File compression using Huffman Encoding

November 21, 2021

Abhay Shukralia (2020CSB1061) ,
Jatin (2020CSB1090) ,
Tanuj Kumar (2020CSB1134)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Sravanthi Chede

Summary: Huffman coding is an efficient algorithm of compressing data without losing information. The algorithm is developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes" .The idea behind the algorithm is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.The least frequent character gets the longest code and the character which is most frequent gets the smallest.The code is assign to character in such a way 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 bits stream.

---

1. Introduction

What is Huffman encoding ?

Huffman encoding is an algorithm used to compress the size of data without loosing it by assigning different code to each characters using Huffman tree . It is a statistical compression method that converts characters into variable length bit strings and produces a prefix code .Most frequently occurring characters are converted to shortest bit strings and least frequent the longest.

Why is data compression important ?

If we talk about today research says that trillions of data is producing every day. If we have to use this data for future we must store it somewhere that is the reason we have to build some data structures which can compress the size of data without loosing it. Data compression can dramatically decrease the amount of storage a file takes .As a result of compression, administrators spend less money and less time on storage. Compression optimizes backup storage performance and has recently shown up in primary storage data reduction.

Implementation of Huffman Encoding algorithm

1.Firstly we need to take the input from the user that contains the data(text file) which we have to compress using this algorithm

1.a first we have to find the frequency of each character.
1.b creating the Huffman Tree.

2.implementation of Huffman tree

2.a 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.b Extract two nodes with the minimum frequency from the min heap.
2.c 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.
2.d Repeat steps 2.b and 2.c until the heap contains only one node. The remaining node is the root node and the tree is complete.

3..Printing the Codes from Huffman Tree

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered. As we have to print encoded code in output file we maintain an array of pointers pointing to the head of linked list in which we store data of array in linked list

4.Implementation of Huffman decoding algorithm
To decode the encoded data we require the Huffman tree. We iterate through the binary encoded data. To find character corresponding to current bits, we use following simple steps.

1. We start from root and do following until a leaf is found.
2.If current bit is 0, we move to left node of the tree.
3.If the bit is 1, we move to right node of the tree.
4.If during traversal, we encounter a leaf node, we print character of that particular leaf node and then again continue the iteration of the encoded data starting from step 1.

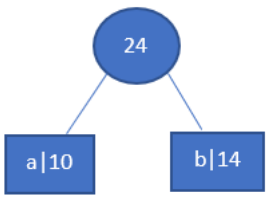
2. Illustration and Algorithms

2.1. Illustration

Character	Frequency
a	10
b	14
c	17
d	18
e	21
f	50

Table 1: Character with their frequencies.

Step 1. Build a min heap that contains 6 nodes, where each node represents root of a tree with single node.
Step 2. Extract two minimum frequency nodes from min heap. Add new internal node with frequency .

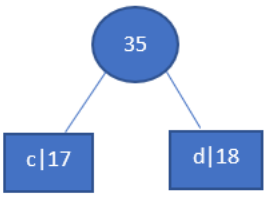


Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

Character	Frequency
c	17
d	18
internal node	24
e	21
f	50

Table 2: Character with their frequencies.

Step 3. Extract two minimum frequency nodes from heap. Add a new internal node with frequency.

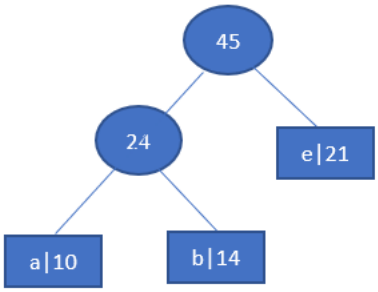


Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes

Character	Frequency
Internal node	24
e	21
Internal node	35
f	50

Table 3: Character with their frequencies.

Step 4. Extract two minimum frequency nodes.Add a new internal node with frequency.

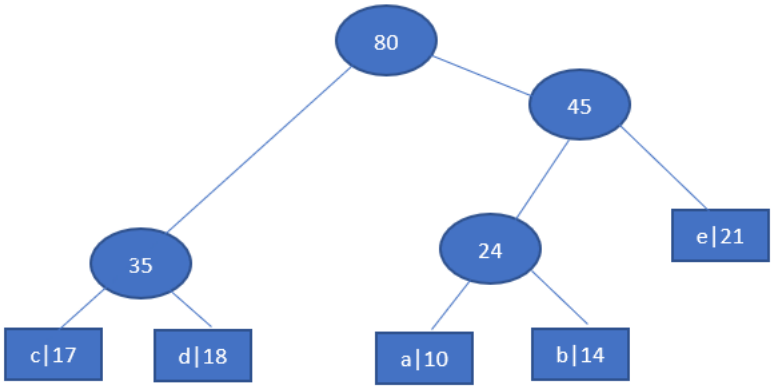


Now min heap contains 3 nodes.

Character	Frequency
Internal node	35
Internal node	45
f	50

Table 4: Character with their frequencies.

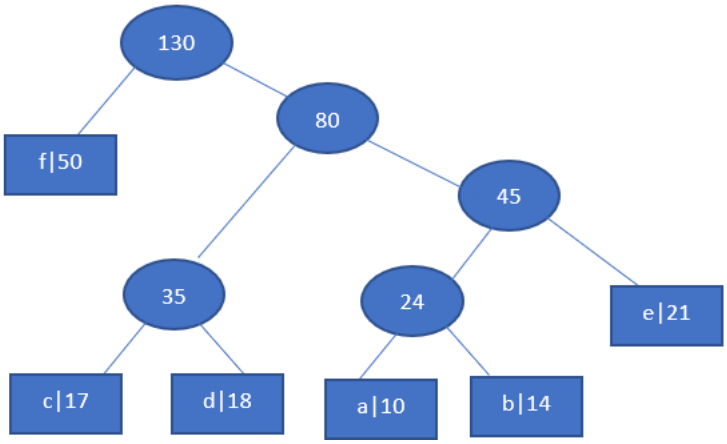
Step 5. Extract two minimum frequency nodes. Add a new internal node with frequency.



