalCompares + binSearch(TempArray, TargetArray[i]);
51                  }
52
53                  averageCompares = totalCompares/TargetArray.length;
54                  double roundedAverageCompares = (double) (Math.round(averageCompares*100.0)/10
55                  System.out.println("This is the number of total comparisons: " + totalCompares
56                  System.out.println("This is the average number of comparisons: " + roundedAver
57
58          }
59
60  }
```

5 Hashing (with chaining)

Hashing doesn't need a whole lot of comparisons in order to find things, running only about 4 comparisons a search for my hash table. This is in an average of 0(n) which is also it's worst case but can be as good as constant time for lookups. This is because you could theoretically have all of your items have only one hash and essentially just be a linked list, which is far slower. A hash table tends not to have that however and will generally be much faster. For my key I take my cleaned string from my file reader and set that word to all lower case and then I take each letter in the work as its ascii value and add them up and the take the total and mod it by 250 to get the key value. For my add method it puts the first magicItem into the hashTable as a new node and then it makes a linked list off of that first node thus for my get method the first get it does is actually a node that has a magicItem not just a pointer to a linked list. Because of how my add is set up my get is more efficient and my get and my first comparison for the chaining are the same.

6 Comparisons for Hashing

Hashing — Average Comparisons — Asymptotic Running time:
Hashing — 2.45 — 0(n)

6.1 Hashing (with chaining)

```
1  public class HashTable extends HashNode
2  {
3
4          static private HashNode [] hashTable = new HashNode[250];
5          public static double totalComparisons = 0;
```

```
6         public static int comparisons = 0;
7
8         //Method for adding magicItem to the Hash Table
9         //For my add it puts the first magicItem into the hashTable as a new node and then it
10        //thus for my get method the first get it does is actually a node that has a magicItem
11        public static void add(int key, String magicItem)
12        {
13                //int location = key;
14                HashNode newNode;
15                newNode = new HashNode();
16                newNode.magicItem = magicItem;
17                if (hashTable[key] == null)
18                {
19                        //the linked list is still empty and this is the first entry
20                        hashTable[key] = newNode;
21                }
22                else
23                {
24                        HashNode runner; // the runner will go throught the linked list wothou
25                        runner = hashTable[key];
26                        HashNode prevNode = new HashNode();
27                        while (runner!=null)
28                        {
29                                prevNode = runner;
30                                runner = runner.next;
31                        }
32                        runner = newNode;
33                        prevNode.next = runner;
34                        //runner.key=key;
35                        runner.magicItem = magicItem;
36           }
37        }
38
39        //Method for getting a word from the hashtable.
40        //Because of how my add is set up my get is more efficient and my get and my first com
41        static String get(String word)
42        {
43                comparisons = 0;
44        int location = getKey(word);
45        //System.out.println("location value for get is " + location.toString());
46        HashNode runner;
47        runner = hashTable[location];
48        while (runner!=null)
49        {
50                totalComparisons++;
51                comparisons++;
52            if (runner.magicItem.equals(word))
53            {
54                return runner.magicItem;
55            }
56            runner = runner.next;
57        }
58        return null;
59
60    }
61
62
```

```
63
64          //For my key I take my cleaned string from my file reader and set that word to all low
65          //and then I take each letter in the work as its ascii value and add them up and the t
66          public static int getKey(String magicItem)
67          {
68                  int nameLenght = magicItem.length();
69                  int total = 0 ;
70                  int key = 0;
71                  magicItem = magicItem.toLowerCase();
72              for(int i = 0; i < nameLenght ; i++)
73              {
74                  char character = magicItem.charAt(i);
75                  int ascii = (int) character;
76                  total = total + ascii;
77              }
78
79              key = total % 250;
80
81              return key;
82          }
83
84
85          public static void main(String[] args) throws FileNotFoundException
86          {
87                  //double totalCompares = 0;
88                  double averageCompares = 0;
89                  String[] HashArray = new String[666];
90                  HashArray = myFileReader.fileArray();
91                  String[] TargetArray = new String[42];
92                  for(int i = 0; i < TargetArray.length; i++)
93                  {
94                          TargetArray[i] = HashArray[i];
95                          //System.out.println(TargetArray[i] + getKey(TargetArray[i]));
96                  }
97
98                  HashArray = MergeSort.mSort(HashArray, 0, HashArray.length-1);
99
100                 //System.out.println();
101                 for(int i = 0; i < HashArray.length; i++)
102                 {
103                         add(getKey(HashArray[i]), HashArray[i]);
104                 }
105                 for(int i = 0; i < TargetArray.length; i++)
106                 {
107                         System.out.println(get(TargetArray[i]) + ": " + comparisons);
108                 }
109
110                 averageCompares = totalComparisons/TargetArray.length;
111                 double roundedAverageCompares = (double) (Math.round(averageCompares*100.0)/10
112                 System.out.println("This is the number of total comparisons: " + totalComparis
113                 System.out.println("This is the average number of comparisons: " + roundedAver
114
115
116
117
118                 //Test for my Key
119                 //System.out.println(getKey("ringofthemercifulblow"));
```

```
120
121         }
122
123 }
```

7 Hash Node

Here I made a class called Hash node which was used to create the linked lists for the chaining in the hash table.

7.1 Hash Node

```
1  public class HashNode
2  {
3
4          public HashNode next = null;
5          public String magicItem;
6
7          public HashNode ()
8          {
9
10         }
11
12         //This section is for setting methods
13         //-----------------------------------------------
14         public void setNext(HashNode next)
15         {
16                 this.next = next;
17         }
18
19         public void setmagicItem(String magicItem)
20         {
21                 this.magicItem = magicItem;
22         }
23         //-----------------------------------------------
24
25         //This section is for getting methods
26         //-----------------------------------------------
27         public String getmagicItem()
28         {
29                 return magicItem;
30         }
31
32         public HashNode getNext()
33         {
34                 return next;
35         }
36         //-----------------------------------------------
37
38         public static void main(String[] args) throws FileNotFoundException
39         {
40                 // TODO Auto-generated method stub
41
```

```
42          }
43
44  }
```

8 FileReader (plus string cleaning)

The fileReader part takes a text file in this case magicitems.txt and it takes a scanner and a reader and turns each line into a string and puts it into an array once the string is in the array we clean up the string by removing all of the spaces, number, and special characters in the sting.

8.1 FileReader

```
1   public class myFileReader
2   {
3           public static String[] fileArray() throws FileNotFoundException
4           {
5                   Scanner scanner = new Scanner(new File("magicitems.txt"));
6                   String[] myArray = new String[666];
7                   int index = 0;
8
9                   while(scanner.hasNextLine())
10                  {
11                          myArray[index] = scanner.nextLine();
12                          myArray[index] = myArray[index].replaceAll("[0-9+,_()/.]", "");
13                          myArray[index] = myArray[index].replaceAll("\\s", "");
14                          myArray[index] = myArray[index].replaceAll("'", "");
15                          myArray[index] = myArray[index].replaceAll("-", "");
16                          myArray[index] = myArray[index].toLowerCase();
17                          myArray[index] = myArray[index].substring(0,1).toUpperCase() + myArray
18                          index = index + 1;
19                  }
20                  scanner.close();
21
22                  return myArray;
23          }
24
25          public static void main(String[] args) throws IOException
26          {
27
28          }
29
30  }
```