**ANALYSIS OF ALGORITHMS : Assignment 2**

**HUFFMAN CODING:**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.\*;

abstract class HuffmanTree implements Comparable<HuffmanTree> {

public final int frequency; // the frequency of this tree

public HuffmanTree(int freq) { frequency = freq; }

// compares on the frequency

public int compareTo(HuffmanTree tree) {

return frequency - tree.frequency;

}

}

class HuffmanLeaf extends HuffmanTree {

public final char value; // the character this leaf represents

public HuffmanLeaf(int freq, char val) {

super(freq);

value = val;

}

}

class HuffmanNode extends HuffmanTree {

public final HuffmanTree left, right; // subtrees

public HuffmanNode(HuffmanTree l, HuffmanTree r) {

super(l.frequency + r.frequency);

left = l;

right = r;

}

}

public class Huffman {

public static char[] characters= new char[1000];

public static String[] codes=new String[1000];

static int i=0;

// input is an array of frequencies, indexed by character code

public static HuffmanTree buildTree(int[] charFreqs) {

PriorityQueue<HuffmanTree> trees = new PriorityQueue<HuffmanTree>();

// initially, we have a forest of leaves

// one for each non-empty character

for (int i = 0; i < charFreqs.length; i++)

if (charFreqs[i] > 0)

trees.offer(new HuffmanLeaf(charFreqs[i], (char)i));

assert trees.size() > 0;

// loop until there is only one tree left

while (trees.size() > 1) {

// two trees with least frequency

HuffmanTree a = trees.poll();

HuffmanTree b = trees.poll();

// put into new node and re-insert into queue

trees.offer(new HuffmanNode(a, b));

}

return trees.poll();

}

public static void printCodes(HuffmanTree tree, StringBuffer prefix) {

assert tree != null;

if (tree instanceof HuffmanLeaf) {

HuffmanLeaf leaf = (HuffmanLeaf)tree;

// print out character, frequency, and code for this leaf (which is just the prefix)

System.out.println(leaf.value + "\t" + leaf.frequency + "\t" + prefix);

characters[i]=leaf.value;

codes[i]=prefix.toString();

i++;

} else if (tree instanceof HuffmanNode) {

HuffmanNode node = (HuffmanNode)tree;

// traverse left

prefix.append('1');

printCodes(node.left, prefix);

prefix.deleteCharAt(prefix.length()-1);

// traverse right

prefix.append('0');

printCodes(node.right, prefix);

prefix.deleteCharAt(prefix.length()-1);

}

}

public static void main(String[] args) throws Exception {

int j=0;

String result;

int[] charFreqs = new int[10000];

char c[] = new char[100000];

FileReader fr = new FileReader("kala.txt");

BufferedReader br = new BufferedReader(fr);

br.read(c);

FileWriter fw1=new FileWriter("result.txt");

fw1.write("");

fw1.close();

for(int i = 0; i < c.length; i++)

{

if(c[i]!='\0')

charFreqs[c[i]]++;

//Do stuff here

}

// we will assume that all our characters will have

// code less than 256, for simplicity

// read each character and record the frequencies

// build tree

HuffmanTree tree = buildTree(charFreqs);

System.out.println("CHAR\tFREQ\tHUFFMAN CODE");

printCodes(tree, new StringBuffer());

FileReader fr1 = new FileReader("kala.txt");

BufferedReader br1 = new BufferedReader(fr1);

br1.read(c);

for(int i = 0; i < c.length; i++)

{

if(c[i]!='\0')

{

for(int k=0;k<characters.length;k++)

{

if(c[i]==characters[k])

{

j=k;

}

}

result=codes[j];

try

{

String filename= "result.txt";

FileWriter fw = new FileWriter(filename,true);

BufferedWriter bw= new BufferedWriter(fw);//the true will append the new data

bw.write(result.toString());

bw.close();//appends the string to the file

fw.close();

}

catch(IOException ioe)

{

System.err.println("IOException: " + ioe.getMessage());

}

}

}

}

}

**INPUT AND OUTPUT:**

Input:

kala.txt:

abccdddeeeeeffffffff

Output:

result.txt:

000010000000010001001001001010101010111111111

**ROD CUTTING :**

import java.util.Arrays;

import java.util.Scanner;

public class RodCutting {

public static int n;

public static int pricelength,i,j;

public static int cost[]= new int[100];

private static int [] max\_cost = new int[100];

private static void inputmethod()

{

Scanner in = new Scanner(System.in);

System.out.println("Prices for how many rods had to be entered ?");

pricelength = in.nextInt();

for(i=0;i<pricelength;i++)

{

System.out.println("Enter the price of rod length " + (i+1)+":");

cost[i]= in.nextInt();

}

System.out.println("The given prices are: ");

for(j=0;j<pricelength;j++)

{

System.out.print(cost[j]);

System.out.print("\t");

}

System.out.println("\n");

System.out.println("Enter the length of the given rod : ");

n=in.nextInt();

}

private static int maxPrice(int n1){

if(n1 == 1 ){

return cost[0];

}

if(n1 == 0){

return 0;

}

int currentVal = 0;

int maxVal = 0;

for(int i = 0; i <= n1 - 1 ; i++ ){

if(max\_cost[n1 - 1 - i] == -1){

max\_cost[n1 - 1 - i] = maxPrice(n1-i-1);

}

currentVal = cost[i] + max\_cost[n1 - i -1];

if(currentVal > maxVal){

maxVal = currentVal;

}

}

return maxVal;

}

public static void main(String[] args) {

inputmethod();

Arrays.fill(max\_cost, -1);

System.out.print("The maximum amount that can be got with this rod will be :");

System.out.println(maxPrice(n));

}

}

**INPUT AND OUTPUT :**

Prices for how many rods had to be entered ?

