

## UNIVERSITY OF MUMBAI

# DEPARTMENT OF COMPUTER SCIENCE

PROJECT REPORT ON

Image Compression with Huffman Encoding

 SUBMMITTED BY

**Mr. Shekade Dnyaneshwar Vitthal**

UNDER THE GUIDANCE OF

**Ms. Anuja Gharpure**

TO

UNIVERSITY OF MUMBAI

IN PARTIAL FULLFILMENT OF

F.Y.M.sc.(COMPUTER SCIENCE)

SEM-1st

ACADEMIC YEAR 2021-2022

## UNIVERSITY OF MUMBAI

# DEPARTMENT OF COMPUTER SCIENCE

# Vidya Nagari, Kalina, Santacruz East, Mumbai, Maharashtra 400098



**CERTIFICATE**

This is to certify that “**Mr. Shekade Dnyaneshwar Vitthal**” Examination Roll No-**69**\_Student of DEPARTMENT OF COMPUTER SCIENCE,UNIVERSITY OF MUMBAI   has successfully completed the project work entitled **“Image Compression with Huffman Encoding”** during the academic year 2020-2021. In partial fulfilment of the requirement for the award of Degree in Masters of Computer Science under University of Mumbai.

It is further certified that student completed all required phases of project.

**PLACE**: Mumbai

**DATE**: \_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_                                                                             \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PROJECT GUIDE                                                                   Head of DEPARTMENT**

(Ms. Anuja Gharpure)                                                         (Ms. Jyotsna Dongardive )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INTERNAL EXAMINAR**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Topics |  |  |
| 1 | Acknowledgement |  |  |
| 2 | introduction |  |  |
| 3 | flow chart |  |  |
| 4 | working |  |  |
| 5 | output |  |  |
| 6 | conclusion |  |  |
| 7 | reference |  |  |

**ACKNOWLEDGEMENT**

A project is always a result of the amalgamation of various ideas and support from countless people. I would like to acknowledge the contribution made by them. I am glad to represent my project on **“**Image Compression with Huffman Encoding**”** This project has been a learning and challenging process for me.

I express my sincere thanks to Ms. Anuja Gharpurealso express my thanks to **Classmates and** staff of the Computer Science Section/Department. I would like to thank my Sister for her encouragement and guidance, which helped me in completing the project.

I immensely grateful to all those because without their inspiration, constant prompting and useful suggestions.

I sincerely appreciate the help provided by those in the careful preparation of the project.