Character	Frequency
f	50
Internal node	80

Table 5: Character with their frequencies.

Step 6. Extract two minimum frequency nodes. Add a new internal node with frequency.



Now min heap contains only one node.

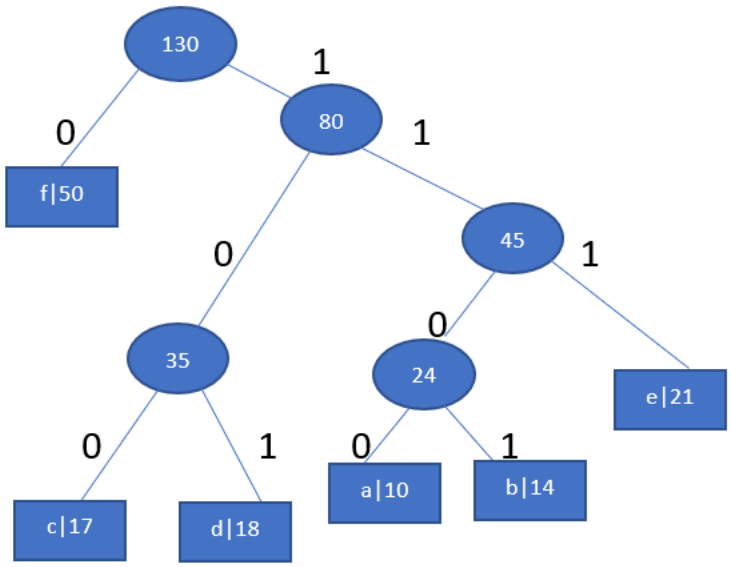
Character	Frequency
Internal node	130

Table 6: Character with their frequencies.

Since the heap contains only one node, the algorithm stops here.

Printing the code using Huffman tree

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



The codes are as follows:

Character	Frequency
f	0
c	100
d	101
a	1100
b	1101
e	111

Table 7: Character with their frequencies.

Total number of characters given are 130.
If We Consider ASCII encoding Total no. of bits used will be 1040 (130*8)bits.
After Encoding given data using Huffman algorithm every character has variable length code.
Total bits used in this case is 314 bits only Which is nearly 70% less.

2.2. Algorithms

In this we are creating 2 structures.
1)Huffman tree node- This node of min heap stores char, frequency of that character and pointer pointing to the left and right child of that node.

2)Huffman Tree- This Structure stores current size of tree, maximum size of tree and pointer pointing to array. Array will store data of Min heap.

Algorithm 1 Heapify (minheap,index)

```
int smallest = index;
int left = 2*index+1;
int right = 2*index+1;
if left < minHeap.size and array[left] < array[smallest] then
    smallest = left;
end if
if right < minHeap.size and array[right] < array[smallest] then
    smallest = right;
end if
if smallest != index then
    exchange A[index] <-> A[smallest];
    Heapify(A, smallest);
end if
```

Algorithm 2 Insertion in min heap (minheap,minheapnode)

```
MinHeap.size++;
int i = minHeap.size - 1;
while i  minHeapNode.freq < array[(i - 1) / 2].freq do
    array[i]=array[i-1/2];
    i = i − 1/2;
end while
array[i]=minheapnode;
```

Algorithm 3 extractmin(minheap)

```
temp=array[0];
array[0]=array[size-1];
size–;
heapify(minheap,0);
```

Algorithm 4 Assigning code to each character(root,arr[],int current)

```
if  (root->left exists) then
    a[current] = 0; printtree(root->left, a, current + 1);
end if
if (root->right exists) then
    a[current] = 1; printtree(root->right, a,current + 1);
end if
if (root is leaf) then
    print a[].
end if
```

Algorithm 5 buildhuffmantree(data[],frequency[],size)

```
new *left,*right,*node;
minheap=createhuffmantree(data,freq,size);
while !IsSize(minHeap) do
    left=extractmin(minheap)
    right=extractmin(minheap)
    node=newnode('random char',left.frequency+right.frequency);
    node.left=left;
    node.right=right;
    insert(minheap,node);
end while
return(extractmin(minheap));
```

3. Conclusions

Huffman Coding can help you compress efficiently text file, but it can also help you do more. . . ethical. . . things like encode images or sounds more efficiently. It's used in some famous image compression algorithms that you might be familiar with like JPEG and PNG, file formats!

While using Huffman coding does not always produce a huge reduction in the amount of bits needed to communicate a message, it's powerful because it's lossless. Even though we reduce the average number of bits needed, we don't lose any of our original message. This is important when we need to make sure that the message we send is exactly the one that is received.

This encoding scheme results in saving lot of storage space, since the binary codes generated are variable in length. Also Huffman encoding is widely used in compression formats like GZIP, PKZIP (winzip) and BZIP2.

4. REFERENCE

1. Ronald L. Rivest Clifford Stein Thomsan H.Cormen, Charles E.Leiserson.Introduction to Algorithms ,3rdedition. MIT Press, 1989.
2. Narasimha Karumanchi.Data Structures and Algorithm Made Easy,. Career Monk `enumerate` environment.
3. https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1126/handouts/220
4. https://math.mit.edu/ goemans/18310S15/huffman-notes.pdf

A. Appendix

Theorem 1 :
If C be an alphabet in which each character c belong to C has frequency c.freq. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the code words for x and y have the same length and differ only in the last bit. (for more details refer http://www.columbia.edu/ cs2035/courses/csor4231.F11/huff.pdf)