**Code:**  [Huffman Coding - Colaboratory (google.com)](https://colab.research.google.com/drive/1l2eD90PGb8J-KWzIpaxpDypCydqIbcaM#scrollTo=uza9innrRSyh)

**import heapq**

**import os**

**huffman\_codes = {}**

**class node:**

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

**self.freq = freq**

**self.symbol = symbol**

**self.left = left**

**self.right = right**

**self.huff = ''**

**def \_\_lt\_\_(self, nxt):**

**return self.freq < nxt.freq**

**def printNodes(node, val=''):**

**newVal = val + str(node.huff)**

**if node.left:**

**printNodes(node.left, newVal)**

**if node.right:**

**printNodes(node.right, newVal)**

**if not node.left and not node.right:**

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

**huffman\_codes[node.symbol] = newVal**

**file = open('test.jpeg', 'rb') # Using 'test.jpg' as the input file**

**bit\_string = ""**

**byte = file.read(1)**

**while len(byte) > 0:**

**byte = ord(byte)**

**bits = bin(byte)[2:].rjust(8, '0')**

**bit\_string += bits**

**byte = file.read(1)**

**file.close()**

**frequencies = {}**

**for i in range(0, len(bit\_string), 8):**

**if bit\_string[i:i + 8] in frequencies:**

**frequencies[bit\_string[i:i + 8]] += 1**

**else:**

**frequencies[bit\_string[i:i + 8]] = 1**

**sorted\_frequencies = sorted(frequencies.items(), key=lambda x: x[1])**

**print(sorted\_frequencies)**

**nodes = []**

**for x in range(len(sorted\_frequencies)):**

**heapq.heappush(nodes, node(sorted\_frequencies[x][1], sorted\_frequencies[x][0]))**

**while len(nodes) > 1:**

**left = heapq.heappop(nodes)**

**right = heapq.heappop(nodes)**

**left.huff = 0**

**right.huff = 1**

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

**heapq.heappush(nodes, newNode)**

**printNodes(nodes[0])**

**bit\_string\_out = ""**

**for i in range(0, len(bit\_string), 8):**

**bit\_string\_out += huffman\_codes[bit\_string[i:i + 8]]**

**output = bytearray()**

**for i in range(0, len(bit\_string\_out), 8):**

**output.append(int(bit\_string\_out[i:i + 8], 2))**

**file = open('test\_compressed.bin', 'wb') # Using 'test\_compressed.bin' for compressed file**

**file.write(output)**

**file.close()**

**print("CR: ", len(bit\_string) / len(bit\_string\_out))**

**# Decompress**

**file = open('test\_compressed.bin', 'rb')**

**bit\_string = ""**

**byte = file.read(1)**

**while len(byte) > 0:**

**byte = ord(byte)**

**bits = bin(byte)[2:].rjust(8, '0')**

**bit\_string += bits**

**byte = file.read(1)**

**file.close()**

**bit\_string\_out = ""**

**i = 0**

**pi = -1**

**while i < len(bit\_string):**

**for key in huffman\_codes:**

**if bit\_string[i:i + len(huffman\_codes[key])] == huffman\_codes[key]:**

**bit\_string\_out += key**

**i += len(huffman\_codes[key])**

**break**

**if pi == i:**

**break**

**pi = i**

**output = bytearray()**

**for i in range(0, len(bit\_string\_out), 8):**

**output.append(int(bit\_string\_out[i:i + 8], 2))**

**file = open('test\_decompressed.jpg', 'wb') # Using 'test\_decompressed.jpg' for decompressed file**

**file.write(output)**

**file.close()**

**# Get file sizes**

**original\_size = os.path.getsize('test.jpg')**

**compressed\_size = os.path.getsize('test\_compressed.bin')**

**decompressed\_size = os.path.getsize('test\_decompressed.jpg')**

**print("Original Image File Size:", original\_size, "bytes")**

**print("Compressed File Size:", compressed\_size, "bytes")**

**print("Decompressed File Size:", decompressed\_size,** "bytes")