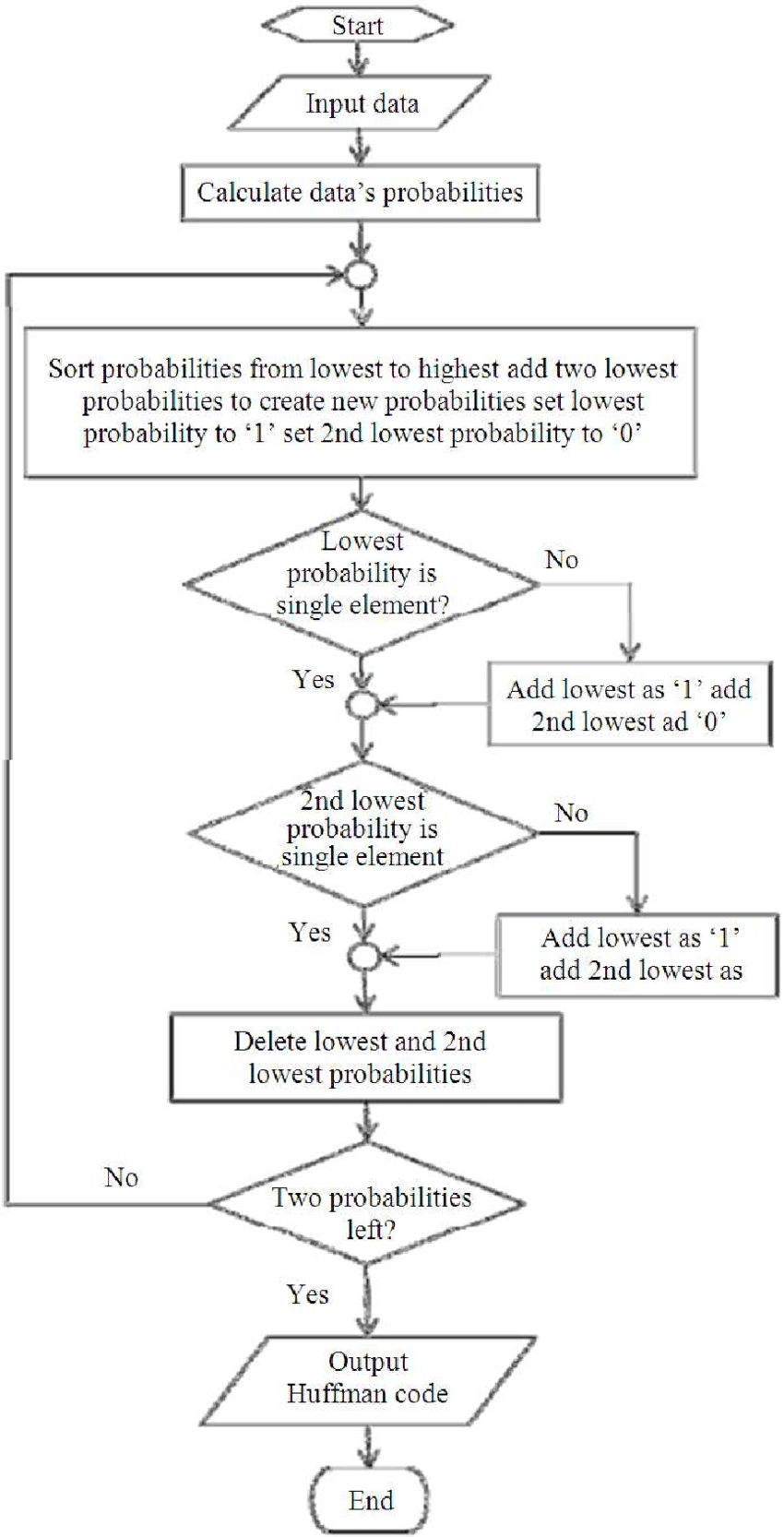
Sincere thanks from,

Mr. Shekade Dnyaneshwar Vitthal.

**INTRODUCTION**

Many image processing and computer vision algorithms are applicable to large-scale data tasks. It is often desirable to run these algorithms on large data sets (e.g. larger than 1 TB) that are currently limited by the computational power of one computer. These tasks are typically performed on a distributed system by dividing the task across one or more of the following features: algorithm parameters, images, or pixels [White et al. 2010]. Performing tasks across a particular parameter is incredibly parallel and can often be perfectly parallel. Face detection and landmark classification are examples of such algorithms [Li and Crandall 2009; Liu et al. 2009]. The ability to parallelize such tasks allows for scalable, efficient execution of resource-intensive applications. The Map Reduce framework provides a platform for such applications. Multichannel signal processing has been the subject of extensive research during the last ten years, primarily due to its importance to colour image processing. The amount of research published to date indicates a great interest in the areas of colour image filtering and analysis. It is extremely adopted that colour giving information regarding that objects and further that information is utilize in increasing the performance of the image system. Colour images are studied in this paper using a vector approach. Three-channel vector is used to represent the value at each image pixel, by giving the colourful image that any vector field where all the dimensions of the vector are associate with pixel’s chromatic properties. Being a two-dimensional (2-D), three channel signal, a colour image needs more computation and storage area, in comparison with grey image, during processing. In particular, the most common processing tasks are noise filtering and enhancement, since these are essential functions of any image processing system, regardless of whether the processed image is utilized for visual interpretation or automatic analysis.

**Flow Chart**



**Working**

**Applications of Huffman Encoding**

used broadly to encode music, images, and certain communication protocols. Typically, a variation of the algorithm is used for improved efficiency. The method described is generally part of general compression algorithms such as Flat-ZIP for images or FLAC for music.

