Huffman Coding

A Lossless Data Compression Technique

Introduction

Data compression: Reduces the size of data for storage or transmission.

Huffman Coding/ Encoding is used for the lossless compression of data.

It assigns variable length code to all the characters depending on the frequency of characters in the given message. It assigns shorter codes to frequent characters and longer codes to infrequent characters.

Developed by: David A. Huffman in 1952

Huffman coding Steps

1. Calculate the frequency of each character.
2. Build a priority queue (min-heap) with characters and their frequencies.
3. Construct the Huffman tree by combining the least frequent characters.
4. Generate the Huffman codes based on the tree.
5. Encode and decode the data using the generated codes.

Example **Step1** (Create Frequency Table)

Input String: "ABRACADABRA"

Frequency of characters:

A: 5

B: 2

R: 2

C: 1

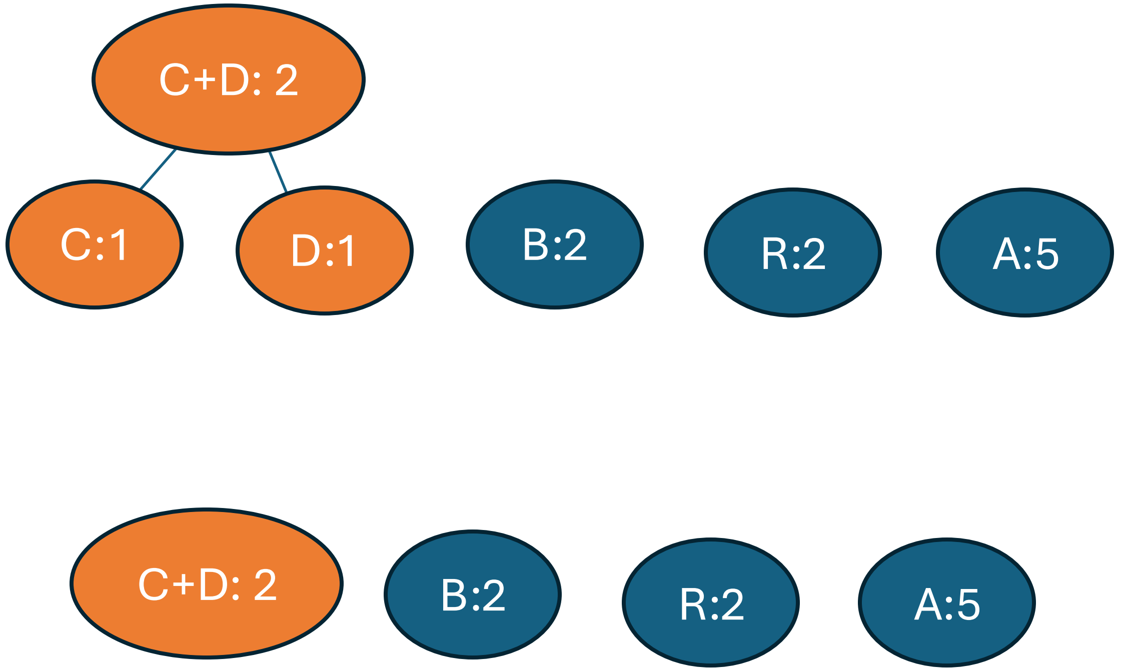
D: 1

Step 2: Build the Priority Queue (Min-Heap)

Initially, each character is placed in a min-heap based on its frequency:

