Huffman Algorithm:

Compression technique used for reducing the size of data .

Example:

Message :a c c a b b d d a e c c b b a e d d c c

Length:20

A : 6 5 binary format :01000001

B : 66 binary :01000010 , C:67 , D :68 , E : 69

8 bit for each alphbit so it will be 20 \*8 = 160 bit

|  |  |  |
| --- | --- | --- |
| Code | Count / frequency | Character |
| 000 | 3 3/20 | A |
| 001 | 5 5/20 | B |
| 010 | 6 6/20 | C |
| 011 | 4 4/20 | D |
| 100 | 2 2/20 | E |

5 Cha \* 8 bits=40 char

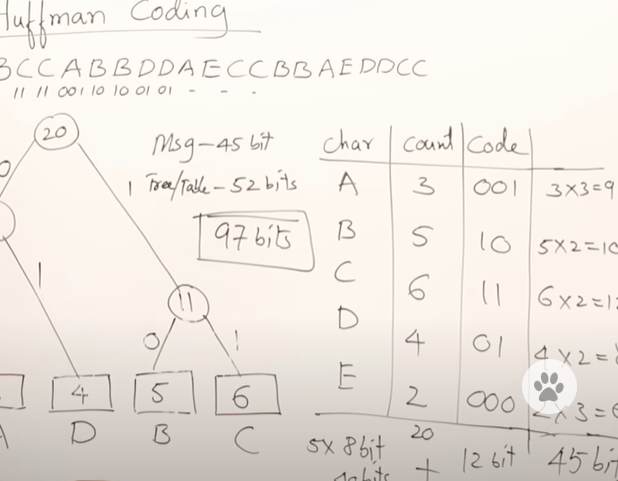
5 cha \* 3 bit = 15 cod

15+40=55

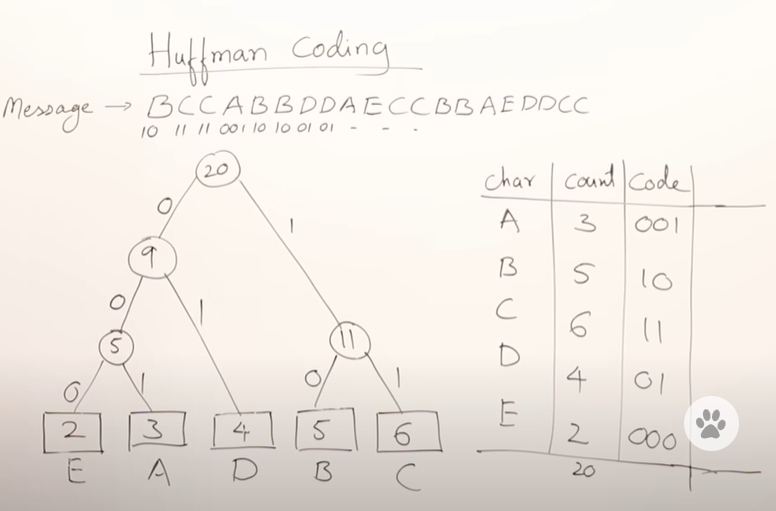
Message :20 \* 3 bits code = 60

Will reduce from 160 to 115

After making reducing



1. Calculate the frequency of each character in the input data.
2. Create a priority queue (min-heap) containing all the characters along with their frequencies.
3. While there is more than one node in the queue:
   * Remove the two nodes with the lowest frequency from the queue.
   * Create a new internal node with a frequency equal to the sum of the frequencies of the two nodes. Make the two nodes its left and right children.
   * Add the new node to the priority queue.
4. The remaining node in the priority queue is the root of the Huffman tree.
5. Traverse the Huffman tree from the root to each leaf, assigning a binary code (0 or 1) to each edge. Typically, going left corresponds to adding a 0 to the code, and going right corresponds to adding a 1.
6. Encode the input data using the generated Huffman codes.



Floyed\_warshall:all-pair shortest path algorithm

The Floyd-Warshall algorithm is used to find the shortest paths between all pairs of vertices in a weighted graph.

1. **Initialization**:
   * Create a distance matrix **dist** of size **n x n**, where **n** is the number of vertices in the graph.
   * Initialize the elements of the matrix **dist** with the weights of the edges between the vertices. If there is no direct edge between two vertices, set the corresponding element to infinity.
   * Also, set the diagonal elements of the matrix to zero since the shortest distance from a vertex to itself is always zero.
2. **Main Algorithm**:
   * For each intermediate vertex **k** (from 1 to **n**):
     + For each pair of vertices **i** and **j** (from 1 to **n**):
       - Check if the shortest path from **i** to **j** passing through vertex **k** is shorter than the current shortest path from **i** to **j**.
       - If so, update the distance **dist[i][j]** to be the sum of distances **dist[i][k]** and **dist[k][j]**.
3. **Termination**:
   * After completing the above steps for all vertices, the distance matrix **dist** will contain the shortest distances between all pairs of vertices.
4. **Handling Negative Cycles**:
   * The Floyd-Warshall algorithm can also detect negative cycles in the graph. A negative cycle is a cycle whose total edge weights sum to a negative value.
   * If the diagonal elements of the distance matrix become negative after the algorithm finishes, it indicates the presence of negative cycles.
5. **Complexity**:
   * The time complexity of the Floyd-Warshall algorithm is O(n^3), where **n** is the number of vertices in the graph.
   * This makes it suitable for finding shortest paths in dense graphs (graphs with many edges) where other algorithms like Dijkstra's algorithm might be less efficient.

