323.35 bucket sorter java

Matthew Flammia, 23661371

Due 12/10/20 6pm

Cover page

Source Code

```
import java.util.NoSuchElementException;
import java.util.Scanner;
import java.io.*;
import java.io.PrintStream;

/**
* Matthew Flammia, 23661371
* CSCI 323.35 Project 3
* To execute, have the source data file in working directory.
* args[0] is the source data file
* args[1] is the final output file name
* args[2] is the observations file name
* Example run command:
* java -cp bin FlammiaM_Project1_Java RadixSortString_Data1.txt outFile1.txt outFile2.txt
**/

public class FlammiaM_Project3_Java{
	public static void main(String[] args) throws Exception{
		//file setup step 0
		File data = new File(args[0]);
		//hashtable creation
		hashTable sorter = new hashTable();
		//creation of stack step 1
		sorter.firstReading(data);
		LLStack inputData = sorter.loadStack(data);

		//step 5
		inputData.printStack(args[2]);
		//first input of data step 6, 7
		sorter.currentPosition = sorter.longestStringLength - 1;
		sorter.moveStack(inputData, sorter.currentPosition, sorter.currentTable);

		//updating table information step 8
		sorter.currentPosition = sorter.currentPosition - 1;
		//moving from table 0 to 1 step 9
		while(sorter.currentPosition >= 0){
			for(int i=0;i<256;i++){
				if(!sorter.hashTable[sorter.currentTable][i].isEmpty()){

sorter.moveNextTable(sorter.hashTable[sorter.currentTable][i],sorter.currentPosition,sorter.next
Table);
				}
			}
```

```
                        sorter.currentTable = sorter.tableIndex();
                        sorter.nextTable = sorter.tableIndex();
                        sorter.printTable(args[2]);
                        sorter.currentPosition = sorter.currentPosition - 1;
                }
                //PRINT SORTED DATA
                sorter.printSortedData(args[1]);
        }
}

class listNode{
        //variables
        String data;
        listNode next;
        //constructors
        listNode(String data){
                this.data = data;
        }
        //methods
        void printNode() throws NullPointerException{
                try{
                        System.out.print("("+this.data+", "+this.next.data+")->");
                }
                catch (Exception e){
                        System.out.print("("+this.data+", NULL)->");
                }
        }
}

class LLStack{
        //variables
        listNode top;
        //constructors
        LLStack(){
                this.top = new listNode("dummyNode");
                this.top.next = null;
        }
        //methods
        void push(listNode node){
                node.next = this.top;
                this.top = node;
        }
        listNode pop(){
                //checks if empty; then returns null if is
```

```
		if(isEmpty()){
			return null;
		}
		//makes a temp node, stores the value of top, then sets new top to top.next
		listNode temp;
		temp = this.top;
		this.top = this.top.next;
		return temp;
	}
	boolean isEmpty(){
		//checks if stack is empty
		if(this.top.next==null){
			return true;
		}
		return false;
	}
	void printStack(String outFile2) throws FileNotFoundException{
		PrintStream fileOut = new PrintStream(new FileOutputStream(outFile2,true));
		System.setOut(fileOut);
		System.out.println("Printing Stack...");
		listNode readHead = this.top;
		while(readHead.next != null){
			readHead.printNode();
			readHead = readHead.next;
		}
		System.out.print("\n");
		fileOut.close();
	}
}

class LLQueue{
	//variables
	listNode head;
	listNode tail;
	listNode dummy;
	//methods
	LLQueue(){
		//constructor
		dummy = new listNode("dummyNode");
		head = new listNode("head");
		tail = new listNode("tail");
		head.next = dummy;
		tail.next = dummy;
	}
```

```
void insertQ(listNode node){
	//special case for first node insert
	if(this.isEmpty()){
		dummy.next = node;
		tail.next = node;
	}
	//case for all other nodes after first
	else{
		tail.next.next = node;
		tail.next = node;
	}
}
listNode deleteQ(){
	//prevents deleting empty queue
	if(isEmpty()){
		return null;
	}
	//special case for single node in queue
	if(dummy.next == tail.next){
		listNode temp = dummy.next;
		tail.next = dummy;
