Project 3

CSCI 230

Jin Y Choi

Email – jchoi101@student.mtsac.edu

May 9th, 2018

Development Environment

System -- PC with Ubuntu 16.04
Compiler – IntelliJ IDEA

Table of Contents

1. Program Notes
2. Results and Discussion
3. Source Code
4. Run Time Output

Program Notes:

The status of the program is complete, and the extra credit for the improved Huffman Coding Algorithm is done.

The first portion using the pattern matching schemes and ouputing relevant data was rather straight forward especially with code provided from the book. The main learning experience was the unique jump mechanisms that BM and KMP employed, last character and fail functions respectively. The last character function stored the last occurrence of certain function so that you could properly jump characters within the text that didn't match to the character where there was as potential match. The fail function computed the proper shifting of the pattern when there was a failed match. Being able to see and trace the code first hand to make sure my output was correct was a valuable learning experience.

The second portion of the project regarding the Huffman Coding provided many new learning materials. The way Huffman Coding handled the conversion of characters into a bit representation via binary tree representation provided efficiency. When I was doing the decompression scheme, I found that reconstructing a tree and utilizing the log n search time for each character was more efficient that running multiple pattern matching attempts from the different possible character and bitrep pairs. My analysis was that while one run through of the pattern would be O(n+m) where n is text length and m is pattern length, worst case you'd have to try it m number of times so it seemed like O(m(n+m)). The reconstruction of the Huffman Coding Tree and subsequent searches required roughly O(mlogn) runtime where n is number of nodes in the tree and m is the number of characters. The analysis showed that utilizing the tree was far more efficient.

I also learned about file size and the link it has to bytes and bits. I noticed the "compression" process we used didn't exactly make the file size smaller but much bigger and this had to do with the fact that each character of the string in the text file took up 1 byte. Through the implementation of extra credit's improved huffman coding, I was able to learn that within the code you can write byte arrays to a file which is then stored in a different manner than text. Initially I tried to utilize the same BufferedWriter but found that this handles text files and for file outputs (such as type .dat) it was correct to utilize a stream. I used a simple scheme to find the correct byte representation of the and got the outputted file size down to the required 7 bytes.

Results and Discussions:

	Algorithm	Pattern	Runtime (ms)	Total comp	Avg comp	Index match
usdeclarPC	BM	america	2.99	37	0.22	169
		aeiou	1.89	2101	0.24	N/A
		waging	1.55	1177	0.2	5777
		dissolutions	1.31	376	0.12	3256
	KMP	america	0.03	182	1.08	169
		aeiou	0.58	9169	1.06	N/A
		waging	0.36	5841	1.01	5777
		dissolutions	0.2	3363	1.03	3256
humanDNA	BM	agagagtaaaaaa	1.82	295	0.19	1568
		aactattctctg	9.19	5493	0.36	15439
		tagtac	2.74	1105	0.92	1204
		agtcgtacxagtcg	2.4	8203	0.53	N/A
	KMP	agagagtaaaaaa	0.14	2119	1.35	1568
		aactattctctg	1.73	19761	1.28	15439
		tagtac	0.1	1469	1.22	1204
		agtcgtacxagtcg	1.54	20232	1.31	N/A

As expected, the KMP match was more efficient than the BM. This is due to the fact that the book code for the BM matching is a simplified version and runs worst case O(nm) while KMP runs at O(n+m). Although the computed runtime does show the difference, the actual time it takes to process is so short, the millisecond differences that showed at various runs fluctuated.
The total and average comparison counts were interesting, as BM performed far fewer average comparisons than the KMP yet the runtime took longer. This I suspect has to do with the computation of jump using Math.min and the HashMap. The oddity in runtime was also present for finding a pattern relatively close to the start of the text as opposed to something that wasn't even within the text. Expected runtime should be longer for having to search the entire text as opposed to just a small fraction of it but the runtime skewed the other way. This is mainly due to length of the pattern, if there is a mismatch found, there are bigger jumps in the text. I suspect it also has to do with simple fluctuations in computer background systems and the outputted runtimes are so negligibly small it outputs skewed data.
For the second portion using huffman coding, the output and compression/decompression works as stated. This portion was more challenging in terms of implementation but less material to discuss results of. The only factor to note is that the improved huffman coding did produce the .dat file with a size of 7 bytes.

