K. K. Wagh Institute of Engineering Education and Research, Nashik.
Department of Computer Engineering
Academic Year 2022-23

Name: Ahire Kalpesh Bapurao	**Class:** BE
Roll No.: 12	**Div:** A
Course: Laboratory Practice III	**Course Code:** 410246

Assignment No: 02

Problem Statement:
Write a program to implement Huffman Encoding using a greedy strategy.

Objective:
To implement Huffam Encoding as well as comprehend greedy strategy.

Course Outcome:
CO5: Implement an algorithm that follows one of the following algorithm design strategies: divide and conquer, greedy, dynamic programming, backtracking, branch and bound.

Overview of Huffman Encoding series:

- Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.
- The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one characteris not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.
- There are mainly two major parts in Huffman Coding
    1. Build a Huffman Tree from input characters.
    2. Traverse the Huffman Tree and assign codes to characters.

Time complexity:

- O(nlogn) where n is the number of unique characters. If there are n nodes, extractMin() is called 2*(n – 1) times. extractMin() takes O(logn) time as it called minHeapify(). So, overall complexity is O(nlogn).
- If the input array is sorted, there exists a linear time algorithm. We will soon be discussing in our next post.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

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 nodeis the root node and the tree is complete.

Program:

```
"""
Created on Sat Oct 22 13:46:40 2022

@author: Bardiya
Name: Asmeeta Lalit Bardiya
Class:BE-A- Computer
Lab Assignment No: 2
Title: Write a program to implement Huffman Encoding using a greedy strategy.
"""

# 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 = []

# frequency of characters
freq = []

# list containing unused nodes
nodes = []

n=int(input("Enter number of chars you want"))

for i in range(n):
   print("Enter character and frequency")
   c=input()
   f=int(input())
   chars.append(c)
   freq.append(f)
# 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)
printNodes(nodes[0])
```

Output:

```
Enter number of chars you want6
Enter character and frequency
a
5
Enter character and frequency
b
9
Enter character and frequency
c
12
Enter character and frequency
d
13
Enter character and frequency
e
16
Enter character and frequency
f
45
f -> 0
c -> 100
d -> 101
a -> 1100
b -> 1101
e -> 111
```

Conclusion:

Thus, I learnt the concept of greedy strategy as well as implemented Huffman encoding using this strategy.