HUFFMAN CODING

NAME: KAWSAR AHMED

ID: 170041021

WHAT IS HUFFMAN CODING?

- Huffman coding is a lossless data compression algorithm.
- In this algorithm, a variable-length code is assigned to input different characters. The code length is related to how frequently characters are used. Most frequent characters have the smallest codes and longer codes for least frequent characters.
- There are mainly two parts. First one to create a Huffman tree, and another one to traverse the tree to find codes.
- Complexity for assigning the code for each character according to their frequency is O(n log n)

WHAT ARE THE BENEFITS OF HUFFMAN CODING?

- Huffman codes provide two benefits:

1. They are space efficient given some corpus.

2. They are prefix codes.

- Given some set of documents for instance, encoding those documents as Huffman codes is the most space efficient way of encoding them, thus saving space. This however only applies to that set of documents as the codes you end up are dependent on the probability of the tokens/symbols in the original set of documents. The statistics are important because the symbols with the highest probability (frequency) are given the shortest codes. Thus the symbols most likely to be in your data use the least amount of bits in the encoding, making the coding efficient.

- The prefix code part is useful because it means that no code is the prefix of another. In morse code for instance A = dot dash and J = dot dash dash dash, how do you know where to break reading the code. This increases the inefficiency of transmitting data using morse as you need a special symbol (pause) to signify the end of transmission of one code. Compare that to Huffman codes where each code is unique, as soon as you discover the encoding for a symbol in the input, you know that that is the transmitted symbol because it is guaranteed not to be the prefix of some other symbol.

HOW DOES HUFFMAN CODING WORK?

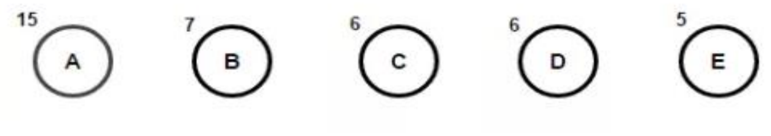
- The algorithm works by creating a binary tree of nodes. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the character itself, the weight (frequency of appearance) of the character. Internal nodes contain character weight and links to two child nodes. As a common convention, bit 'O' represents following the left child and 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.
- We will use priority queue for building Huffman tree where the node with lowest frequency is given highest priority. Below are the complete steps -

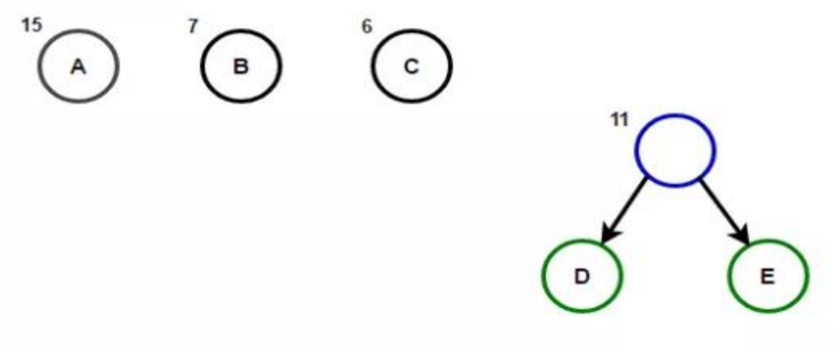
1. Create a leaf node for each character and add them to the priority queue.

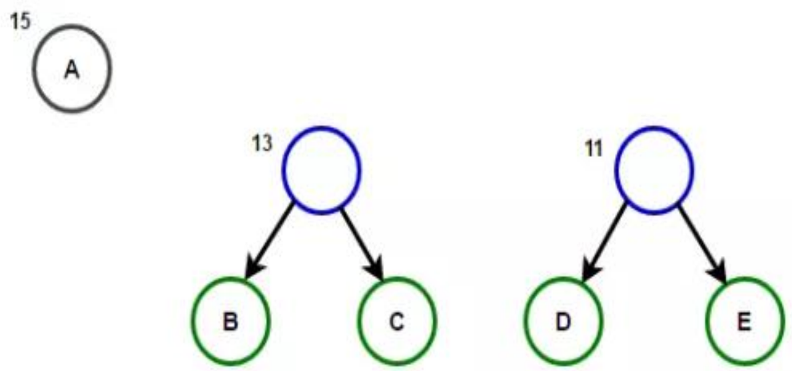
2. While there is more than one node in the queue:
   (i) Remove the two nodes of highest priority (lowest frequency) from the queue
   (ii) Create a new internal node with these two nodes as children and with frequency equal to the sum of the two nodes' frequencies.
   (iii) Add the new node to the priority queue.
3. The remaining node is the root node and the tree is complete.