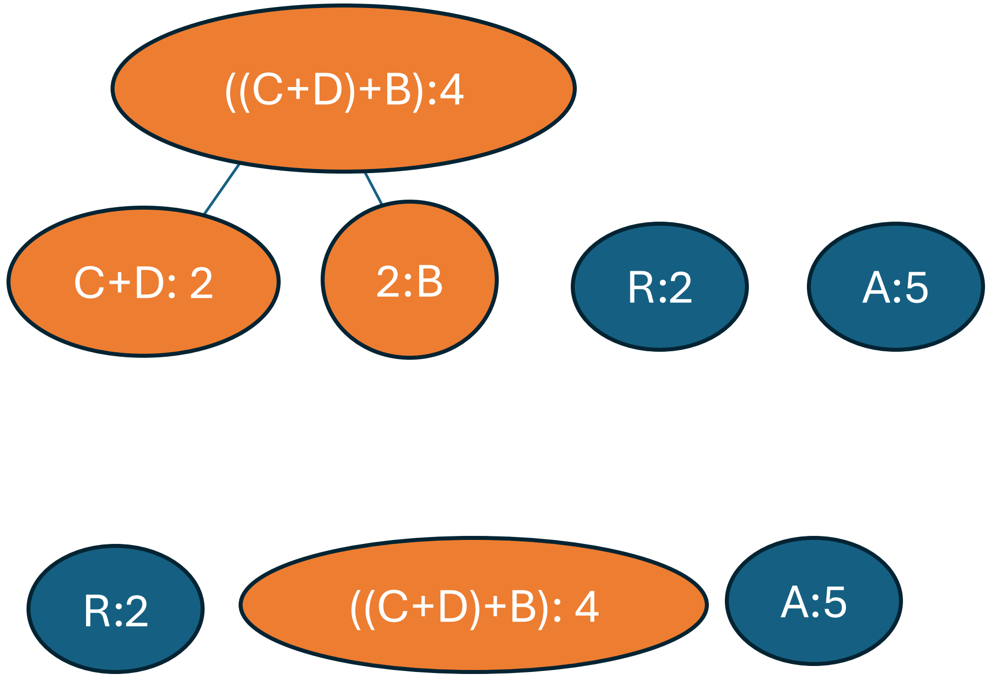
Min-heap: [C:1, D:1, B:2, R:2, A:5]

Step 3: Build the Huffman Tree



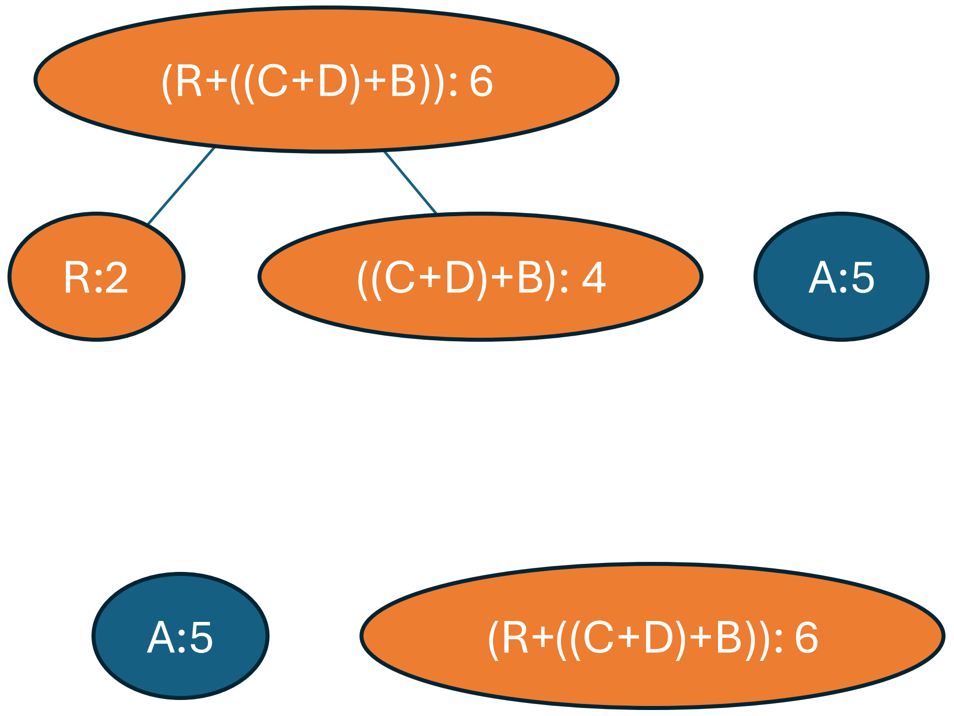
- Iteration 1:
  - Combine nodes C (1) and D (1) to create a new node with frequency 2 (C+D).
  - New heap: [(C+D):2, B:2, R:2, A:5]