Source Code:

```java
/*  Java Class: AbstractMatch.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: Abstract Class that holds the data for the two matching schemes.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
public abstract class AbstractMatch {
    protected int comparisons;
    protected double runTime;
    private String type;
    protected String pattern;
    public AbstractMatch(String type){
        comparisons = 0;
        runTime = 0;
        this.type = type;
    }
    public int getComparisons() {
        return comparisons;
    }
    public double getRunTime() {return runTime/1000000.0; }
    public void resetComp() { comparisons = 0; }
    public void resetRunTime() {runTime = 0; }
    public String generateReport(int index, int length){
        String result;
        double avg = 0.0;
        if (index == -1) { avg = comparisons/(double)length; }
        else { avg = comparisons/(double)index;}
        result = "Pattern: " + pattern + "\nAlgorithm: " + type + "\nRun Time: " +
runTime/1000000.0 + "ms" + "\nTotal Comparisons: " + comparisons + "\nAverage comparisons: "
+ avg;
        if (index == -1){ result = result + "\nMatch not found"; }
        else { result = result + "\nMatch found at index: " + index; }
        result = result + "\n";
        resetComp();
        resetRunTime();
        return result;
    }
    public abstract int find(String pattern, String text);
}

/*  Java Class: BMMatch.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: Boyer Moore algorithm from the book, modified to output the data required
for project.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
/*  Java Class: BMMatch.java
    Author: Jin Choi & Henry Nguyen
    Class: CSCI 230
    Date: April 18, 2018
    Description: The book source code for the boyer moore pattern matching.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
import java.util.HashMap;
import java.util.Map;
public class BMMatch extends AbstractMatch{
    public BMMatch(){
        super("Boyer-Moore");
    }
    public int find(String text, String pattern){
        this.pattern = pattern;
        long startTime = System.nanoTime();
        int result = findinternal(text,pattern);
        long endTime = System.nanoTime();
        runTime = endTime - startTime;
        return result;
    }
    private int findinternal(String text, String pattern) {
        int n = text.length();
        int m = pattern.length();
        if (m == 0) {
            return 0;                           // trivial search for empty string
        }
        Map<Character,Integer> last = new HashMap<>();   // the 'last' map
        for (int i=0; i < n; i++)
            last.put(text.charAt(i), -1);               // set -1 as default for all text
characters
        for (int k=0; k < m; k++)
            last.put(pattern.charAt(k), k);             // rightmost occurrence in pattern
is last
        // start with the end of the pattern aligned at index m-1 of the text
        int i = m-1;                                     // an index into the text
        int k = m-1;                                     // an index into the pattern
        while (i < n) {
            if (text.charAt(i) == pattern.charAt(k)) {                  // a matching
character
                if (k == 0) {
                    return i;                        // entire pattern has been found
                }
                i--;                                         // otherwise, examine previous
                k--;                                         // characters of text/pattern
            } else {
                i += m - Math.min(k, 1 + last.get(text.charAt(i))); // case analysis for
jump step
                k = m - 1;                                   // restart at end of pattern
            }
            comparisons++;
        }
        return -1;                                       // pattern was never found
    }
}

/*  Java Class: KMPMatch.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: Knutt-Morris-Pratt algorithm from the book, modified to output the data
required for project.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
/*  Java Class: KMPMatch.java
    Author: Jin Choi & Henry Nguyen
    Class: CSCI 230
    Date: April 18, 2018
    Description: The book source code for the KMP pattern matching.
    I certify that the code below is my own work.
   Exception(s): N/A
```

```java
*/
public class KMPMatch extends AbstractMatch{
    public KMPMatch(){
        super("Knuth-Morris-Pratt");
    }
    public int find(String text, String pattern){
        this.pattern = pattern;
        long startTime = System.nanoTime();
        int result = findinternal(text,pattern);
        long endTime = System.nanoTime();
        runTime = endTime - startTime;
        return result;
    }
    private int findinternal(String text, String pattern) {
        int n = text.length();
        int m = pattern.length();
        if (m == 0) {
            return 0;                            // trivial search for empty string
        }
        int[] fail = computeFailKMP(pattern);            // computed by private utility
