## **LEC-12 File/Code Compression**

## **Through Huffman coding scheme**

* Huffman coding is a popular algorithm used for data compression.
* It is a variable-length prefix coding scheme that assigns shorter codes to more frequently occurring characters and longer codes to less frequently occurring characters.
* Huffman coding is widely used in applications like file compression (e.g., ZIP files), image compression (e.g., JPEG), and video compression (e.g., H.264).

**Algorithm Steps:**

## **Frequency Calculation:**

Calculate the frequency of each character in the given text or data.

## **Priority Queue (Min Heap):**

Create a priority queue (min heap) of nodes, each containing a character and its frequency.

## **Build Huffman Tree:**

**While there is more than one node in the priority queue:**

* Remove the two nodes with the lowest frequencies.
* Create a new node with a frequency equal to the sum of the two removed nodes' frequencies. This node becomes the parent of the two removed nodes.
* Insert the new node back into the priority queue.

## **Generate Huffman Codes:**

* Traverse the Huffman tree from the root.
* Assign '0' to the left edge and '1' to the right edge.
* The path from the root to a character's node represents its Huffman code.

**Generic Code Compression:**

File (a,b,c,d,e)

**Let’s say:**

Above file consists of different words. All the data of the file has following letters/characters: a,b,c,d,e

|  |  |  |
| --- | --- | --- |
| **Letters** | **Frequency/Probability** | **Code(Generic Method)** |
| a | 0.1 | 000 |
| b | 0.4 | 001 |
| c | 0.2  **No of bits required** | 010 |
| d | 0.2 | 100 |
| e | 0.1 | 101 |

=**3**

Now let’s find out the codes using Huffman method:

**Method 1:**

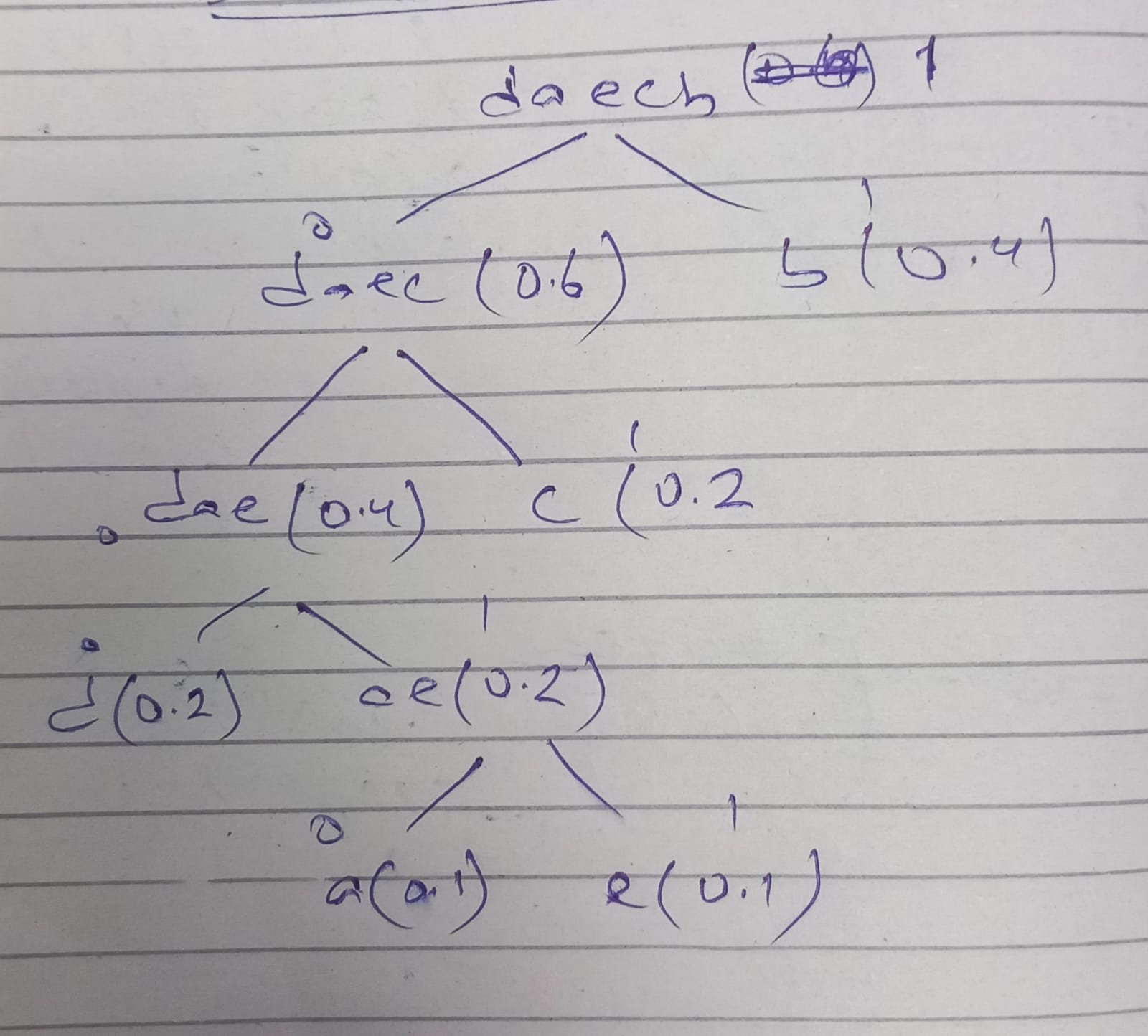
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **a**(0.1) | **b**(0.4) | **b**(0.4) | **b**(0.4) | **b**(0.4) | **b**(0.4) | **daec**(0.6) |
| **.b**(0.4) | **c**(0.2) | **c**(0.2) | **c(**0.2) | **dae**(0.4) | **daec**(0.6) | **b**(0.4) |
| **c**(0.2)  **Descending Sorting** | **d**(0.2) | **d**(0.2) | **dae**(0.4) | **c**(0.2) |  | **=daecb** |
| **d**(0.2) | **a**(0.1) | **ae**(0.2) |  |  |  |  |
| **e**(0.1) | **e**(0.1) |  |  |  |  |  |

**Huffman table:**

|  |  |
| --- | --- |
| **a** | 0010 |
| **b** | 1 |
| **c** | 01 |
| **d** | 000 |
| **e** | 0011 |

4\*0.1 + 1\* 0.4 + 2\*0.2 + 3\*0.2 + 4\*0.1 = **2.2**

**Tree:**

****