Step 3: Build the Huffman Tree



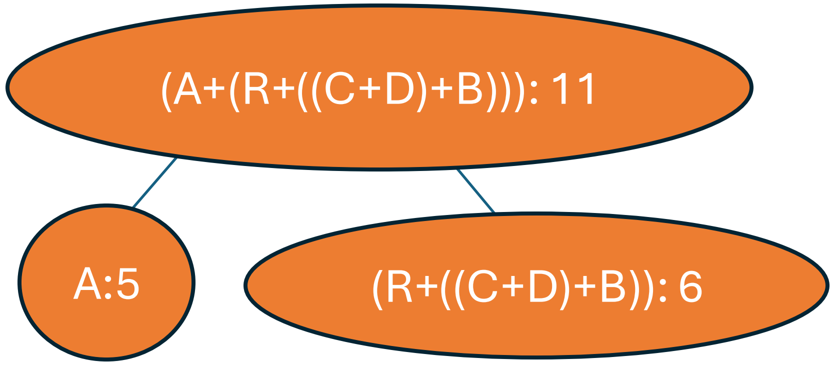
- Iteration 2:
  - Combine nodes B (2) and (C+D) (2) to create a new node with frequency 4 ((C+D)+B).
  - New heap: [R:2, ((C+D)+B):4, A:5]

Step 3: Build the Huffman Tree



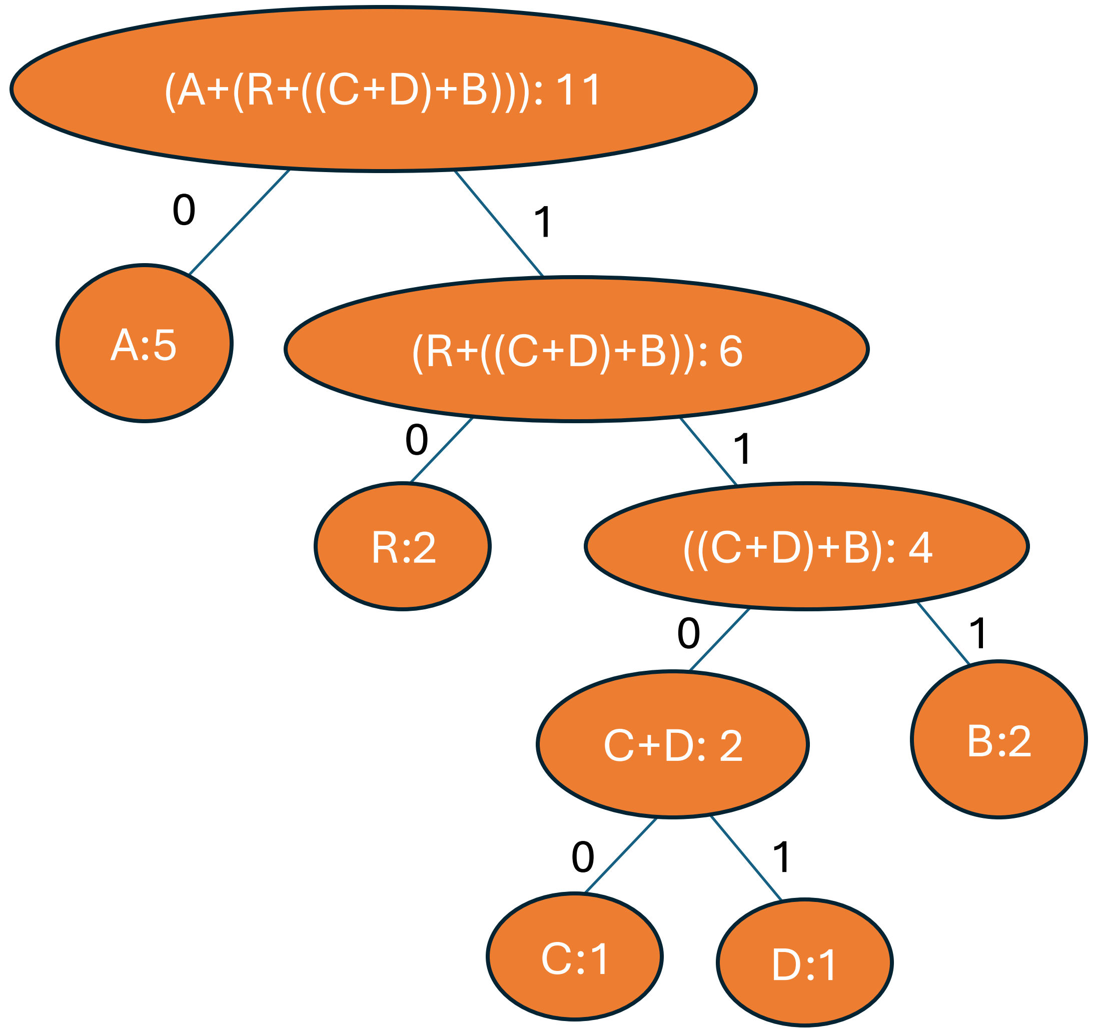
- Iteration 3:
  - Combine nodes R (2) and ((C+D)+B) (4) to create a new node with frequency 6 (R+((C+D)+B)).
  - New heap: [A:5, (R+((C+D)+B)):6]

