VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

PROJECT REPORT

Project Title:

Data Compression Using Huffman Coding

CSE 2003-DSA

Submitted By: -

Swayam Prakash Pati (19BCT0161)

Disha Jain (19BCT0219)

Rachit Elhance (19BCT0187)

Rahul Thakar (19BCT0192)

Under the Guidance of: -

Prof. PARVEEN SULTANA H

In partial fulfilment for the award of the degree of

B.Tech

In

Computer Science and Engineering Spc. In IOT

Contents

Aim and Objective

Implementing lossless data compression technique. Huffman coding is based on the frequency of occurrence of a data item.

Abstract

Data compression is also called as source coding. It is the process of encoding information using fewer bits than an un- coded representation is also making a use of specific encoding schemes. Compression is a technology for reducing the quantity of data used to represent any content without excessively reducing the quality of the picture. It also reduces the number of bits required to store and/or transmit digital media. Compression is a technique that makes storing easier for large amount of data. There are various techniques available for compression in my paper work, I have analyzed Huffman algorithm and compare it with other common compression techniques like Arithmetic, LZW and Run Length Encoding.

Basic Principle and Methodology

Huffman coding is an encoding algorithm used for lossless data compression in computer science and information theory. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix-free code (that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol) that expresses the most common characters using shorter strings of bits than are used for less common source symbols.

Creating the tree:

1. Start with as many leaves as there are symbols.

2. Enqueue all leaf nodes into the first queue (by probability in increasing order so that the least likely item is in the head of the queue).

3. While there is more than one node in the queues:

    1. Dequeue the two nodes with the lowest weight.

    2. Create a new internal node, with the two just-removed nodes as children (either node can be either child) and the sum of their weights as the new weight.

    3. Enqueue the new node into the rear of the second queue.

4. The remaining node is the root node; the tree has now been generated.

Process Module

- **Main Module: -** This will accept the string to be encoded and call the tree building module.
- **Build Huffman Tree Module: -** Accept the string to be encoded which then uses further modules together to build the tree from frequency of occurrence of symbols.
- **Node Module: -** To create nodes for each leaf to be created for each symbol in the string.
- **Encode Module: -** Encode the string according to Huffman Algorithm and return in coded format.
- **Decode Module: -** To decode the encoded string just to demonstrate that the original string can be retrieved.

Code Implementation

```
#include <iostream>
#include <string>
#include <queue>
#include <unordered_map>
using namespace std;

// A Tree node
struct Node
{
    char ch;
    int freq;
```

```
    Node *left, *right;
};



// Function to allocate a new tree node
Node* getNode(char ch, int freq, Node* left, Node* right)
{
    Node* node = new Node();

    node->ch = ch;
    node->freq = freq;
    node->left = left;
    node->right = right;

    return node;
}

// Comparison object to be used to order the heap
struct comp
{
    bool operator()(Node* l, Node* r)
    {
        // highest priority item has lowest frequency
        return l->freq > r->freq;
    }
};

// traverse the Huffman Tree and store Huffman Codes
// in a map.
void encode(Node* root, string str,
            unordered_map<char, string> &huffmanCode)
```

```
{
    if (root == nullptr)
        return;


    // found a leaf node
    if (!root->left && !root->right) {
        huffmanCode[root->ch] = str;
    }

    encode(root->left, str + "0", huffmanCode);
    encode(root->right, str + "1", huffmanCode);
}

// traverse the Huffman Tree and decode the encoded string
void decode(Node* root, int &index, string str)
{
    if (root == nullptr) {
        return;
    }

    // found a leaf node
    if (!root->left && !root->right)
    {
        cout << root->ch;
        return;
    }

    index++;

    if (str[index] =='0')
```

```
        decode(root->left, index, str);
    else
        decode(root->right, index, str);
}


// Builds Huffman Tree and decode given input text
void buildHuffmanTree(string text)
{
    // count frequency of appearance of each character
    // and store it in a map
    unordered_map<char, int> freq;
    for (char ch: text) {
        freq[ch]++;
    }

    // Create a priority queue to store live nodes of
    // Huffman tree;
    priority_queue<Node*, vector<Node*>, comp> pq;

    // Create a leaf node for each character and add it
    // to the priority queue.
    for (auto pair: freq) {
        pq.push(getNode(pair.first, pair.second, nullptr, nullptr));
    }

    // 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.top(); pq.pop();
    Node *right = pq.top();    pq.pop();

    // 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.push(getNode('\0', sum, left, right));
}

// root stores pointer to root of Huffman Tree
Node* root = pq.top();

// traverse the Huffman Tree and store Huffman Codes
// in a map. Also prints them
unordered_map<char, string> huffmanCode;
encode(root, "", huffmanCode);

cout << "Huffman Codes are :\n" << '\n';
cout<<"\tCharacter\t Code\n\n";
for (auto pair: huffmanCode) {
    cout <<"\t"<<pair.first << "\t\t" << pair.second << '\n';
}

cout << "\nOriginal string was :\n" << text << '\n';

// print encoded string
string str = "";
for (char ch: text) {
    str += huffmanCode[ch];
```

```
    }

    cout << "\nEncoded string is :\n" << str << '\n';

    // traverse the Huffman Tree again and this time
    // decode the encoded string
    int index = -1;
    cout << "\nDecoded string is: \n";
    while (index < (int)str.size() - 2) {
        decode(root, index, str);
    }
}

// Huffman coding algorithm
int main()
{
    string text;
    cout<<"Enter the string: ";
    getline(cin,text);
    buildHuffmanTree(text);
    return 0;
}
```

