Data compression

Data compression is the process of reducing the size of a data file. It can be classified into two main types: lossless compression and lossy compression. Both types aim to reduce the amount of data needed to store or transmit information, but they differ significantly in their approach and applications.

Lossless Compression

Lossless compression techniques reduce file size without losing any data. When the file is decompressed, it is restored to its original state with no loss of information. This type of compression is essential for applications where every bit of original data is important, such as text files, executable programs, and sensitive data storage.

Example:

- ZIP files: When you compress files into a ZIP archive, you can later unzip them to retrieve the exact original files.
- PNG images: PNG uses lossless compression, so images retain all their original detail and quality.

Data compression

Lossless Compression Techniques :

- **Run-Length Encoding (RLE):** Compresses sequences of repeating characters.
  - Example: The string "AABBCCDDEE" can be compressed to "5A3B2C1D2A".

- **Huffman Coding:** Uses variable-length codes to represent characters based on their frequencies.
  - Example: In a text with characters 'a' and 'b' where 'a' is more frequent, 'a' might be encoded as '0' and 'b' as '1'.

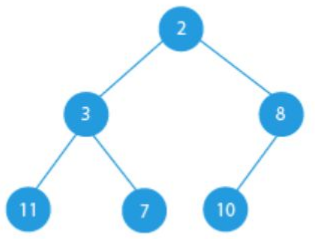
Min-Heap in Huffman Coding

Min-Heap:

- A data structure where the smallest element is always accessible at the root.
- Efficient for retrieving and managing the smallest element.

Role in Huffman Coding:

- **Initial Setup:** Insert each character with its frequency into the min-heap.
- **Processing:** Repeatedly extract the two nodes with the smallest frequencies, combine them into a new node, and reinsert this node into the heap.
- **Tree Construction:** Continue until a single node remains, which represents the root of the Huffman tree.

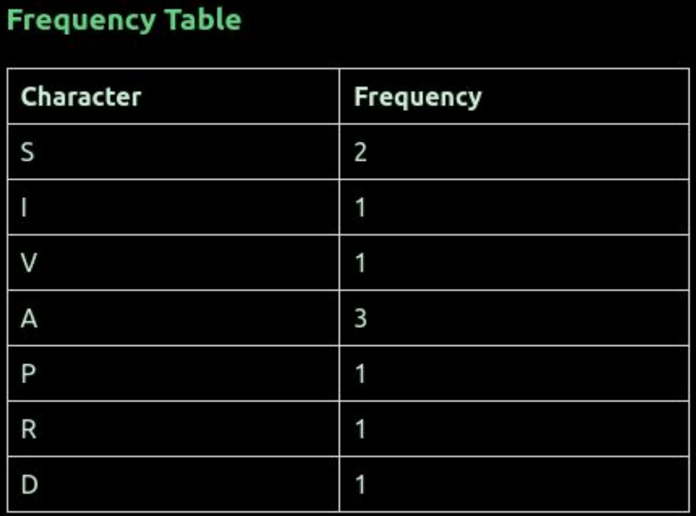


Min-Heap

Character Frequency

Character frequency is fundamental to the Huffman coding algorithm, which is a popular method for lossless data compression. The basic idea behind Huffman coding is to use shorter codes for more frequent characters and longer codes for less frequent characters. This approach minimizes the overall length of the encoded data, resulting in efficient compression.

Eg : “SIVAPRASAD”



