# Data Structures and Algorithms

# INFO 6205

# Homework 4

# Due: October 9, 2020

Put all your java, compiled class files and documentation files into a zip file named Homework4.zip and submit it via dropbox on Canvas before the END of due date. Put your name on all .java files. There will be a short quiz on this homework.

1. Write a recursive method *countBinary* that has one integer parameter *n* and returns the number of binary strings of length n that do not have two consecutive 1's. For example, for n = 4, the number of binary strings of length 4 that do not contain two consecutive 0's is 7: 1111, 1110, 1101, 1011, 1010, 0111, 0110, 0100

2. The Recursive operations for Factorial and Fibonacci sequence was discussed in class.

A) For factorial 6!a) Show recursive stack operations, provide details step-by-step, b**)** Walk through your stack operations and provide the result. c**)** Write Java code with input factorial 6! d**)** Compile and run your program, what is the running time of your algorithm?

B) For Towers of Hanoi problem with n=6 discs, how does the algorithm work? What data structures would you use? provide step by step operations. Write Java code, compile and run program.

C) For Power (x, n) with n=6, a) write Iterative algorithm step-by-step, and b) write Recursive algorithm, step-by-step, c) Write Java code for a & b, compile and run

D) For Fibonacci sequence with n=6, the following fibonacci diagram shows its Tree Structure a) Is this diagram iterative or recursive?

b) What data structure is used to implement recursion?

c) Provide Tree Structure for n=7 step-by-step. What differences do you see in

diagrams between n=6 and n=7?

d) What are Pros and Cons between iterative and recursive Algorithms?

e) Write Java code for both n=6 and n=7.

f) Compile and Run your programs, report results

3. An *n*-bit Gray code is a list of the 2*n* different *n*-bit binary numbers such that each entry in the list differs in precisely one bit from its predecessor. The *n* bit binary reflected Gray code is defined recursively. How does algorithm works for n=6, describe step-by-step. Write Java code, compile and run program.

4. Consider the following, Input Data: {12, 22, 38, 3, 9, 82, 10, 31, 24, 33 }

a) Graphically build a Circular queue for input data. Discuss and show Head

and Tail pointers at each step:

i) enqueue all input data

ii) dequeue three elements

iii) enqueue two elements

iv) dequeue all elements

b) Write Java code for the Circular queue, provide enqueue, dequeue, isEmpty, isFull,

and displayQueue methods, to show the status of the queue with steps described in (a).

Compile code and Run with input data.

5. Consider String "Huffman coding is a data compression algorithm.”

Note: It uses frequency of text data elements to generate data compression

a) Generate a binary Huffman Tree step-by-step

b) Show binary data both before and after compression. Analyze difference.

c) Consider attached Java code. That code is not currently working.

Compile the code and see the errors and fix the problems.

d) Show as to why/how the algorithm uses PriorityQueue

e) Compile Java code and run it with the input string provided above.

f) Compare the result that you did step-by-step with output of running program.

import java.util.Comparator;

import java.util.HashMap;

import java.util.Map;

import java.util.PriorityQueue;

// A Tree node

class Node

{

char ch;

int freq;

Node left = null, right = null;

Node(char ch, int freq)

{

this.ch = ch;

this.freq = freq;

}

public Node(char ch, int freq, Node left, Node right) {

this.ch = ch;

this.freq = freq;

this.left = left;

this.right = right;

}

}

class HuffmanEncoding

{

// traverse the Huffman Tree and store Huffman Codes in a map.

public void encode(Node root, String str, Map<Character,String> huffmanCode)

{

if (root == null)

return;

// found a leaf node

if (root.left == null && root.right == null) {

huffmanCode.put(root.ch, str);

}

encode(root.left, str + '0', huffmanCode);

encode(root.right, str + '1', huffmanCode);

}

// traverse the Huffman Tree and decode the encoded string

public int decode(Node root, int index, StringBuilder sb)

{

if (root == null)

return index;

// found a leaf node

if (root.left == null && root.right == null)

{

System.out.print(root.ch);

return index;

}

index++;

if (sb.charAt(index) == '0')

index = decode(root.left, index, sb);

else

index = decode(root.right, index, sb);

return index;

}

// Builds Huffman Tree and huffmanCode and decode given input text

public static void buildHuffmanTree(String text)

{

// count frequency of appearance of each character and store it in a map

for (char c: text.toCharArray()) {

freq.put(c, freq.getOrDefault(c, 0) + 1);

}

// Create a priority queue to store live nodes of Huffman tree

// Notice that highest priority item has lowest frequency

pq = new PriorityQueue<>(Comparator.comparingInt(l -> l.freq));

// Create a leaf node for each character and add it

// to the priority queue.

for (var entry : freq.entrySet()) {

pq.add(new Node(entry.getKey(), entry.getValue()));

}

// do till there is more than one node in the queue

while (pq.size() != 1)

{

// Remove the two nodes of highest priority

// (lowest frequency) from the queue

Node left = pq.poll();

Node right = pq.poll();

// Create a new internal node with these two nodes as children

// and with frequency equal to the sum of the two nodes

// frequencies. Add the new node to the priority queue.

int sum = left.freq + right.freq;

pq.add(new Node('\0', sum, left, right));

}

// root stores pointer to root of Huffman Tree

Node root = pq.peek();

// traverse the Huffman tree and store the Huffman codes in a map

Map<Character, String> huffmanCode = new HashMap<>();

encode(root, "", huffmanCode);

// print the Huffman codes

System.out.println("Huffman Codes are : " + huffmanCode);

System.out.println("Original string was : " + text);

// print encoded string

StringBuilder sb = new StringBuilder();

for (char c: text.toCharArray()) {

sb.append(huffmanCode.get(c));

}

System.out.println("Encoded string is : " + sb);

// traverse the Huffman Tree again and this time

// decode the encoded string

int index = -1;

System.out.print("Decoded string is: ");

while (index < sb.length() - 2) {

index = decode(root, index, sb);

}

}

public static void main(String[] args)

{

String text = "Huffman coding is a data compression algorithm.";

buildHuffmanTree(text);

}

}