Lookup Optimisation of H-Tree

Harshmohan Kulkarni
kulkarnihs20.comp@coep.ac.in

Ashwini Matange
asm.comp@coep.ac.in

***Abstract*—Repeatedly searching on a Depth-First or Breath-First Search fashion, some nodes are visited many times as search is required to start at the root. H-Tree [1] is a structure proposed which uses Hash Table as a primary data structure. We propose to improve this structure to reduce number of lookup operations needed on the H-Tree by balancing the tree. This method also decreases cache size of the H-Tree.**

***Index Terms*—H-Tree, Huffman coding, hash-table, optimisation, Balancing**

I. Introduction

H-Tree was a structure proposed previously. This structure mainly focuses on problems which include a rooted tree and some nodes in the tree are frequently visited. It makes the access to the nodes of this tree faster but at the cost of increased space complexity of the algorithm. This is done by storing the path of a node from its rooted node in a Hash Table linked to the node.

But this structure is not the fastest in all its applications. While using it in some of the applications it can be optimised by performing some operations on the Tree which is being converted to H-Tree to make the H-Tree give paths to the required nodes at a faster rate by reducing the number of look ups required on the Hash Table for getting the path of a node from it's root.

In this paper, we will look at the application of H-Tree in Huffman Encoding. We use Huffman coding is used in a lot of compression algorithms.

II. Problem and Algorithm

For using the Huffman Encoding technique for compression of text files, initially we count the frequency of each character from the text file. The we add it to a Min-Heap. Then we extract two elements from the Min-Heap in each iteration, form a head pointing to both the elements as the one with less value in the left and other in the left. Then we put the new element back in the Min-Heap with its value as sum of values of both the elements it is pointing. We repeat this procedure till only one element is left in the Min-Heap which is the root node of the Huffman tree formed.

Then according to the algorithm of H-Tree creation, the Huffman tree is traversed and paths of leaf nodes (which are the nodes containing characters in the Huffman Tree) are stored in the Hash Table linked with the character.

While decoding the compressed file, we perform look up on the Hash Table for searching the code of the character which we have read. If the code is not present in the Hash Table, then we read the next bit of the code and perform operations to similar to above.

But, as the difference between the minimum and maximum length of the codes for characters in the encoded file can be large at times, the look ups performed on the Hash Table increase marginally and the algorithm of decoding slows down.

To conquer this problem, we perform balancing of the tree while the Huffman tree is built. This ensures that the heights of the elements in the tree do not cross a certain threshold helping in making the look up operations on the Hash Table faster.

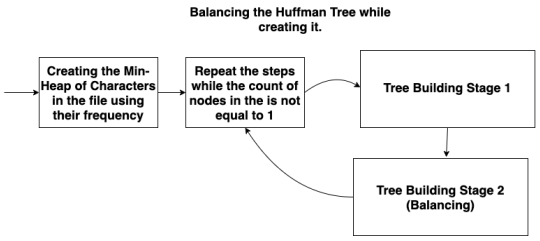

Fig. 1. Algorithm for Balancing

A. Initialisations

Create a Min-Heap containing the frequencies of all the characters in the text file to be compressed. Repeat Stage 1 and two till only one element is left in the Min-Heap.

B. Tree Building Stage 1

- Remove first two elements in the Min-Heap and create a new node. Design all elements in such a way that the can point to two other elements, store a value and the character (we'll store the value of character on the element only if it is a leaf node).
  NOTE : Any representation of Tree can be taken into consideration.
- Make the left pointer of the element point to the element with less value of frequency from both the nodes extracted and right pointer to other node. Assign the value of new node as the addition of values of left and right node.

C. Tree Building Stage 2 (Balancing)

- Check the balance factor of the new node created and perform LL and RR rotations if required. Repeat this procedure for left as well as right elements.
  NOTE : Only LL and RR rotations are considered as

Identify applicable funding agency here. If none, delete this.

the other two may hamper the condition of keeping the elements with characters as leaf elements.
- Add the node back into the Min-Heap.

III. Conclusion and Results

In this section we present various results from experiments conducted on our own implementation of the algorithm. We coded different text files from The Canterbury Corpus benchmark file set [2], and then proceeded to decode the resulting files using an H-Tree data structure, counting the number of hash table lookup operations needed. The data set consisted on the following text files:

TABLE I
DESCRIPTION

File	***About***	***Character***
alice29.txt	English text	152090
Lcet10.txt	Technical writing	427655
Bible.txt	The King James version of the bible	4047393
World192.txt	The CIA world fact book	2473401
E.coli	Complete genome of the E. Coli bacterium	4638691
Alphabet.txt	Repetitions of the alphabet	100001
Pi.txt	The first million digits of pi	1000001

In the tables given below, the length of the longest and the shortest Huffman code for a file are given for balanced as well as unbalanced Huffman Tree.

TABLE II
HUFFMAN CODE LENGTH FOR BALANCED HUFFMAN TREE

File	***Minimum Length***	***Maximum Length***	***Difference***
alice29.txt	5	8	3
Lcet10.txt	4	8	4
Bible.txt	4	8	4
World192.txt	5	9	4
E.coli	2	2	0
Alphabet.txt	4	5	1
Pi.txt	3	4	1

TABLE III
HUFFMAN CODE LENGTH FOR UNBALANCED HUFFMAN TREE

File	***Minimum Length***	***Maximum Length***	***Difference***
alice29.txt	2	17	15
Lcet10.txt	3	16	13
Bible.txt	2	17	15
World192.txt	3	20	17
E.coli	2	2	0
Alphabet.txt	4	5	1
Pi.txt	3	4	1

The table given below contains the Look up ratio reduction of the H-Tree with and without balancing the Huffman Tree. The look up reduction ratio is calculated as the ratio of number of look ups required on the Huffman Tree and the look ups on the tree from which the Huffman Tree is made.

Some observations made on the the tables given above:

TABLE IV
LOOK UP RATIO FOR UNBALANCED AND BALANCED HUFFMAN TREE

File	***Unbalanced***	***Balanced***
alice29.txt	1.27	2.67
Lcet10.txt	1.74	1.86
Bible.txt	1.29	1.89
World192.txt	1.65	2.42
E.coli	2	2
Alphabet.txt	2.6957	2.6957
Pi.txt	2.43	2.43

- As the difference between the Maximum and Minimum length decreases, the Look up ratio attains a better value.
- On and average there is 50% increase in the lookup ratio on trying on some more data sets.
- With increase in size of the file being compressed, the look up ratio increases marginally.

References

[1] H-Tree: A data structure for fast path-retrieval in rooted trees.
[2] Tim Bell, Matt Powell, "The Canterbury Corpus"