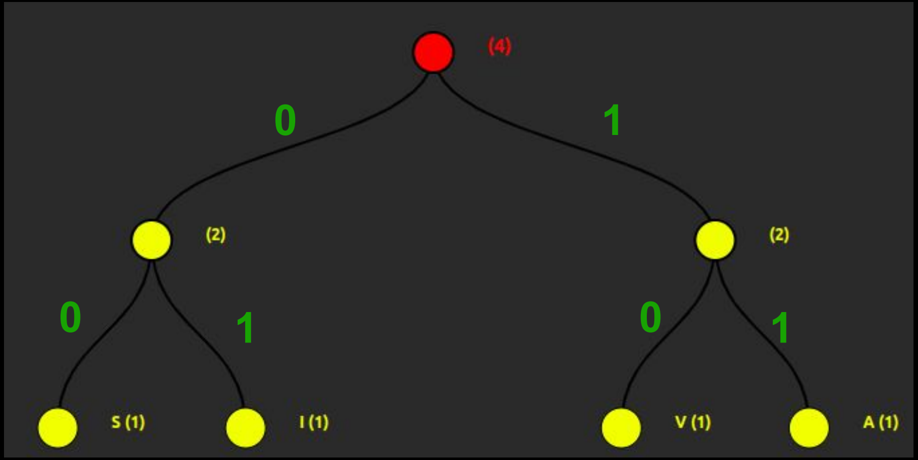
Frequency Table

Character	Frequency
S	2
I	1
V	1
A	3
P	1
R	1
D	1

Huffman Tree

What is a Huffman Tree?

A Huffman Tree is a type of binary tree used for efficient data compression. It represents characters and their frequencies in such a way that more frequent characters are closer to the root of the tree, minimizing the overall number of bits needed to encode data.



Building the Huffman Tree

Start with Nodes : Each character is represented as a leaf node with its frequency.

Combine Nodes : Create a priority queue (or min-heap) where nodes are ordered by frequency. The node with the smallest frequency has the highest priority.

Combine Nodes:

1) Extract the two nodes with the lowest frequencies from the priority queue.
2) Create a new internal node with these two nodes as children. The frequency of this new node is the sum of the frequencies of the two child nodes.
3) Insert this new node back into the priority queue.

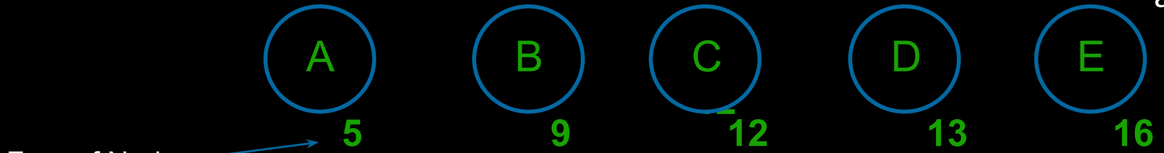
Repeat: Continue extracting the two lowest-frequency nodes, combining them, and inserting the new nodes back into the priority queue until only one node remains. This final node is the root of the Huffman Tree.

Creation: Internal nodes are created by combining two nodes and setting them as children of the new node.

Insertion: Each new internal node is inserted back into the priority queue to be considered in subsequent combinations.