**Understanding Prefix Codes**

* Prefix are bit sequences composed of Os’ and 1s
* To compose an encoded bitstream there is a chance that each bit could use the same sequence leading to ambiguity
* Huffman codes / Prefix codes prevent this from happening by assigning a unique code to each character

**Constructing the 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.

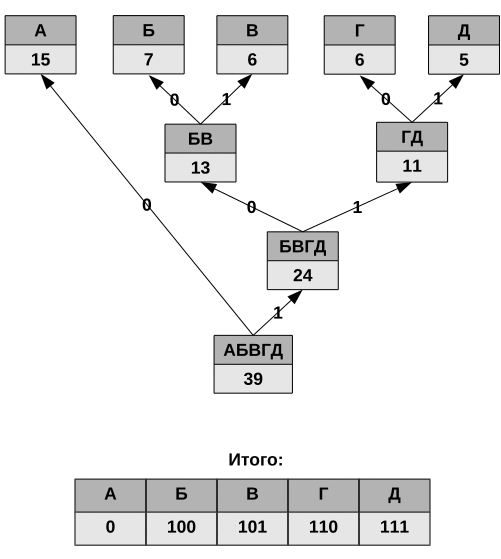
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

4. Repeat steps 2 and 3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

**Visualisation**

Each character has individual frequency

Left is 0 and right is 1



**The problem with Image Compression**

* Harder than text encoding
* Each format has various ways of storing the binary pixel data Tedious to extract pixel data from raw binary format

**Code**

from PIL import Image

# A Huffman Tree Node

class node:

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

        # frequency of symbol

        self.freq = freq

        # symbol name (charecter)

        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 = ''

class pixel\_node:

    #Define node construction method

    def \_\_init\_\_(self,right=None,left=None, parent=None, weight=0, code=None):

        self.left = left

        self.right = right

        self.parent = parent

        self.weight = weight #weight

        self.code = code #Node value

# 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}")

# charecters for huffman tree

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

# frequency of charecters

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

# list containing unused nodes

nodes = []

# converting ccharecters and frequencies

# into huffman tree nodes

for x in range(len(chars)):

    nodes.append(node(freq[x], chars[x]))

while len(nodes) > 1:

    # sort all the nodes in ascending order

    # based on theri frequency

    nodes = sorted(nodes, key=lambda x: x.freq)

    # pick 2 smallest nodes

    left = nodes[0]

    right = nodes[1]

    # 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)

    # remove the 2 nodes and add their

    # parent as new node among others

    nodes.remove(left)

    nodes.remove(right)

    nodes.append(newNode)

printNodes(nodes[0])

def pixel\_frequency(pxl\_lst):

    pxl\_freq = {}

    for i in pxl\_lst:

        if i not in pxl\_freq.keys():

            pxl\_freq[i] = 1

        else:

            pxl\_freq[i] += 1

    return pxl\_freq

def construct\_node(pixel):

    node\_lst = []

    for i in range(len(pixel)):

        node\_lst.append(pixel\_node(weight = pixel[i][1], code=str(pixel[i][0])))

    return node\_lst

def construct\_tree(node\_lst):

    node\_lst = sorted(node\_lst ,key=lambda pixel\_node:pixel\_node.weight)

    while(len(node\_lst) != 1):

        #0 is left, 1 is right

        node0, node1 = node\_lst[0], node\_lst[1]

        merge\_node = pixel\_node(left=node0, right=node1, weight=node0.weight + node1.weight)

        node0.parent = merge\_node

        node1.parent = merge\_node

        node\_lst.remove(node0)

        node\_lst.remove(node1)

        node\_lst.append(merge\_node)

        node\_lst = sorted(node\_lst ,key=lambda pixel\_node:pixel\_node.weight)

    return node\_lst

def huffman\_encoding(img):

    width = img.size[0]

    height = img.size[1]

    im = img.load()

    pixel\_lst = []

    for i in range(width):

        for j in range(height):

            pixel\_lst.append(im[i, j])

    pixel\_freq = pixel\_frequency(pixel\_lst)

    pixel\_freq = sorted(pixel\_freq.items(), key=lambda item:item[1])

    node\_lst = construct\_node(pixel\_freq)

    huff\_tree\_head = construct\_tree(node\_lst)[0]

    encoding\_table = {}

    for x in node\_lst:

        curr\_node = x

        encoding\_table.setdefault(x.code, "")

        while(curr\_node != huff\_tree\_head):

            if curr\_node.parent.left == curr\_node:

                encoding\_table[x.code] = "1" + encoding\_table[x.code]

            else:

                encoding\_table[x.code] = "0" + encoding\_table[x.code]

            curr\_node = curr\_node.parent

    for key in encoding\_table.keys():

        print("Source Pixel: " + key + "\nCode Strength after encoding:" + encoding\_table[key])

    print("Encoding Table: ", encoding\_table)

def decoding(w, h, encoding\_table, coding\_res):

    code\_read\_now = ''  # The currently read code

    new\_pixel =[]

    i = 0

    while (i != coding\_res.\_\_len\_\_()):

