Huffman Coding and Decoding

AL-Nasser ID:437101744
Computer Science
Riyadh, Saudi Arabia
Sulaiman.n.n7@gmail.com

Allam ID:437104519
Computer Science
Riyadh,Saudi Arabia
Ferasalallam@gmail.com

I. Introduction

Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding.

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol.

More common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted.

II. Huffman Coding And Decoding

A. *Coding and Encoding*

The technique works by creating a binary tree of nodes. These are stored in a Priority Queue, the size of which depends on the number of symbols, . A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the **symbol** itself, the **weight** (frequency of appearance) of the symbol and optionally, a link to a **parent** node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain a **weight**, links to **two child nodes** and an optional link to a **parent** node. bit '1' represents following the left child and bit '0' represents following the right child. Generally speaking, the process of decompression is simply a matter of translating the stream of prefix codes to individual byte values, usually by traversing the Huffman tree node by node as each bit is read from the input stream (reaching a leaf node necessarily terminates the search for that particular byte value).

B. *Decoding*

The Decoding technique is that we take the root node and the Encoded text and we take every codeword for the length of the encoded text and we look for the leaf node that represents this certain codeword we traverse the tree from top to bottom and we look for the leaf by comparing between '1' and '0' until we reach the character then we add it to the string which will eventually Decode the text to its original form.

III. IMPLEMENTED APPROACH

Our Implemented approach was that we used a priority queue of nodes, the node contains the character and the frequency and the other left child and the right child. We also used a TreeMap which contains all the characters and their code word.

For the GUI we used the javax swing library to implement our GUI we add three buttons one to browse the text file the other is to encode the text file and the third one will build our tree from bottom to up.

A. *Units*

- The text file will be measured in “Bytes” and the output file as well
- The execution speed will be measured in “Milliseconds”

B. *Algorithm*

1)Building the tree

```
1: n := |C|;
2: Q := C;
3: for i := 1 to n − 1 do
4: allocate a new node z
5: z.left := x := Extract-Min(Q);
6: z.right := y := Extract-Min(Q);
7: z.freq := x.freq + y.freq;
8: Insert(Q, z);
9: end for
10: return ROOT {return the root of the tree}
```

O(nlogn)

2)Assigning the codeword

We start for the root node

```
1:if the root is not Null do
2:if the right child is not null do
3:we call the same method with the right child and '0'
4:if the right child is not null do
5:we call the same method with the right child and '1'
6: if the node is leaf node do
7: assign the character with the called '1' or '0'
```

$T(n)=2T(n/2)+O(1)$

Using the master theorem, we know that the algorithm Is O(n)

3)Encoding the text

```
1:for i=0 to input length do
2:save to file ← the code word of each character of the text
```

```
3: show output
O(n) where n is the text length
```

4)Decoding

```
1:String decoded
2: Node←root Node
3:for i=0 to the length of the encoded text do{
4: tempnode← Node
5:while tempnode isn’t leaf and i<encoded length{
6: if tempnode charat(i)==0 do tempnode go right
7: else do tempnode go left
8: i++;
9:}
10: if tempnode != null do add the character to decoded
O(nlogn) Where n is the encoded text length
```

IV. Example Of Execution

In our example of the execution we used many text files to measure our time complexity and the execution time.

We will browse the text file from the GUI and then encode it the encoding will write the encoded text into a file and measure its size.

a) In the follwing table we used the example “sulaiman nasser alnasser” and we took each character alone and filled it’s codeword and the codeword bits, the character frequency.

We take the codeword bits and we multiply it with the freqeuncy so we can get the number of bits that will be stored in the encoded text file.

Example	**sulaiman nasser alnasser**										
charecter	s	a	n	e	l	r	M	i	u	Space	sum
Codeword	11	10	0000	0010	0100	0001	001110	010111	0101	011	
Codeword bits	2	2	4	4	4	4	6	6	4	3	
character frequency	5	5	3	2	2	2	1	1	1	3	
Codeword bits* frequency	10	10	12	8	8	8	6	6	4	9	81

b)The Measuring of performance and execution time

Of the algorithm

We used the System.currentMillies() to get the execution time of the algorithm
And we came up with these results –

The Execution time with a small file (around 107 byte text) is approximately
(48 + 60 + 47 + 45 + 48 + 51 + 58 + 44 + 44 + 40) / 10 = 84.5 ms

The Execution time with a more bigger file (around 1570 byte text) is approximately

(121 + 127 + 225 + 298 + 227 + 440 + 390 + 242 + 321 + 490 /) 10 = 2440 ms

The Execution time with even more bigger file (around 6,288 byte text) is approximately

(920 + 2239 + 3409 + 4180 + 3210 + 4090 + 4309 + 4230 + 5312 + 4324)/10 = 3620 ms

References

[1] Huffman coding Wiki https://en.wikipedia.org/wiki/Huffman_coding#Problem_definitiont

[2] Some of the code prior to change https://gist.github.com/ahmedengu/aa8d85b12fccf0d08e895807edee7603

[3] Some of the GUI code prior to change https://github.com/sukalyansen123/Huffman-Tree-Animation-GUI

[4] Algorithm: http://math.ubbcluj.ro/~tradu/TI/huffmancode.pdf

[5]how to use java swing to implement buttons and draw lines and squares

https://www.youtube.com/watch?v=zG8CrISqPpU