**Assignment 2**

import heapq

from collections import defaultdict

class HuffmanNode:

def \_\_init\_\_(self, char, freq):

self.char = char

self.freq = freq

self.left = None

self.right = None

# Implementing comparison operators for heapq

def \_\_lt\_\_(self, other):

return self.freq < other.freq

def \_\_eq\_\_(self, other):

return self.freq == other.freq

def build\_huffman\_tree(freq\_table):

priority\_queue = [HuffmanNode(char, freq) for char, freq in freq\_table.items()]

heapq.heapify(priority\_queue)

while len(priority\_queue) > 1:

left\_node = heapq.heappop(priority\_queue)

right\_node = heapq.heappop(priority\_queue)

merged\_freq = left\_node.freq + right\_node.freq

merged\_node = HuffmanNode(None, merged\_freq)

merged\_node.left = left\_node

merged\_node.right = right\_node

heapq.heappush(priority\_queue, merged\_node)

return priority\_queue[0]

def build\_huffman\_codes(node, code='', code\_dict={}):

if node is None:

return

if node.char is not None:

code\_dict[node.char] = code

build\_huffman\_codes(node.left, code + '0', code\_dict)

build\_huffman\_codes(node.right, code + '1', code\_dict)

def encode\_text(text, code\_dict):

encoded\_text = ''.join(code\_dict[char] for char in text)

return encoded\_text

def main():

text = input("Enter the text to be encoded: ")

freq\_table = defaultdict(int)

for char in text:

freq\_table[char] += 1

huffman\_tree = build\_huffman\_tree(freq\_table)

huffman\_codes = {}

build\_huffman\_codes(huffman\_tree, '', huffman\_codes)

encoded\_text = encode\_text(text, huffman\_codes)

print("Original text:", text)

print("Encoded text:", encoded\_text)

print("Huffman codes:", huffman\_codes)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

Enter the text to be encoded: hello world

Original text: hello world

Encoded text: 11100001010110111101111001010001

Huffman codes: {'e': '000', 'd': '001', 'r': '010', 'w': '011', 'l': '10', 'o': '110', 'h': '1110', ' ': '1111'}