INSTITUTE OF COMPUTER TECHNOLOGY

B-TECH COMPUTER SCIENCE ENGINEERING 2025-26

SUBJECT:-Algorithm Analysis and Design

NAME: Rahul Prajapati

ENRLL. NO: 23162171020

BRANCH: CYBER SECURITY

BATCH: 52

PRACTICAL_9

Aim:

Huffman coding assigns variable length code words to fixed length input characters based on their frequencies. More frequent characters are assigned shorter code words and less frequent characters are assigned longer code words. All edges along the path to a character contain a code digit. If they are on the left side of the tree, they will be a 0 (zero). If on the right, they'll be a 1 (one). Only the leaves will contain a letter and its frequency count. All other nodes will contain a null instead of a character, and the count of the frequency of all of it and its descendant characters.

Construct the Huffman tree for the following data and obtain its Huffman code.

Characters	A	B	C	D	E	-
Frequency/ Probability	0.5	0.35	0.5	0.1	0.4	0.2

(i) Encode text CAD-BE using the above code.

Input: CAD-BE

Output: 10011100110111100

(ii) Decode the text 1100110110 using the above information.

Input: 0011011100011100

Output: E-DAD

CODE:

Practical9_1.py > ...

```python
import heapq

def build_huffman_tree(chars, freqs):
    heap = []
    for c, f in zip(chars, freqs):
        heapq.heappush(heap, (f, c, None, None))
    while len(heap) > 1:
        f1, c1, l1, r1 = heapq.heappop(heap)
        f2, c2, l2, r2 = heapq.heappop(heap)
        new_node = (f1 + f2, None, (f1, c1, l1, r1), (f2, c2, l2, r2))
        heapq.heappush(heap, new_node)
    return heap[0]

def generate_codes(node, code="", codes=None):
    if codes is None:
        codes = {}
    freq, char, left, right = node
    if char is not None:
        codes[char] = code
        return codes
    generate_codes(left, code + "0", codes)
    generate_codes(right, code + "1", codes)
    return codes

def encode_text(text, codes):
    return "".join(codes[ch] for ch in text)

def decode_text(encoded, root):
    decoded = ""
    node = root
    for bit in encoded:
        freq, char, left, right = node
        node = left if bit == "0" else right
        f, c, l, r = node
        if c is not None:
            decoded += c
            node = root
    return decoded
```

```python
characters = ['A', 'B', 'C', 'D', 'E', '-']
frequencies = [0.5, 0.35, 0.5, 0.1, 0.4, 0.2]

root = build_huffman_tree(characters, frequencies)
codes = generate_codes(root)
print("Huffman Codes:", codes)

text = "CAD-BE"
encoded = encode_text(text, codes)
print("Encoded:", encoded)

decoded = decode_text(encoded, root)
print("Decoded:", decoded)
```

OUTPUT:

```
                                                    python .\Practical9_1.py
Huffman Codes: {'E': '00', 'A': '01', 'C': '10', 'D': '1100', '-': '1101', 'B': '111'}
Encoded: 10011100110111100
Decoded: CAD-BE
PS C:\Users\Hp\OneDrive\Desktop\SEM_05\Algorithm Analysis & Design\SOURCE_CODES>
```