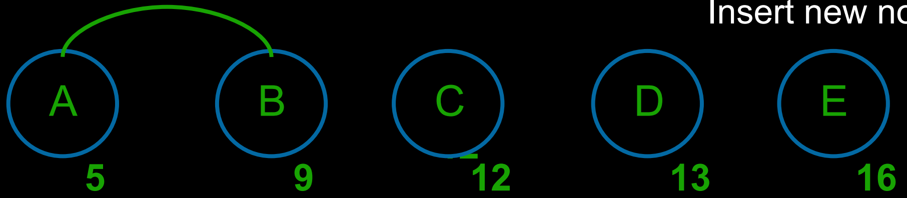
Example of Huffman Tree

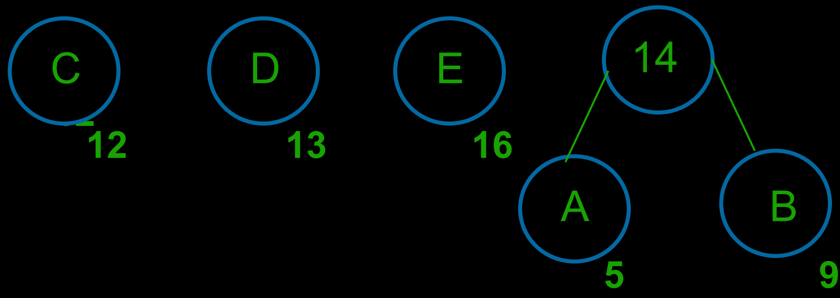
Arrange nodes with ascending freq



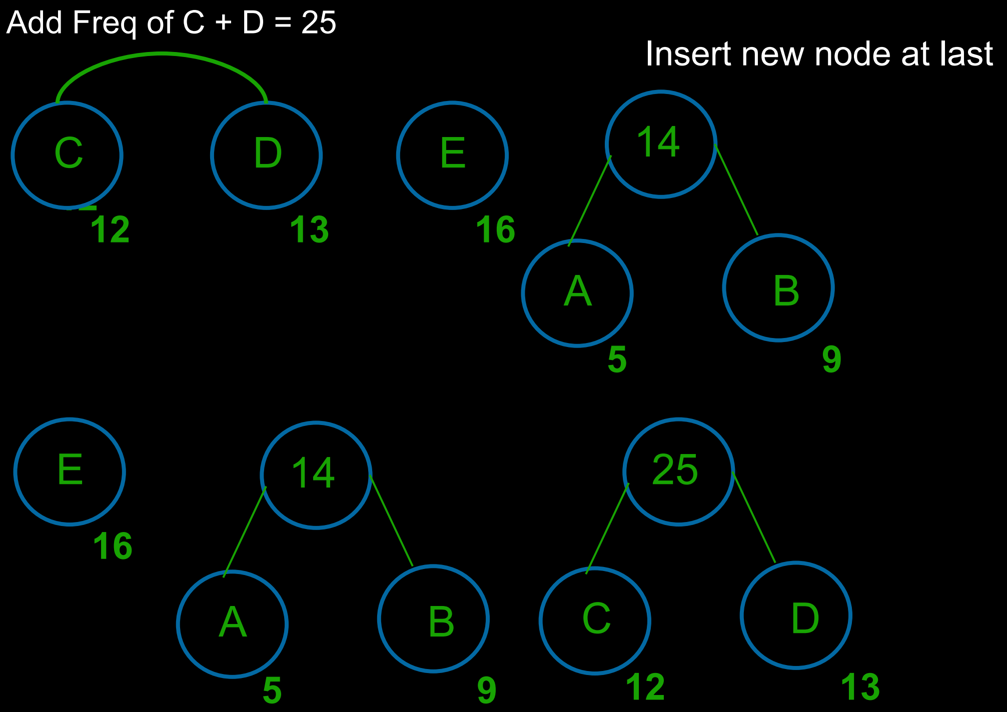
Add Freq of A + B = 14

Insert new node at last

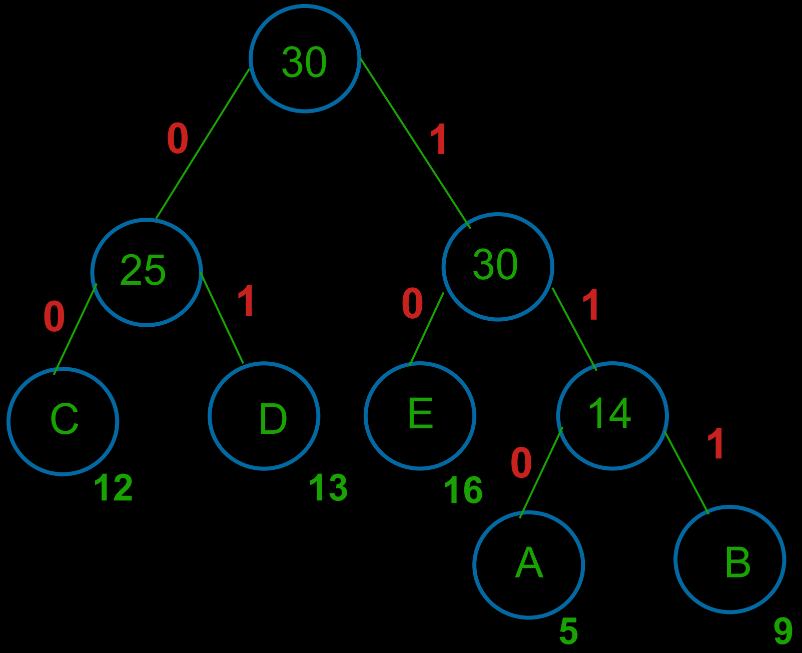




Example of Huffman Tree



Example of Huffman Tree


30
0
1
25
30
0
1
0
1
C
D
E
14
12
13
16
0
1
A
B
5
9