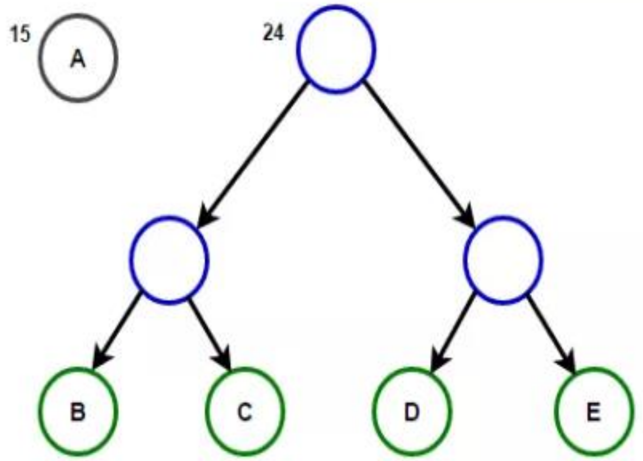
CREATING A HUFFMAN TREE

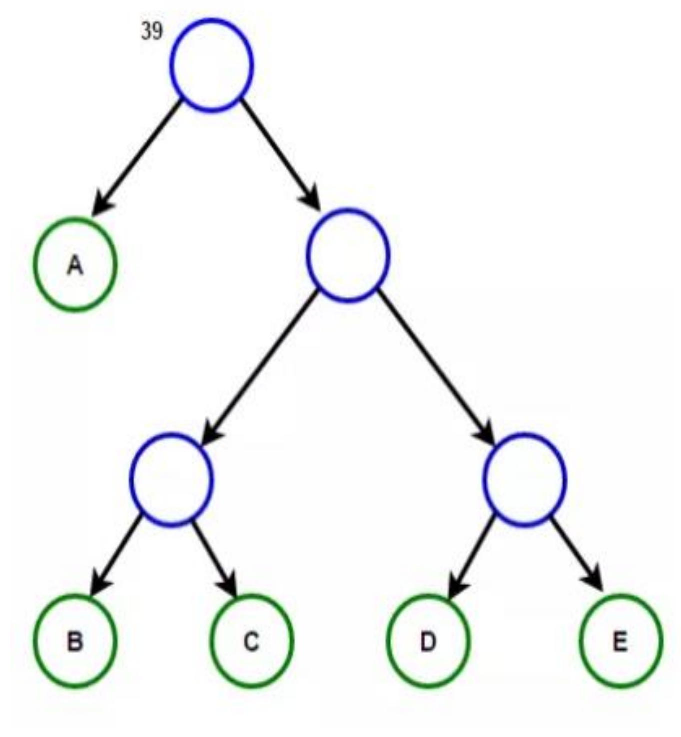
- Consider some text consisting of only 'A', 'B', 'C', 'D' and 'E' character and their frequencies are 15, 7, 6, 6, 5 respectively.
- Creation of the HUFFMAN TREE for this scenario according to the above mentioned algorithm is illustrated below-

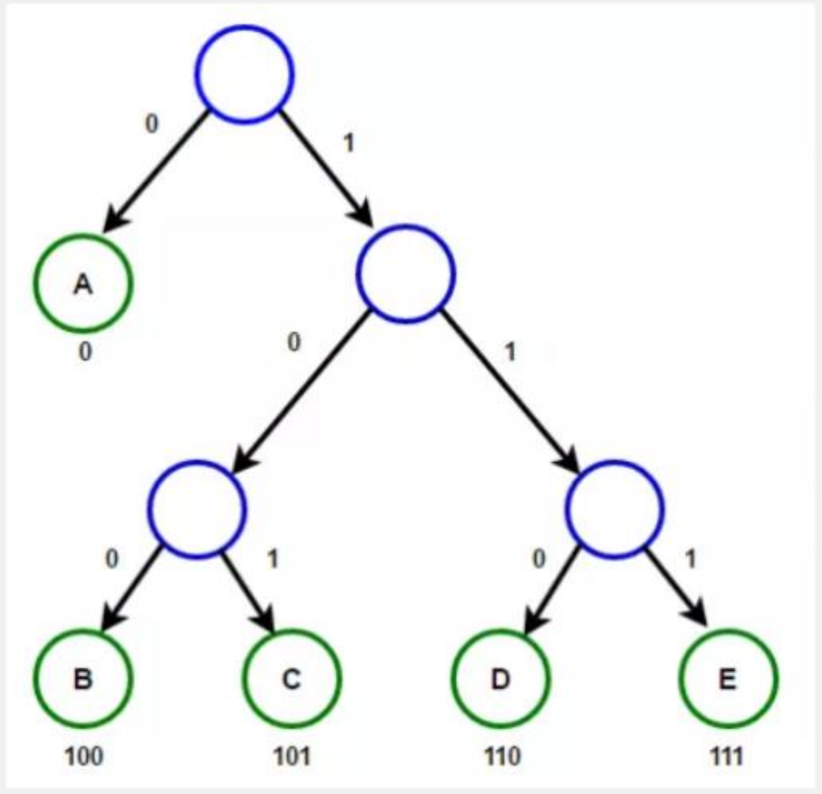





15
A
13
11
B
C
D
E


15
A
24
B
C
D
E





Huffman Tree

The path from root to any leaf node stores the Huffman Code corresponding to character associated with that leaf node.