**Compression code:-**

We use Huffman coding algorithm , it is a lossless data compression algorithm

in this method we assign the length of code as a variable to input characters lengths of the assigned codes are based on the frequencies of corresponding characters. Which, The most frequent character has the smallest code and the least frequent character has the largest code.

For example if we have string = “ ABACCAABB”, A will have the smallest code because it has the most frequent and C will have the largest code because it has the least frequent.

**To build the code for Huffman coding we have two steps:-**

1. Build a Huffman tree from input characters.
2. Traverse the Huffman tree and assign codes to characters.

Steps to carry out (Build a Huffman tree from input characters):-

1. Build a leaf node for each character and build a min heap of all leaf nodes. Min Heap is used as a priority queue. The value of frequency is used to compare two nodes in min heap. Initially, the least frequent character is at root.
2. Extract two nodes which have the minimum frequency from the min heap.
3. build a new internal node which has 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.
4. Repeat the steps b and c until the heap contains only one node. The remaining node is the root node and the tree is complete.
5. After these steps we carry out the step 2 Traverse the Huffman tree and assign codes to characters.

**Graph to show Huffman coding method:-**

Diagram

Description automatically generated

**In this graph, we show that any right branch has ‘0’ binary and any left branch has ‘1’ binary.**

**Background of Huffman Algorithm:-**

1. **Step 1** : get frequencies
2. Scan the file which I want to compress it and count the occurrence of each character.
3. Sort the character based on their frequency
4. **Step 2**: Build tree and assign codes
5. Build a Huffman-code tree (binary tree).
6. Traverse the tree to assign codes.
7. **Step 3**: Encoding (compression)
8. Scan the file again and replace each character by its code.
9. **Step 4**: Decoding (decompression)

i. Huffman tree is the key to decompress the file.

1. Step 5: I used stackOverFlow to get answer of some of my problems.

* **Time complexity of Huffman algorithm:-**

The time complexity of the Huffman algorithm is O(nlogn). Using a heap to store the weight of each tree, each iteration requires O(logn) time to determine the cheapest weight and insert the new weight.

* **Space complexity of Huffman algorithm:-**

for every encoded symbol you have to traverse the tree in order to decode that symbol. The tree contains m nodes and, on average, it takes O(log m) node visits to decode a symbol. So the time complexity would be

O(n log m). Space complexity is O(m) for the tree and O(n) for the decoded text.

* Text

  Description automatically generatedScreen shots:-