        int j = 0;                                       // index into text
        int k = 0;                                       // index into pattern
        while (j < n) {
            if (text.charAt(j) == pattern.charAt(k)) {       // pattern[0..k] matched thus
far
                if (k == m - 1) {
                    return j - m + 1;            // match is complete
                }
                j++;                                         // otherwise, try to extend
match
                k++;
            } else if (k > 0)
                k = fail[k-1];                               // reuse suffix of P[0..k-1]
            else { j++; }
            comparisons++;
        }
        return -1;                                       // reached end without match
    }
    private int[] computeFailKMP(String pattern) {
        int m = pattern.length();
        int[] fail = new int[m];                         // by default, all overlaps are
zero
        int j = 1;
        int k = 0;
        while (j < m) {                                  // compute fail[j] during this
pass, if nonzero
            if (pattern.charAt(j) == pattern.charAt(k)) {                // k + 1
characters match thus far
                fail[j] = k + 1;
                j++;
                k++;
            } else if (k > 0)                              // k follows a matching prefix
                k = fail[k-1];
            else                                           // no match found starting at j
                j++;
        }
        return fail;
    }
}

/*  Java Class: PatternMatchTester.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: The tester class that tests various patterns in and not in the text for
both the usdelcarPC.txt and humanDNA.txt.
    I certify that the code below is my own work.
   Exception(s): N/A
screes
*/
import java.io.*;
import java.util.stream.Stream;
public class PatternMatchTester {
    public static void main(String[] args) {
        try{
            FileReader usDeclar = new
FileReader("/home/john/Documents/CSCI230/usdeclarPC.txt");
            FileReader humanDNA = new
FileReader("/home/john/Documents/CSCI230/humanDNA.txt");
            BMMatch BM = new BMMatch();
            KMPMatch KMP = new KMPMatch();
            String USDec = toStringB(usDeclar).toString().toLowerCase();
            String DNA = toStringB(humanDNA).toString().toLowerCase();
            int index = BM.find(USDec, "america");
            String USDecResult = BM.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = BM.find(USDec, "aeiou");
            USDecResult = BM.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = BM.find(USDec, "waging");
            USDecResult = BM.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = BM.find(USDec, "dissolutions");
            USDecResult = BM.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = KMP.find(USDec, "america");
            USDecResult = KMP.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = KMP.find(USDec, "aeiou");
            USDecResult = KMP.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = KMP.find(USDec, "waging");
            USDecResult = KMP.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = KMP.find(USDec, "dissolutions");
            USDecResult = KMP.generateReport(index, USDec.length());
            System.out.println(USDecResult);
            index = BM.find(DNA, "agagagtaaaaaa");
            String DNAResult = BM.generateReport(index, DNA.length());
            System.out.println(DNAResult);
            index = BM.find(DNA, "aactattctctg");
            DNAResult = BM.generateReport(index, DNA.length());
            System.out.println(DNAResult);
            index = BM.find(DNA, "tagtac");
            DNAResult = BM.generateReport(index, DNA.length());
            System.out.println(DNAResult);
            index = BM.find(DNA, "agtcgtacxagtcg");
            DNAResult = BM.generateReport(index, DNA.length());
            System.out.println(DNAResult);
            index = KMP.find(DNA, "agagagtaaaaaa");
            DNAResult = KMP.generateReport(index, DNA.length());
            System.out.println(DNAResult);
            index = KMP.find(DNA, "aactattctctg");
            DNAResult = KMP.generateReport(index, DNA.length());
            System.out.println(DNAResult);
            index = KMP.find(DNA, "tagtac");
            DNAResult = KMP.generateReport(index, DNA.length());
            System.out.println(DNAResult);
```

```java
            index = KMP.find(DNA, "agtcgtacxagtcg");
            DNAResult = KMP.generateReport(index, DNA.length());
            System.out.println(DNAResult);
        }catch(IOException e){
            e.printStackTrace();
        }
    }
    private static StringBuilder toStringB(FileReader file){
        StringBuilder result = new StringBuilder();
        BufferedReader reader = new BufferedReader(file);
        Stream<String> stream = reader.lines();
        Object[] temp = stream.toArray();
        for(Object s : temp){
            result.append(s.toString());
            result.append(System.getProperty("line.separator"));
        }
        return result;
