Lab 34-35

Caelan Bratland, Havin Lim

11/23/22

Sources: none

We confirm that the above list of sources is complete AND that we have not talked to anyone else (e.g., CSC 207 students) about the solution to this problem.

Lab 34

CountChars.java

```java
public class CountChars {

    Pair[] pairs;

    public CountChars() {
        this.pairs = new Pair[26];
    }

    /**
     * Returns the index of the char ch in pairs array.
     * @param ch the char key
     * @return the index in pairs array
     */
    public int find(char ch) {
        int start = ch % pairs.length; // hash code
        int idx = start;
        while (true) {
            if (pairs[idx] != null) {
                if (pairs[idx].getKey() == ch) return idx;
                else {
                    idx++;
                    idx %= pairs.length;
                    if (idx == start) {
                        rehash();
                    }
                }
            } else {
                return idx;
            }
        }
    }
```

```java
/**
 * Inserts the key and value pair into the hash map.
 * @param ch the char key
 * @param v the int value
 * @return the old value associated with the key char
 */
public Integer put(char ch, int v) {
    int idx = find(ch);
    Pair old = pairs[idx];
    pairs[idx] = new Pair(ch, v);
    if (old == null) return null;
    else return old.getValue();
}

/**
 * Returns the value associated with key.
 * @param ch the char key
 * @return the value associated with key
 */
public Integer get(char ch) {
    int idx = find(ch);
    Pair pair = pairs[idx];
    if (pair == null) return null;
    else return pair.getValue();
}

/**
 * Doubles the length of the internal array.
 */
private void rehash() {
    Pair[] old = pairs;
    pairs = new Pair[old.length * 2];
    for (int i = 0; i < old.length; ++i) {
        if (old[i] == null) continue;
        put(old[i].getKey(), old[i].getValue());
    }
}

/**
```

```
    * Returns the internal pair array.
    * @return pair array
    */
   public Pair[] getArray() {
       return pairs;
   }
}
```

CountCharsTester.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CountCharsTester {

   public static void main(String[] args) throws FileNotFoundException {
       File file = new File("/usr/share/dict/words");
       Scanner sc = new Scanner(file);
       CountChars map = new CountChars();

       while(sc.hasNextLine()) {
           String str = sc.nextLine();
           for(int i=0; i<str.length(); i++) {
               char ch = str.charAt(i);
               Integer val = map.get(ch);
               if (val == null) {
                   val = 0;
               }
               map.put(ch, val + 1);
           }
       }

       Pair[] array = map.getArray();
       for(int i = 0; i<array.length; i++) {
           if (array[i] == null) continue;
           System.out.println(array[i].getKey() + ": " +
array[i].getValue());
       }

       sc.close();
```

```
    }


}
```

Lab 35

HeapSort.java

```
package heapsort;

public class HeapSort
{
   /** Helper method to print an array */
   public static <AnyType> void printArray(AnyType[] array)
   {
       System.out.print("[");

       for (int i=0 ; i<array.length ; i++) {
           System.out.print(array[i]);
           if (i != array.length - 1) {
               System.out.print(" ");
           }
       }

       System.out.println("]");
   }

   /** Transpose entries at i and j in an array */
   public static <AnyType> void swapReferences(AnyType[] array, int i, int
j)
   {
       AnyType tmp = array[i];
       array[i] = array[j];
       array[j] = tmp;
   }

   /** Heap sort method. Modifies array so that it is in sorted order */
   public static <AnyType extends Comparable<? super AnyType>>
   void heapSort(AnyType[] array)
   {
```

```
        // Build Heap
        for (int i=array.length/2 - 1; i>=0 ; i-- )
        {
            percDown(array, i, array.length);
        }

        for (int i=array.length-1 ; i>0 ; i-- )
        {
            swapReferences( array, 0, i ); // deleteMax
            percDown( array, 0, i); // Maintain heap ordering property
        }

    }

    /** Percolate down method. The object to be percolated is at
     *  array[hole], and the current heap size (number of elements in
     *  the heap) is given. */
    public static <AnyType extends Comparable<? super AnyType>>
    void percDown(AnyType[] array, int hole, int size)
    {
        int child;
        AnyType tmp = array[hole];

        for(; hole * 2 + 1 < size; hole = child ) {
            child = hole * 2 + 1;
            if (child + 1 != size && array[child +
1].compareTo(array[child]) > 0)
                child++;
            if (array[child].compareTo(tmp) > 0)
                array[hole] = array[child];
            else
                break;
        }
        array[hole] = tmp;
    }

    public static final int SAMPLE_SIZE = 30;
    public static final int SAMPLE_RANGE = 100;

    /** Test driver method. */
```

```
    public static void main(String[] args)
    {
        int[] testSizes = { 2, 3, 7 };
        for (int i = 0; i < testSizes.length; ++i) {
            Integer array[] = new Integer[testSizes[i]];

            for (int j=0 ; j<array.length ; j++)
                array[j] = (int)(SAMPLE_RANGE*Math.random());

            System.out.println("Before (size " + testSizes[i] + "): ");
            printArray(array);

            heapSort(array);

            System.out.println("After (size " + testSizes[i] + "): ");
            printArray(array);
        }
    }


}
```

Answers

1. Why is array[hole] stored in tmp?

Tmp is the value we're starting with, and the goal of percDown is to find the correct place to put it. At the end, it assigns the tmp value to the correct place.

2. What does the following test condition of the for-loop do? hole * 2 <= currentSize

It checks if there is a child to move to. If it's false, then we've reached the bottom of the heap to a node with no children.

3. What is the purpose of the following assignment statement? child = hole * 2;

We're moving to the next child. Hole * 2 is the position of the left child of hole.

4. What is it meant if the following test condition of the if-statement is false? child != currentSize

If child is equal to the current size, then there is no right node of the parent.

5. What does child++ do?

Moves from the left child to the right child node.

6. What do the two calls to the compare method accomplish?

The first one determines the smaller value of either the left or right child nodes, and the second one checks if the smaller child value is less than the value we're moving.

7. Is the element stored in array[hole] lost after executing the following statement? array[hole] = tmp;

No, because in the previous for loop, hole is set to child, and the child value was moved to the previous hole.

8. What does the break statement imply?

If the child is larger than the value we're moving (tmp), then that means we've found the final place to put it.

Output

Before (size 2):

[27 51]

After (size 2):

[27 51]

Before (size 3):

[83 87 34]

After (size 3):

[34 83 87]

Before (size 7):

[13 91 18 56 33 55 47]

After (size 7):

[13 18 33 47 55 56 91]