Step 3: Build the Huffman Tree



- Iteration 4:
  - Combine nodes A (5) and (R+((C+D)+B)) (6) to create the final tree.
  - Final tree: [(A+(R+((C+D)+B))):11]

Step 4: Generating Huffman codes



- Traverse the Huffman tree from root to leaves:
- Left branch = 0, Right branch = 1.
- Final Huffman Codes:
  - A: 0
  - B: 111
  - R: 10
  - C: 1100
  - D: 1101

Step 5: Encode the String

- **Input String:** "ABRACADABRA"
- Using the Huffman codes:
  - A → 0
  - B → 111
  - R → 10
  - A → 0
  - C → 1100
  - A → 0
  - D → 1101
  - A → 0
  - B → 111
  - R → 10
  - A → 0
- **Encoded String: 01111001100011010111100**
  - `01111001100011010111100`

Compression Efficiency

- → Original Input:
  - "ABRACADABRA" (11 characters, 88 bits if using 8-bit ASCII).
- → Huffman Encoded String:
  - Message size= = ∑ (frequency x Code length) =23 bits
  - Table size=5*8 (8 bit for each character)**+**(1+2+3+4+4) (code size of each character)=54 bits
- → Compression Ratio:
  - Original Size: 88 bits
  - Compressed Size= (Message size + Table size) =23+54=77 bits
  - Compression Ratio: 88/77 ≈ 1.14
  - Each character in the example above will require 3 bits if we take into account the fixed code size.

Time Complexity

- The time complexity analysis of Huffman Coding is as follows-
  - extractMin( ) is called 2 x (n-1) times if there are n nodes.
  - As extractMin( ) calls minHeapify( ), it takes O(logn) time.

- Thus, Overall time complexity of Huffman Coding becomes **O(nlogn)**, where n is the number of unique characters in the given text.

Applications of Huffman Coding

File Compression: ZIP files, GZIP

Image Compression: JPEG

Text File Compression: LZW-based formats like GIF

Conclusion

Huffman Coding is an efficient, widely used compression algorithm that achieves lossless compression by assigning shorter codes to frequent characters and longer codes to rare ones.

It is highly effective when data has a skewed frequency distribution, but it requires preprocessing and might not perform well with uniformly distributed data.