    }
}


/*  Java Class: HuffmanCoding.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: Main class for the Huffman Coding algorithm.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
import java.text.DecimalFormat;
import java.util.*;
public class HuffmanCoding {
    private HuffmanTree tree;
    private TreeMap<Character,Integer> frequency;
    private PriorityQueue<HuffmanTree> priq;
    private String bitRep;
    private int numChar;
    private int totalBits;
    public String getBitRep() {
        return bitRep;
    }
    public void setBitRep(String bitRep) {
        if (bitRep.length() > 8){
            StringBuilder temp = new StringBuilder(bitRep);
            for (int i = 8; i < bitRep.length();i+=9){
                temp.insert(i,' ');
            }
            this.bitRep = temp.toString();
        }
        else{
            this.bitRep = bitRep;
        }
    }
    public HuffmanCoding(){
        priq = new PriorityQueue(new NodeComp());
        frequency = new TreeMap<>();
    }
    public String compress(String s){
        setBitRep("");
        numChar = s.length();
        freq(s);
        treeify();
        Set set = frequency.entrySet();
        String result = tree.computeBits();
        result = result + "*****\nNumber of characters: " + numChar + "\nNumber of bits: ";
        String bitRep = "";
        for (Character c : s.toCharArray()){
            TreeNode node = tree.findCharInLeaves(c);
            String bit = node.getBitRep();
            totalBits += node.getNumBits();
            bitRep = bitRep + bit;
        }
        result = result + totalBits;
        setBitRep(bitRep);
        return result;
    }
    public String decompress(String s){
        String result = "";
            HuffmanTree DeCompTree = new HuffmanTree();
            String[] arr = s.split("\n");
            int ref = 0;
            for (int i = 0; i < arr.length; i++){
                if (arr[i].equals("*****")){
                    ref = i + 1;
                    break;
                }
                String[] innerArr = arr[i].split(" ");
                if (innerArr[0].equals("\\n")){
                    innerArr[0] = "\n";
                }
                if (innerArr[0].equals("\u2423")){
                    innerArr[0] = " ";
                }
                String character = innerArr[0];
                String bitPath = innerArr[1];
                TreeNode curr = DeCompTree.getRoot();
                for (int j = 0; j < bitPath.length(); j++){
                    if (bitPath.charAt(j) == '0'){
                        if (curr.getLeft() == null){
                            DeCompTree.addLeft(curr, new TreeNode(0));
                        }
                        curr = curr.getLeft();
                    }
                    if (bitPath.charAt(j) == '1'){
                        if (curr.getRight() == null){
                            DeCompTree.addRight(curr,new TreeNode(1));
                        }
                        curr = curr.getRight();
                    }
                    if (j == bitPath.length()-1){
                        curr.setVal(character.charAt(0));
                    }
                }
            }
            int numChar = 0;
            int numBits = 0;
            String bits = "";
            for (int k = ref; k < arr.length; k++){
                String str = arr[k];
                str = str.replaceAll("[^0-9]+", "");
                if (k == ref){
                    numChar = Integer.parseInt(str);
                }
                if (k == ref+1){
                    numBits = Integer.parseInt(str);
                }
                if (k == arr.length-1){
                    bits = str;
                }
            }
            TreeNode finder = DeCompTree.getRoot();
            for (int i = 0; i < bits.length();i++){
                Character num = bits.charAt(i);
                if (num == '0'){
                    finder = finder.getLeft();
                }
                if (num == '1'){
                    finder = finder.getRight();
                }
                if (finder.isLeaf()){
                    result = result + finder.getVal();
                    finder = DeCompTree.getRoot();
                }
            }
        return result;
    }
    private void freq(String s){
        char[] temp = s.toCharArray();
        Arrays.sort(temp);
        for (char c : temp){
            if (frequency.get(c) != null){
                frequency.put(c,frequency.get(c) + 1);
            }
            frequency.putIfAbsent(c, 1);
        }
    }
    private void treeify(){
        Set set = frequency.entrySet();
        for (Object p : set){
            Map.Entry curr = (Map.Entry)p;
            priq.add(new HuffmanTree(new TreeNode((Integer)curr.getValue(),
(Character)curr.getKey())));
        }
        while (priq.size() > 1){
            HuffmanTree T1 = priq.poll();
            HuffmanTree T2 = priq.poll();
            TreeNode T1Root = T1.getRoot();
            TreeNode T2Root = T2.getRoot();
            HuffmanTree T = new HuffmanTree(new TreeNode(T1Root.getKey()
