**UIT 2402 – ADVANCED DATA STRUCTURES AND ALGORITHM ANALYSIS**

**EX 5: Huffman Coding**

**Name:** B.VASUNDHARA

**Roll no:** 3122 21 5002 119

**ALGORITHM:**

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

**Program Code:**

# A Huffman Tree Node

import heapq

class node:

    def \_\_init\_\_(self, freq, symbol, left=None, right=None):

        # frequency of symbol

        self.freq = freq

        # symbol name (character)

        self.symbol = symbol

        # node left of current node

        self.left = left

        # node right of current node

        self.right = right

        # tree direction (0/1)

        self.huff = ''

    def \_\_lt\_\_(self, nxt):

        return self.freq < nxt.freq

# utility function to print huffman

# codes for all symbols in the newly

# created Huffman tree

def printNodes(node, val=''):

    # huffman code for current node

    newVal = val + str(node.huff)

    # if node is not an edge node

    # then traverse inside it

    if(node.left):

        printNodes(node.left, newVal)

    if(node.right):

        printNodes(node.right, newVal)

        # if node is edge node then

        # display its huffman code

    if(not node.left and not node.right):

        print(f"{node.symbol} -> {newVal}")

# characters for huffman tree

chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters

freq = [5, 9, 12, 13, 16, 45]

# list containing unused nodes

nodes = []

# converting characters and frequencies

# into huffman tree nodes

for x in range(len(chars)):

    heapq.heappush(nodes, node(freq[x], chars[x]))

while len(nodes) > 1:

    # sort all the nodes in ascending order

    # based on their frequency

    left = heapq.heappop(nodes)

    right = heapq.heappop(nodes)

    # assign directional value to these nodes

    left.huff = 0

    right.huff = 1

    # combine the 2 smallest nodes to create

    # new node as their parent

    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

    heapq.heappush(nodes, newNode)

# Huffman Tree is ready!

printNodes(nodes[0])

**OUTPUT:**

A screenshot of a computer

Description automatically generated