USING HUFFMAN ENCODING TO REDUCE TIME COMPLEXITY OF SORTING AND SEARCHING SYSTEMS IN LARGE SCALE APPLICATIONS

Mentor

Prof. Kaviya Elakkiya

AP/SCOPE

Presented by

22BCE5252-ANIRUDH SRIDHAR

22BRS1373-LOGAMITHRAN M.R.

INTRODUCTION

- Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.
- The variable-length codes assigned to input characters are "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.
- In this project, we aim to apply the data compression principles of the Huffman coding algorithm to encode data from frequency tables to improve the efficiency of storage and retrieval of the frequency tables.

- The frequency tables are represented in the form of binary trees. Each element of the frequency table is coded from the binary tree in a way such that the elements with the highest ffrequceny are coded t oahve the least number of bits.
- Subsequently, the element with the lowest frequency will have the highest number of bits. This ensures a faster and more efficient sorting system because binary numbers take lower storage space and hence are faster to iterate through than integers and floating point numbers.

RESEARCH SURVEY

- A survey of similar research topics was conducted prior to idea selection, finding results related to the Huffman Encoding Algorithm.
- The survey found that most applications were related to data compression and the use of Huffman Encoding in the use of compression.
- Papers were found related to the Huffman Binary Tree, but nothing related to the Huffman Encoding Algorithm in the use of frequency analysis and optimization.

PROBLEM DEFINITION

- In this particular project, we are using Huffman Encoding to reduce the time complexity of a storage system.
- We are considering a shop situation where there is an extensive searching system. If we reduce each frequency table into binary numbers, this system will take up lesser overall space, and this helps us sort and search through it faster.
- We can also restock certain items quicker than others because we can surf through the binary tables and restock on items whose frequencies have lower number of bits.
- This helps us keep rotating the stocks of the items in the shop, so that itemflow can be maintained in an equilibrium state.

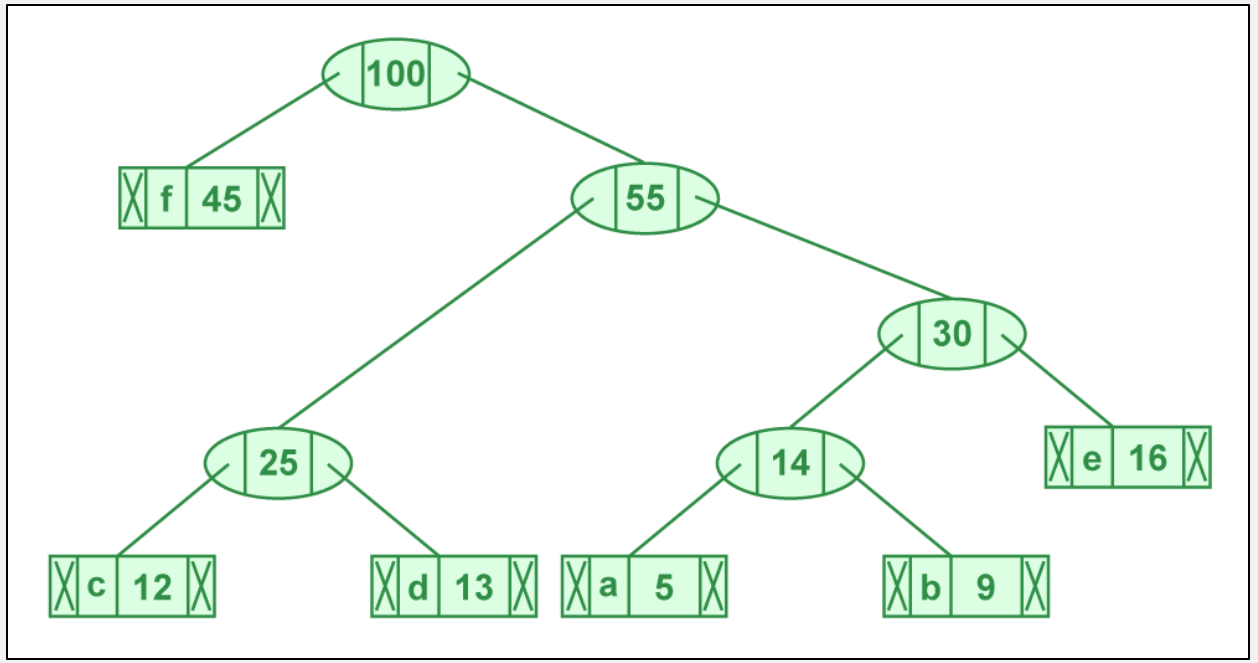
OBJECTIVE

- To analyse the data of frequency tables and compress them into binary trees.
- To analyse the binary tree formation to optimize the data in the frequency tables.
- To implement the Huffman Encoding algorithm to compress the binary trees and create a method to obtain the frequency table data in binary form, increasing data retrieval speed for larger data sets

METHODOLOGY

- A frequency table is obtained and a node with its label being the sum of the frequencies of the two least elements is made.
- The process is repeated until the frequency table only has elements whose frequencies are higher than that of the nodes left.
- The frequencies of the node and element which are the lowest two frequencies are combined and this process is continued till the table is reduced to one single node.
- A code is then assigned to the tree to print it. The tree is then traversed from left to right, and the left child of a node is assigned a “0” and the right child of a node a “1”.
- Whenever we encounter a node that doesn’t have any child nodes, aka a leaf node, the binary code till that point for each element of the frequency table is printed.
- This way, the larger the dataset, the easier it is to store the elements in memory because they can all be encoded as binary numbers, hence taking the least amount of space needed.

DESIGN DIAGRAM



IMPLEMENTATION RESULTS

```
#include <cstdlib>
#include <iostream>
#define max 100
struct minhnode {
    char data;
    unsigned freq;
    struct minhnode *left, *right;
};
struct minheap {
    unsigned size;
    unsigned capacity;
    struct minhnode** array;
};
```

```
struct minhnode* nnode(char data, unsigned freq) {
    struct minhnode* temp= (struct minhnode*)malloc(sizeof(struct minhnode));

    temp -> left = temp -> right = NULL;
    temp -> data = data;
    temp -> freq = freq;

    return temp;
}

struct minheap* createminheap(unsigned capacity) {
    struct minheap* minheap1= (struct minheap*)malloc(sizeof(struct minheap));

    minheap1 -> size = 0;
    minheap1 -> capacity = capacity;
    minheap1 -> array=(struct minhnode**)malloc((minheap1-> capacity)* sizeof(struct(minhnode*)));

    return minheap1;
}
```

```
void swapminheapnodes(struct minhnode** a, struct minhnode** b) {


    struct minhnode* t= *a;
    *a = *b;
    *b = t;
}
```

//This is the piece of code to create a new node, using pointers and structs and swapping those two nodes using a temp pointer.

SUMMARY

- In conclusion, the Encoding Algorithm is able to be effectively used to optimize frequency tables for faster access and summarization.

REFERENCES

- https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/
- https://www.geeksforgeeks.org/counting-frequencies-of-array-elements/
- https://iq.opengenus.org/huffman-encoding/