+T2Root.getKey(),null));
            T.getRoot().setLeft(T1Root);
            T.getRoot().setRight(T2Root);
            T.setSize(T.getSize() + T1.getSize() + T2.getSize());
            T1Root.setParent(T.getRoot());
            T2Root.setParent(T.getRoot());
            priq.add(T);
        }
        tree = priq.remove();
    }
    private static class NodeComp implements Comparator<HuffmanTree> {
        @Override
        public int compare(HuffmanTree o1, HuffmanTree o2) {
            if (o1.getRoot().getKey() > o2.getRoot().getKey()){
                return 1;
            }
            else if (o1.getRoot().getKey() == o2.getRoot().getKey()){
                return 0;
            }
            else{
                return -1;
            }
        }
    }
    public HuffmanTree getTree() {
        return tree;
    }
    public TreeMap<Character, Integer> getFrequency() {
        return frequency;
    }
    public int getNumChar() {
        return numChar;
    }
    public int getTotalBits() {
        return totalBits;
    }
    public void setTotalBits(int totalBits) {
        this.totalBits = totalBits;
    }
}

/*  Java Class: HuffmanTester.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: The driver class for the Huffman Coding. Implemented the improved huffman
coding extra credit by outputting moneyOut without the bit representation and instead
storing it as bytes within the moneyCompress.dat file.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
import java.io.*;
import java.util.stream.Stream;
public class HuffmanTester {
    public static void main(String[] args) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter("/home/john/Documents/CSCI230/result.txt",true))) {
            FileReader moneyIn = new FileReader("/home/john/Documents/CSCI230/moneyIn.txt");
            FileReader moneyOut = new
FileReader("/home/john/Documents/CSCI230/moneyOut.txt");
            String money = toStringB(moneyIn).toString();
            HuffmanCoding huffman = new HuffmanCoding();
            String test = huffman.compress(money);
            String compressionText = "The result of compression: \n";
            System.out.println(compressionText + test + "\n");
            writer.append(compressionText + test);
            writer.newLine();
            try(BufferedOutputStream datWriter = new BufferedOutputStream(new
FileOutputStream("/home/john/Documents/CSCI230/moneyCompress.dat"))){
                String[] bytes = huffman.getBitRep().split(" ");
                byte[] output = new byte[bytes.length];
                for (int i = 0; i < bytes.length;i++){
                    output[i] = ByteSize(bytes[i]);
                }
                datWriter.write(output);
            }
            catch(IOException e){
                e.printStackTrace();
            }
            money = toStringB(moneyOut).toString();
            test = huffman.decompress(money);
            String decompressionText = "The result of decompression: \n";
            System.out.println(decompressionText + test + "\n");
            writer.append(decompressionText + test);
            writer.newLine();
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
```

```java
        }
    }
    private static StringBuilder toStringB(FileReader file){
        StringBuilder result = new StringBuilder();
        BufferedReader reader = new BufferedReader(file);
        Stream<String> stream = reader.lines();
        Object[] temp = stream.toArray();
        for(int i = 0; i < temp.length;i++){
            result.append(temp[i].toString());
            if (i != temp.length-1){
                result.append(System.getProperty("line.separator"));
            }
        }
        return result;
    }
    private static byte ByteSize(String s){
        byte result;
        result = (byte)Integer.parseInt(s,2);
        return result;
    }
}

/*  Java Class: HuffmanTree.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: A simple class that represents the binary tree produced by the Huffman
Coding.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
import java.util.ArrayList;
import java.util.Queue;
import java.util.concurrent.ArrayBlockingQueue;
public class HuffmanTree {
    private TreeNode root;
    private int size;
    private ArrayList<TreeNode> leaves = new ArrayList<>();
    public HuffmanTree(){
        root = new TreeNode(0);
        size = 1;
    }
    public HuffmanTree(TreeNode t){
        root = t;
        size = 1;
    }
    public void findLeaves(TreeNode curr){
        if (curr.isLeaf()){
            leaves.add(curr);
        }
        if (curr.getLeft() != null){
            findLeaves(curr.getLeft());
        }
        if (curr.getRight() != null){
            findLeaves(curr.getRight());
        }
    }
    public TreeNode find(TreeNode node, Character c){
        if (node.getVal() == c){
            return node;
        }
        if (node.getLeft() != null){
            find(node.getLeft(),c);
        }
        if (node.getRight() != null){
            find(node.getRight(),c);
        }
        return null;
    }
    public TreeNode findCharInLeaves(Character c){