		dummy.next = null;
		return temp;
	}
	//generic node removal
	else{
		listNode temp = dummy.next;
		dummy.next = dummy.next.next;
		return temp;
	}
}
boolean isEmpty(){
	if(tail.next == dummy){
		return true;
	}
	return false;
}
void printQueue(int table, int index, String outFile2) throws FileNotFoundException{
	System.out.print("Table["+table+"]["+index+"]: ");
	listNode readHead = this.head.next;
	while(readHead.next!=this.tail.next){
		readHead.printNode();
		readHead = readHead.next;
	}
```

```
                readHead.printNode();
                readHead.next.printNode();
                System.out.print("NULL\n");
        }
}
class hashTable{
        LLQueue[][] hashTable;
        int currentTable;
        int nextTable;
        int longestStringLength;
        int currentPosition;
        //constructor
        hashTable(){
                this.hashTable = new LLQueue[2][256];
                for(int i=0;i<2;i++){
                        for(int j=0;j<256;j++){
                                this.hashTable[i][j] = new LLQueue();
                        }
                }
                this.currentTable = 0;
                this.nextTable = 1;
                this.longestStringLength = 0;
                this.currentPosition = 0;
        }

        //methods
        void firstReading(File inputData) throws FileNotFoundException{
                this.longestStringLength = 0;
                Scanner reader = new Scanner(inputData);
                while(reader.hasNext()){
                        String temp = reader.next();
                        if(this.longestStringLength < temp.length()){
                                this.longestStringLength = temp.length();
                        }
                }
        }
        LLStack loadStack(File inputData) throws FileNotFoundException{
                LLStack stack = new LLStack();
                Scanner reader = new Scanner(inputData);
                while(reader.hasNext()){
                        String temp = padString(reader.next());
                        stack.push(new listNode(temp));
                }
                return stack;
```

```
	}
	void moveStack(LLStack stack, int currentPosition, int currentTable){
		while(!stack.isEmpty()){
			listNode current = stack.pop();
			char currentCharacter = getChar(current, currentPosition);
			int hashIndex = currentCharacter;
			this.hashTable[currentTable][hashIndex].insertQ(current);
		}
	}
	void moveNextTable(LLQueue queue, int currentPosition, int nextTable){
		while(!queue.isEmpty()){
			listNode current = queue.deleteQ();
			current.next = null;
			char currentCharacter = getChar(current, currentPosition);
			int hashIndex = currentCharacter;
			this.hashTable[nextTable][hashIndex].insertQ(current);
		}
	}
	int tableIndex(){
		int table = (this.currentTable + 1) % 2;
		return table;
	}
	char getChar(listNode input, int currentPosition){
		String temp = input.data;
		char current = temp.charAt(currentPosition);
		return current;
	}
	String padString(String data){
		while(data.length()!=this.longestStringLength){
			data = data + " ";
		}
		return data;
	}
	void printTable(String outFile2) throws FileNotFoundException{
		PrintStream fileOut = new PrintStream(new FileOutputStream(outFile2,true));
		System.setOut(fileOut);
		System.out.println("Printing Table...");
		for(int i=0;i<256;i++){
			if(!this.hashTable[this.currentTable][i].isEmpty()){

this.hashTable[this.currentTable][i].printQueue(this.currentTable,i,"outFile1.txt");
			}
		}
		fileOut.close();
```

```
	}
	void printSortedData(String outFile1) throws FileNotFoundException{
		PrintStream fileOut = new PrintStream(new FileOutputStream(outFile1,true));
		System.setOut(fileOut);
		for(int i=0;i<256;i++){
			if(!this.hashTable[this.currentTable][i].isEmpty()){
				while(!this.hashTable[this.currentTable][i].isEmpty()){
					listNode temp =
this.hashTable[this.currentTable][i].deleteQ();
					System.out.println(temp.data);
				}
			}
		}
		fileOut.close();
	}
}
```

Output:

```
4
5
7
8
8
8
8
8
11
11
12
12
12
12
13
13
14
14
15
15
15
15
15
15
16
16
```

17
19
19
21
21
21
21
21
21
22
23
25
26
26
27
29
29
29
30
31
32
33
37
38
38
38
38
41
42