                 #Read one later each time

        code\_read\_now = code\_read\_now + coding\_res[i]

        for key in encoding\_table.keys():

                         #If the currently read code exists in the code table

            if code\_read\_now == encoding\_table[key]:

                new\_pixel. append(key)

                code\_read\_now = ' '

                break

        i +=1

         #Construct a new image

    decode\_image = Image.new( 'L' ,(w,h))

    k = 0

         #

    for i in range(w):

        for j in range(h):

            decode\_image.putpixel((i,j),(int(new\_pixel[k])))

        k+=1

    decode\_image.save('decode.bmp')

    print("Decoding has been completed: the picture is stored as decode.bmp")

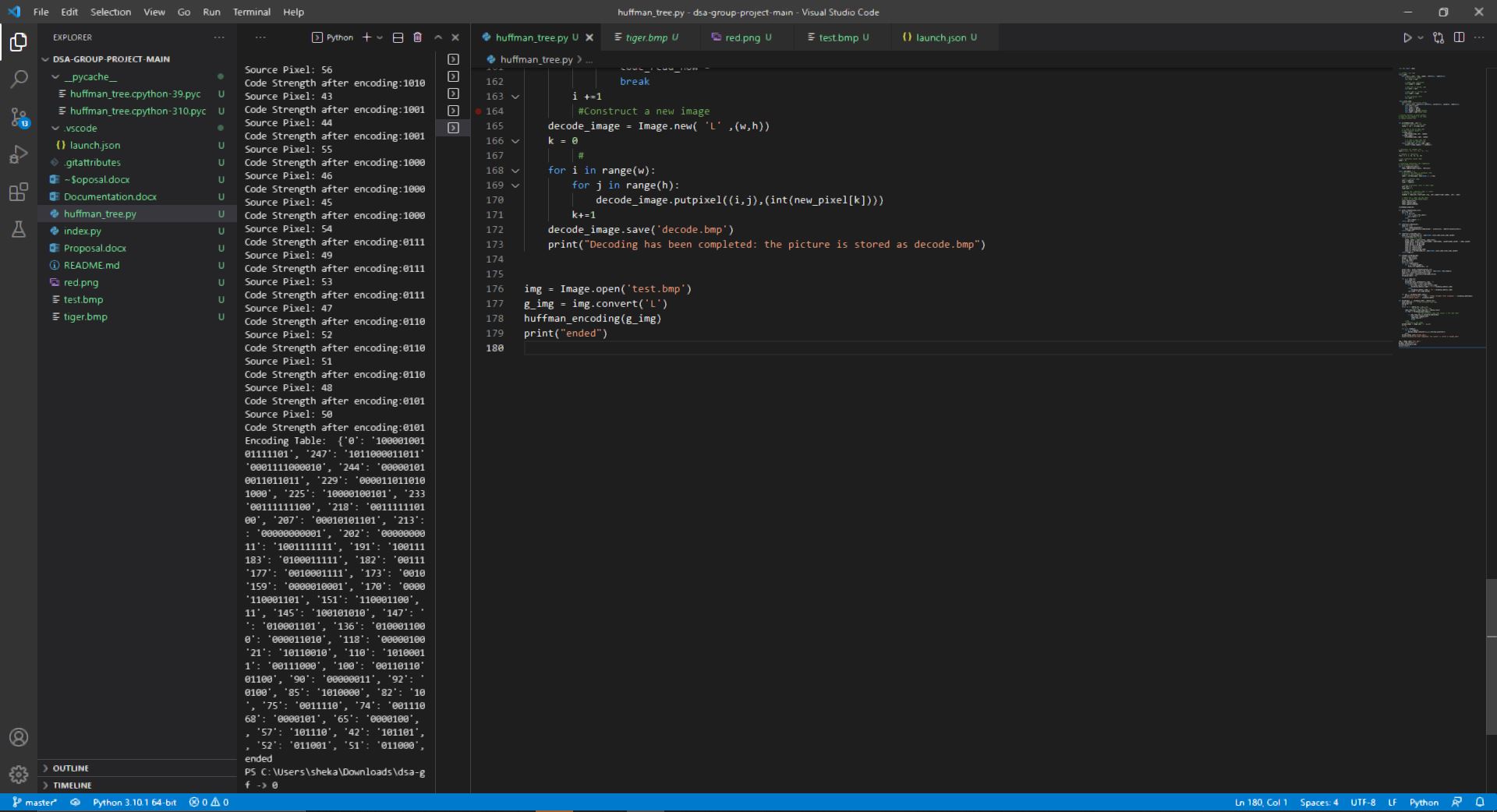
img = Image.open('tiger.bmp')