        TreeNode result = null;
        for (TreeNode node : leaves){
            if (node.getVal() == c){
                result = node;
                break;
            }
        }
        return result;
    }
    public void addLeft(TreeNode parent, TreeNode child){
        size++;
        parent.setLeft(child);
        child.setParent(parent);
    }
    public void addRight(TreeNode parent, TreeNode child){
        size++;
        parent.setRight(child);
        child.setParent(parent);
    }
    public String computeBits(){
        findLeaves(root);
        StringBuilder result = new StringBuilder();
        for (TreeNode node : leaves) {
            StringBuilder current = new StringBuilder();
            TreeNode curr = node;
            Character c = curr.getVal();
            while (!curr.equals(root)) {
                TreeNode parent = curr.getParent();
                if (curr.equals(parent.getLeft())) {
                    current.append("0");
                } else if (curr.equals(parent.getRight())) {
                    current.append("1");
                }
                curr = curr.getParent();
            }
            node.setNumBits(current.length());
            StringBuilder temp = new StringBuilder(current);
            node.setBitRep(temp.reverse().toString());
            if (c == '\n'){
                current.append(" n\\");
            }
            else if (c == ' '){
                current.append(" \u2423");
            }
            else{
                current.append(" " + c);
            }
            current.reverse();
            result.append(current + "\n");
        }
        return result.toString();
    }
    public TreeNode getRoot() {
        return root;
    }
    public void setRoot(TreeNode root) {
        this.root = root;
    }
```

```java
    public int getSize() {
        return size;
    }
    public void setSize(int size){
        this.size = size;
    }
    public ArrayList<TreeNode> getLeaves() {
        return leaves;
    }
    /*      FOR DEBUGGING     */
    public String toString(){
        String result = "";
        Queue<TreeNode> queue = new ArrayBlockingQueue<>(getSize());
        queue.add(root);
        while (queue.size() > 0){
            TreeNode curr = queue.remove();
            if (curr.getVal() != null){
                if (curr.getVal().equals('\n')){
                    result = result + "<" + curr.getKey() + "," + "\\n" + ">" +
curr.isLeaf() + " ";
                }
                else if (curr.getVal().equals(' ')){
                    result = result + "<" + curr.getKey() + "," + "\u2423" + ">" +
curr.isLeaf() + " ";
                }
                else{
                    result = result + "<" + curr.getKey() + "," + curr.getVal() + ">" +
curr.isLeaf() + " ";
                }
            }
            else{
                result = result + "<" + curr.getKey() + "," + curr.getVal() + ">" +
curr.isLeaf() + " ";
            }
            if (curr.getLeft() != null){
                queue.add(curr.getLeft());
            }
            if (curr.getRight() != null){
                queue.add(curr.getRight());
            }
        }
        return result;
    }
}

/*  Java Class: TreeNode.java
    Author: Jin Choi
    Class: CSCI 230
    Date: May 9th, 2018
    Description: The Node within the Huffman Tree. Used a separate class as opposed to
nested classes like those used in the book for,in my opinion, cleaner code management.
    I certify that the code below is my own work.
   Exception(s): N/A
*/
import java.util.Queue;
public class TreeNode {
    private Integer key;
    private Character val;
    private TreeNode parent, left, right;
    private String bitRep;
    private int numBits;
    public TreeNode(Integer key){
        this.key = key;
        this.val = null;
        parent = null;
        left = null;
        right = null;
    }
    public TreeNode(Integer key, Character val){
        this.key = key;
        this.val = val;
        parent = null;
        left = null;
        right = null;
    }
    public boolean isLeaf() { return val != null; }
    public TreeNode getParent() { return parent; }
    public void setParent(TreeNode parent) { this.parent = parent; }
    public TreeNode getLeft() { return left; }
    public void setLeft(TreeNode left) {
        this.left = left;
    }
    public TreeNode getRight() { return right; }
    public void setRight(TreeNode right) {
        this.right = right;
    }
    public Integer getKey() { return key; }
    public void setKey(Integer key) { this.key = key; }
    public Character getVal() { return val; }
    public void setVal(Character val) { this.val = val; }
    public String toString(TreeNode t){
        StringBuilder result = new StringBuilder();
        result.append("<" + t.getKey() + "," + t.getVal()+ ">");
        return result.toString();
    }
    public String getBitRep() {
        return bitRep;
    }
    public void setBitRep(String bitRep) {
        this.bitRep = bitRep;
    }
    public int getNumBits() {
        return numBits;
    }
    public void setNumBits(int numBits) {
        this.numBits = numBits;
    }
}
```

