

**UNITED INTERNATIONAL UNIVERSITY**

**Report - Group – E1**

**Topic - Huffman coding Algorithm**

**Submitted By**

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Sadia Islam** | **011193156** |
| **Md. Ahasan Khan** | **011201122** |
| **MD.Shahidul Islam Sujon** | **011193107** |
| **Ovie Rahaman Sheikh** | **011201306** |

**Course – Data Structure & Algorithms II**

**Course Code – CSE 2217**

**Section – E**

**Department – CSE**

**Table of Contents**

Introductions of Huffman Coding

**Introductions of Huffman coding**

Huffman coding is a type of prefix coded algorithm that is used to encode or decode data. One of the most common applications of this algorithm is lossless data compression.

**Inventor**

The process of finding or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes, which he published in 1952.

**Application**

1. **Cryptography**
2. **Compressing files to PKZIP, MP3, JPEG etc.**

**Principle**

The principle behind Huffman coding is to assign variable-length codes to input characters and the lengths of the codes are dependent on the frequency of the associated characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.

2. Travers 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 filed us 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 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.

Huffman Coding Applications

1. Huffman coding is used in conventional compression formats like GZIP, BZIP2, PKZIP, etc.

2. For text and fax transmissions.

Huffman Coding Complexity

The time complexity for encoding each unique character based on its frequency is O(nlog n).

Extracting minimum frequency from the priority queue takes place 2\*(n-1) times and its complexity is O(log n). Thus the overall complexity is O(nlog n).

**Why Huffman code is better**

**Naive approach**

Suppose our string is: a b a b d c e c a b a c d a d e c a b c

This string contains 20 characters.

We know if we can compress data our total cost will be decrease. So, we need to reduce our text size for transmit it from sender to receiver.

|  |  |
| --- | --- |
| Character | Frequency |
| a | 6 |
| b | 4 |
| c | 5 |
| d | 3 |
| e | 2 |
|  | Total frequency = 20 |

In nave approach we can see our size of Total massage = 20\*8=160 bit

Because of ASCII value is 8 bit per letter so we need to compress the size of our massage we can compress by two method

We can apply these 2 methods:

1. Fixed length code
2. Variable length code

**Fixed length code**

If we use 2 bits binary we can cover =22 = 4 letters

a=00

b=01

c=10

d=11

E=?

Hare is 5 separate character So, we need to use 3-bit code and we can assign 23 = 8 letters

|  |  |  |
| --- | --- | --- |
| Character | Frequency | Code |
| a | 6 | 000 |
| b | 4 | 001 |
| c | 5 | 010 |
| d | 3 | 011 |
| e | 2 | 100 |
| Total letters \* ascii bit = 5\*8=40 |  | Total bit = 5\*3=15 |

**Total massage bit= 20\*3=60**

So total bit is: 40+15=55

So total massage size Is = 60+55=115 bit.

For fixed length code if we take 3-bit binary code we can apply 8 letter but if we have greeter then 4 and less than 8 letter we also use 3 bit so this algorithm is not efficient for compress message so we need a better one

**Huffman coding with simulation**

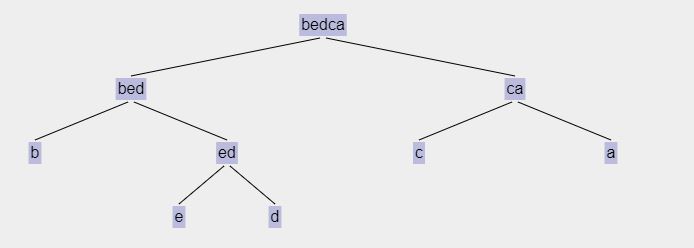
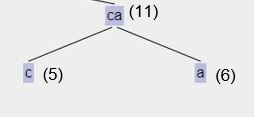
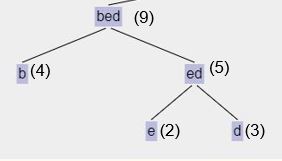
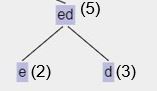
we can reduce more size by using Huffman code algorithm. And it is more efficient than fixed length code.

Huffman coding benefit is which letter occur maximum time those letter codes is lower bit and minimum time occurred letter size is larger bit. So, this algorithm is more efficient then fixed length code.

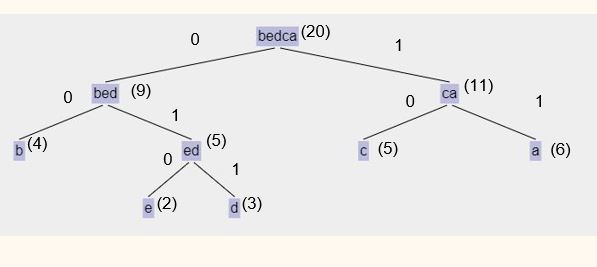
|  |  |
| --- | --- |
| Character | Frequency |
| a | 6 |
| b | 4 |
| c | 5 |
| d | 3 |
| e | 2 |
|  | Total frequency = 20 |

For Huffman coding we need to sort character non-decreasing order based on frequency

|  |  |
| --- | --- |
| Character | Frequency |
| e | 2 |
| d | 3 |
| b | 4 |
| c | 5 |
| a | 6 |



Assign left side edge 0 and right-side edge 1



|  |  |  |  |
| --- | --- | --- | --- |
| Character | Frequency | Code | Frequency \* code |
| a | 6 | 11 | 6\*2=12 |
| b | 4 | 00 | 4\*2=8 |
| c | 5 | 10 | 5\*2=10 |
| d | 3 | 011 | 3\*3=9 |
| e | 2 | 010 | 2\*3=6 |
| Total letters \* ascii bit = 5\*8=40 | + | Total bits = 12 | Sum=45 |

Total bits of size is (40+12+45) =97 bits

So hare we can see a occur 6 time and a code is 2 bit which is 00 and e occur only 2 time and its code is 011 which is 3 bit. And our 3 length code has not contain 2 length code on prefix.