10

Enter the price of rod length 1:

2

Enter the price of rod length 2:

6

Enter the price of rod length 3:

8

Enter the price of rod length 4:

10

Enter the price of rod length 5:

20

Enter the price of rod length 6:

9

Enter the price of rod length 7:

10

Enter the price of rod length 8:

2

Enter the price of rod length 9:

12

Enter the price of rod length 10:

10

The given prices are:

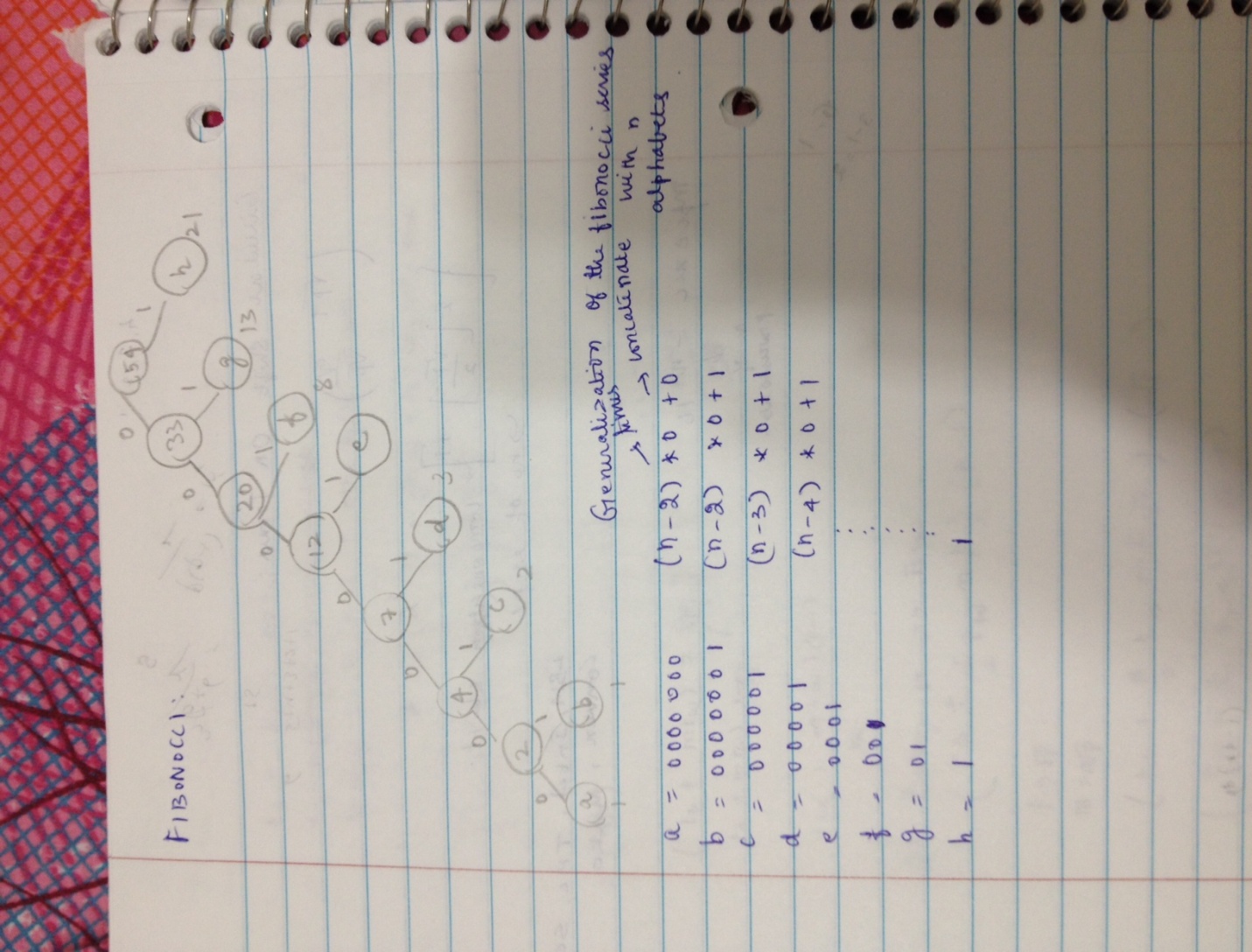
2 6 8 10 20 9 10 2 12 10

Enter the length of the given rod :

10

The maximum amount that can be got with this rod will be :40

**FIBONACCI SERIES OF INPUT IN HUFFMAN CODING :**

****