Run Time Output:

```
Pattern: america
Algorithm: Boyer-Moore
Run Time: 2.989654ms
Total Comparisons: 37
Average comparisons: 0.21893491124260356
Match found at index: 169

Pattern: aeiou
Algorithm: Boyer-Moore
Run Time: 1.889762ms
Total Comparisons: 2101
Average comparisons: 0.2419113413932067
Match not found

Pattern: waging
Algorithm: Boyer-Moore
Run Time: 1.551754ms
Total Comparisons: 1177
Average comparisons: 0.20373896486065432
Match found at index: 5777

Pattern: dissolutions
Algorithm: Boyer-Moore
Run Time: 1.31435ms
Total Comparisons: 376
Average comparisons: 0.11547911547911548
Match found at index: 3256

Pattern: america
Algorithm: Knuth-Morris-Pratt
Run Time: 0.026167ms
Total Comparisons: 182
Average comparisons: 1.0769230769230769
Match found at index: 169

Pattern: aeiou
Algorithm: Knuth-Morris-Pratt
Run Time: 0.576252ms
Total Comparisons: 9169
Average comparisons: 1.0557282671272308
Match not found

Pattern: waging
Algorithm: Knuth-Morris-Pratt
Run Time: 0.356442ms
Total Comparisons: 5841
Average comparisons: 1.0110784144019387
Match found at index: 5777

Pattern: dissolutions
Algorithm: Knuth-Morris-Pratt
Run Time: 0.204061ms
Total Comparisons: 3363
Average comparisons: 1.0328624078624078
Match found at index: 3256
```

```
Pattern: agagagtaaaaaa
Algorithm: Boyer-Moore
Run Time: 1.815224ms
Total Comparisons: 295
Average comparisons: 0.1881377551020408
Match found at index: 1568

Pattern: aactattctctg
Algorithm: Boyer-Moore
Run Time: 9.190227ms
Total Comparisons: 5493
Average comparisons: 0.35578729192305203
Match found at index: 15439

Pattern: tagtac
Algorithm: Boyer-Moore
Run Time: 2.740088ms
Total Comparisons: 1105
Average comparisons: 0.9177740863787376
Match found at index: 1204

Pattern: agtcgtacxagtcg
Algorithm: Boyer-Moore
Run Time: 2.398625ms
Total Comparisons: 8203
Average comparisons: 0.5308697903184054
Match not found

Pattern: agagagtaaaaaa
Algorithm: Knuth-Morris-Pratt
Run Time: 0.138624ms
Total Comparisons: 2119
Average comparisons: 1.3514030612244898
Match found at index: 1568

Pattern: aactattctctg
Algorithm: Knuth-Morris-Pratt
Run Time: 1.725943ms
Total Comparisons: 19761
Average comparisons: 1.279940410648358
Match found at index: 15439

Pattern: tagtac
Algorithm: Knuth-Morris-Pratt
Run Time: 0.102058ms
Total Comparisons: 1469
Average comparisons: 1.2200996677740863
Match found at index: 1204

Pattern: agtcgtacxagtcg
Algorithm: Knuth-Morris-Pratt
Run Time: 1.540907ms
Total Comparisons: 20232
Average comparisons: 1.309345068599534
Match not found
```

```
The result of compression:
n 000
m 001
o 010
\n 0110
r 0111
d 100
y 1010
␣ 1011
e 11
*****
Number of characters: 18
Number of bits: 54

The result of decompression:

more money needed
```