Experiment No. : 4

Title: Implement Huffman Algorithm using Greedy approach

Batch:B1 **Roll No.:16010420133** **Experiment No.:4**

Aim: To Implement Huffman Algorithm using Greedy approach and analyse its time Complexity.

Algorithm of Huffman Algorithm: Refer Coreman for Explaination

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n - 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Explanation and Working of Variable Length Huffman Algorithm:
Huffman coding (also known as Huffman Encoding) is an algorithm for doing data compression, and it forms the basic idea behind file compression. This post talks about the fixed-length and variable-length encoding, uniquely decodable codes, prefix rules, and Huffman Tree construction.
The idea to compress data can be implemented by using “variable-length encoding”. We can exploit the fact that some characters occur more frequently than others in a text (refer to this) to design an algorithm that can represent the same piece of text using a lesser number of bits. In variable-length encoding, we assign a variable number of bits to characters depending upon their frequency in the given text. So, some characters might end up taking a single bit, and some might end up taking two bits, some might be encoded using three bits, and so on.

Working:
The technique works by creating a binary tree of nodes. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the character itself, the weight (frequency of appearance) of the character. Internal nodes contain character weight and links to two child nodes. As a common convention, bit 0 represents following the left child, and a bit 1 represents following the right child. A finished tree has n leaf nodes and n-1 internal nodes. It is recommended that Huffman Tree should discard unused characters in the text to produce the most optimal code lengths.

Steps:

- Create a leaf node for each character and add them to the priority queue.
- While there is more than one node in the queue:

1. Remove the two nodes of the highest priority (the lowest frequency) from the queue.
2. Create a new internal node with these two nodes as children and a frequency equal to the sum of both nodes' frequencies.
3. Add the new node to the priority queue.

- The remaining node is the root node and the tree is complete.

Derivation of Huffman Algorithm:

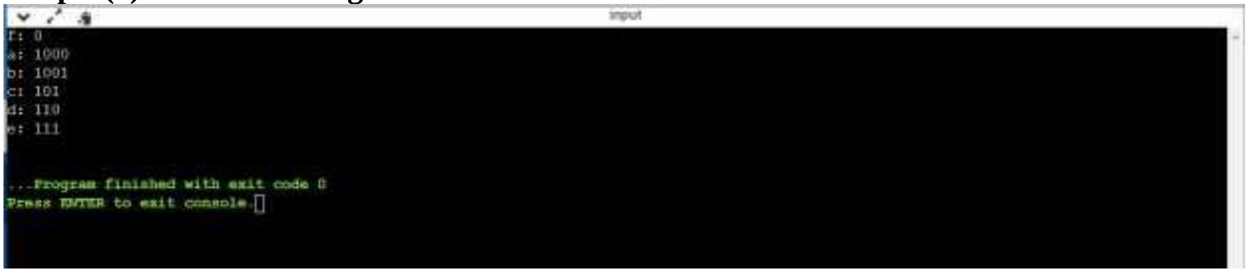
Time complexity Analysis
The time complexity for encoding each unique character based on its frequency is O(nlog n).
Using a heap to store the weight of each tree, each iteration requires O(logn) time to determinethe cheapest weight and insert the new weight. There are O(n) iterations, one for each item.

Program(s) of Huffman Algorithm:

```
#include <bits/stdc++.h>
using namespace std;
struct MinHeapNode {
char data;
unsigned freq;
MinHeapNode *left, *right;
MinHeapNode(char data, unsigned freq)
{
left = right = NULL;
this->data = data;
this->freq = freq;
}
};
struct compare {
bool operator()(MinHeapNode* l, MinHeapNode* r)
{
return (l->freq > r->freq);
}
};
void printCodes(struct MinHeapNode* root, string str)
{
```

```
if (!root)
return;
if (root->data != '$')
cout << root->data << ": " << str << "\n";
printCodes(root->left, str + "0");
printCodes(root->right, str + "1");
}
void HuffmanCodes(char data[], int freq[], int size)
{
struct MinHeapNode *left, *right, *top;
priority_queue<MinHeapNode*, vector<MinHeapNode*>, compare> minHeap;
for (int i = 0; i < size; ++i)
minHeap.push(new MinHeapNode(data[i], freq[i]));
while (minHeap.size() != 1) {
left = minHeap.top();
minHeap.pop();
right = minHeap.top();
minHeap.pop();
top = new MinHeapNode('$', left->freq + right->freq);
top->left = left;
top->right = right;
minHeap.push(top);
}
printCodes(minHeap.top(), "");
}
int main()
{
char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
int freq[] = { 6, 8, 15, 17, 20, 65 };
int size = sizeof(arr) / sizeof(arr[0]);
HuffmanCodes(arr, freq, size);
return 0;
}
```

Output(o) of Huffman Algorithm:



```
f: 0
a: 1000
b: 1001
c: 101
d: 110
e: 111


...Program finished with exit code 0
Press ENTER to exit console.
```

Post Lab Questions:- Differentiate between Fixed length and Variable length Coding with suitable example.
Fixed length encoding means that each thing you encode ends up the same length.

Variable length encoding means that it may not.
Onecharacter might be of 1 bit, other of 3 bits, another of 2 bits.

In the below table, Code 1 is a fixed-length code, and, code 2 and code 3 are variable-length codes.

xi	Code 1	Code 2	Code 3
X1	00	0	0
X2	01	1	10
X3	10	00	110
X4	11	11	111

Conclusion: (Based on the observations):
Learnt to Implement Huffman Algorithm using Greedy approach and analyse its time Complexity. I understood and implemented the variable length Huffman coding algorithm. The Huffman Coding turns to be a simple and efficient way to encode data into a short representations without loosing any piece of information.

Outcome:
CO 1: Analyse time and space complexity of basic algorithms.
CO 2: Implement Greedy and Dynamic Programming algorithms.

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.