Data Structure II

REPORT TITLE

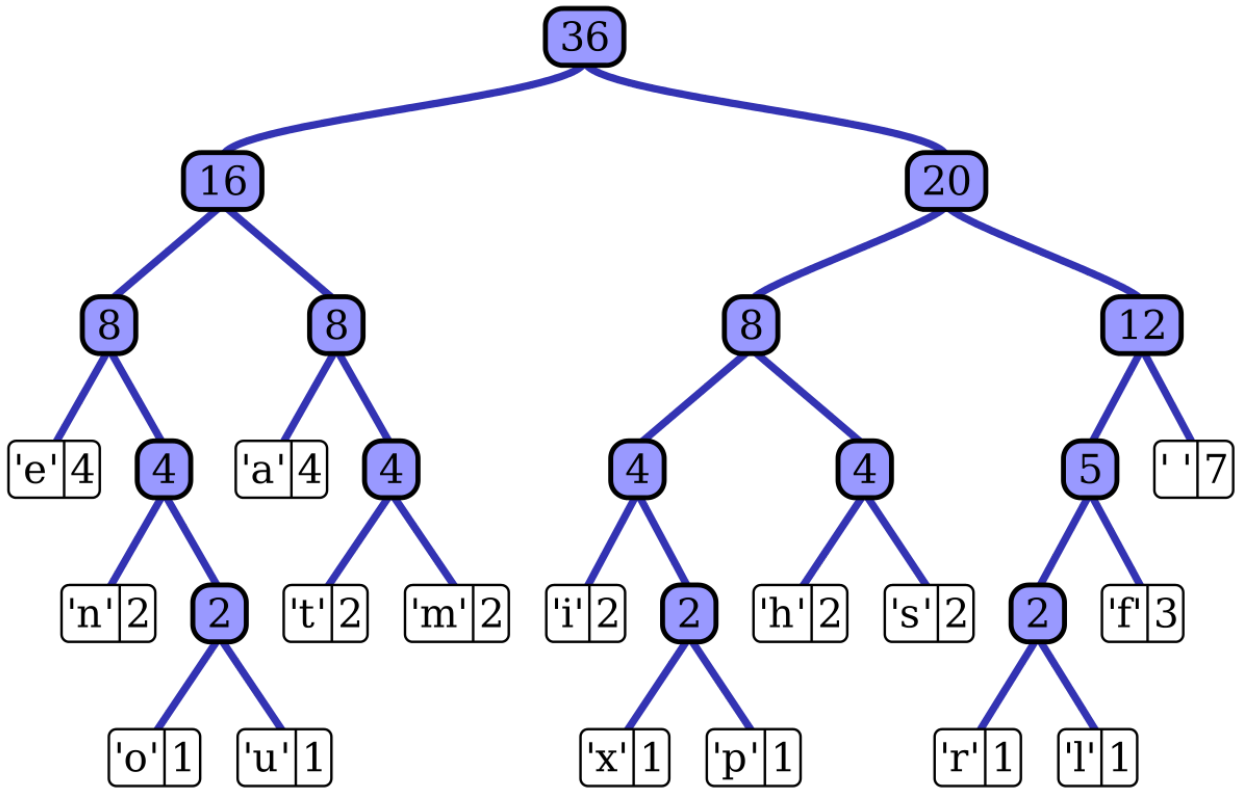
Huffman Compression & Decompression

Name:	**ID:**
Ahmed Mos`ad Mahmoud	4604
Ahmed Medhat	4612
Mahmoud Fouad	4656

Introduction

a **Huffman code** is a particular type of optimal prefix code that is commonly used for lossless data compression.

Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.



Huffman Coding Major parts:

1) Build a Huffman Tree from input characters.

2) Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree:

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Data Structures used:

- Hashmaps: Used to Map Characters and their frequencies and store the Characters and their equivalent key.
- ArrayList: Used to Store a Stream of 0 and 1 after switching the string of characters to a stream of 0s and 1s code
- Priority queues: Used to store the nodes of the tree in an ordered queue to build the minimum heap.

Algorithms used:

1- getFreqencies(String filepath): Maps each Character with its Frequency after reading the file(Calls Read(String filepath)) Complexity is O(n) as it loops on n chars.

2- Read(filepath): Reads the file and stores it in a StringBuilder.

3- HuffmanTree() Constructor: Creates Tree nodes from hashmap and assigns value and character to each node and puts it in the priority queue with the help of node comparator class.

4- generateTree(Node root, String s):Builds a Min heap using the priority queue using recursion and assigns binary values while traversing through the tree (0 for left, 1 for right). Complexity is o(logn).

5- Compress(String filepath): Compresses the given file by converting each char in the Original StringBuilder read from the file into its code generated from the Huffman Tree and then takes every 8 binary numbers converts them into integer and use the integer to get its

corresponding unicode character and stores it in a StringBuilder then prints it to the file to compress the file size.

Complexity is O(nlogn).

Header Format of file

X=Y

Where X is the character(key in Hashmap) and Y is the corresponding code to the char according to the Huffman tree(Hashmap value).

Then it is followed by the Encoded text.

Time Complexity

Huffman is a greedy algorithm with time complexity of the **O(nlogn)**. Using a heap to store the weight of each tree, each iteration requires **O(logn)** time to determine the cheapest weight and insert the new weight. There are **O(n)** iterations, one for each item.

User manual:

1-Choose whether to Compress or Decompress.

2-Enter file name.