Output

```
PS C:\Users\Sam\Desktop\Materials> g++ dsapro.cpp
PS C:\Users\Sam\Desktop\Materials> ./a
Enter the string: Huffman coding is a data compression algorithm.
Huffman Codes are :

        Character       Code

        h               111110
        f               11110
        i               1110
        t               11011
        l               110100
        o               1100
        n               1011
        r               10101
        d               0010
        g               0001
        H               00001
        u               00000
        c               0011
        a               010
        e               110101
                        011
        m               1000
        .               111111
        s               1001
        p               10100

Original string was :
Huffman coding is a data compression algorithm.

Encoded string is :
000010000011110111101000010101101100111100001011101011000101111101001011010011001001011011010011001111001000101001010111010110011001111011001011011010110100000111001010111010111101000001
10111101101111111101000111111

Decoded string is:
Huffman coding is a data compression algorithm.
PS C:\Users\Sam\Desktop\Materials>
```

```
PS C:\Users\Sam\Desktop\Materials> ./a
Enter the string: Swayam Disha Rachit and Rahul have made this project.
Huffman Codes are :

        Character       Code

        a               111
        c               11011
        r               010011
        h               001
        n               01000
        y               00000
        t               0110
        i               0101
        R               11001
        S               00001
        j               101000
        w               00010
        .               010010
        D               101001
        s               10101
        p               00011
        e               0111
                        100
        o               101100
        l               101101
        d               10111
        v               110000
        u               110001
        m               11010

Original string was :
Swayam Disha Rachit and Rahul have made this project.

Encoded string is :
000010001011100000111110101001010010101101010011111001100111111011001010101101001110100010111100110011110001101101100001111100000011110011010111101110111000110010101011010110000011010011101100101000011111010110110010010

Decoded string is:
Swayam Disha Rachit and Rahul have made this project.
PS C:\Users\Sam\Desktop\Materials>
```

```
PS C:\Users\Sam\Desktop\Materials> ./a
Enter the string: She sells sea shells on the sea shore.
Huffman Codes are :

        Character       Code

        a               1000
        l               011
        h               010
                        110
        e               00
        s               111
        .               10011
        n               10010
        r               10100
        S               10101
        t               10110
        o               10111

Original string was :
She sells sea shells on the sea shore.

Encoded string is :
10101010001101110001101111111011100100011011101000011011111110101111001011010110010001101110010001101110101011101000010011

Decoded string is:
She sells sea shells on the sea shore.
PS C:\Users\Sam\Desktop\Materials>
```

Conclusion

I have studied the implementation and various applications of compression using Huffman Coding and its advantages and disadvantages. I have concluded that Huffman coding is very efficient for more frequently occurring sequences of content with fewer bits and reduces the file size dramatically. It is easy to implement, fast and lossless algorithm and is also used in JPEG compression. It produces optimal and compact code but relatively slow. Huffman algorithm is based on statistical model which adds to overhead.

References

1. Geeksforgeeks
2. Wikipedia
3. Google
4. Programwiz

Thank You