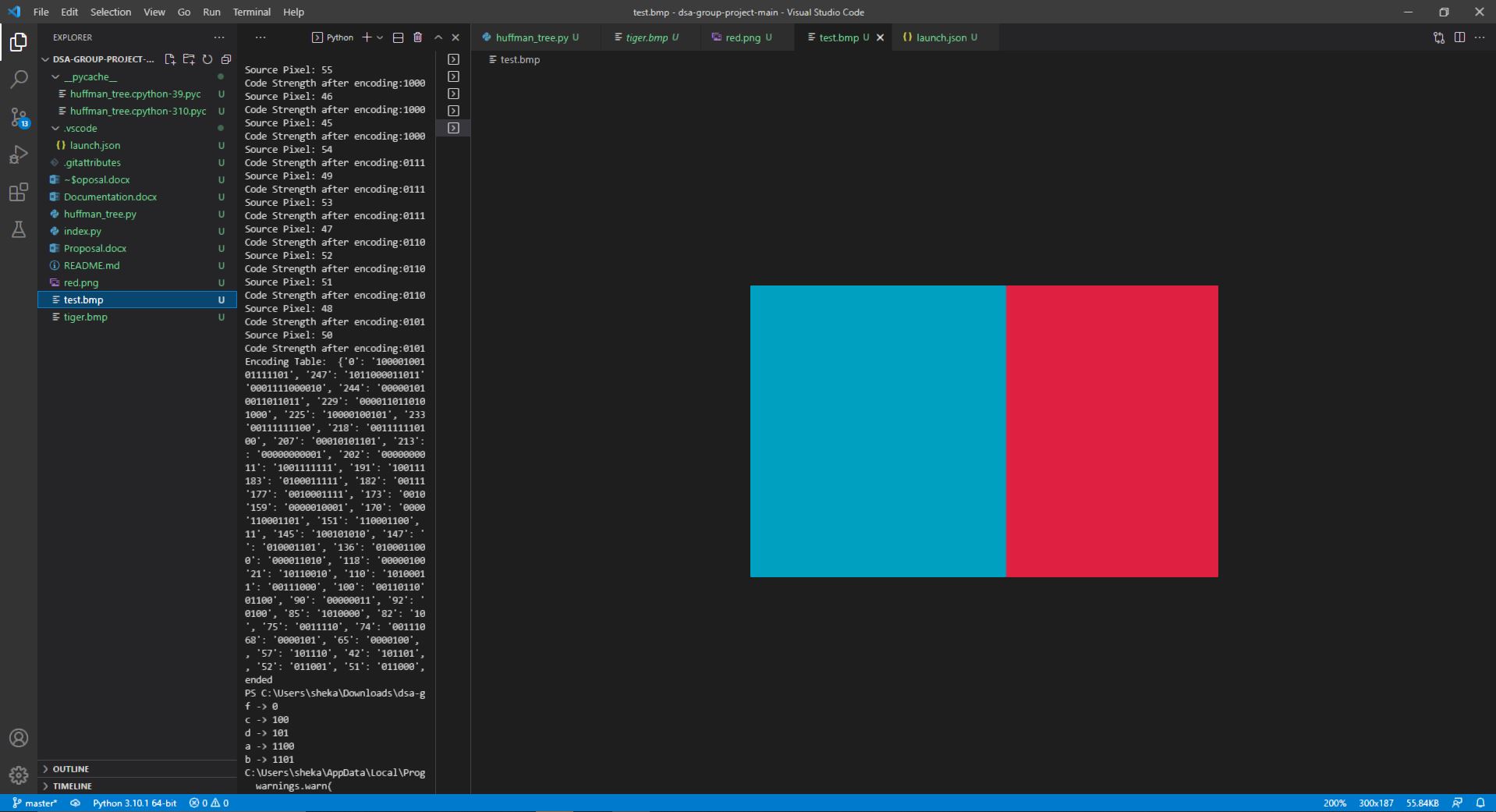
g\_img = img.convert('L')

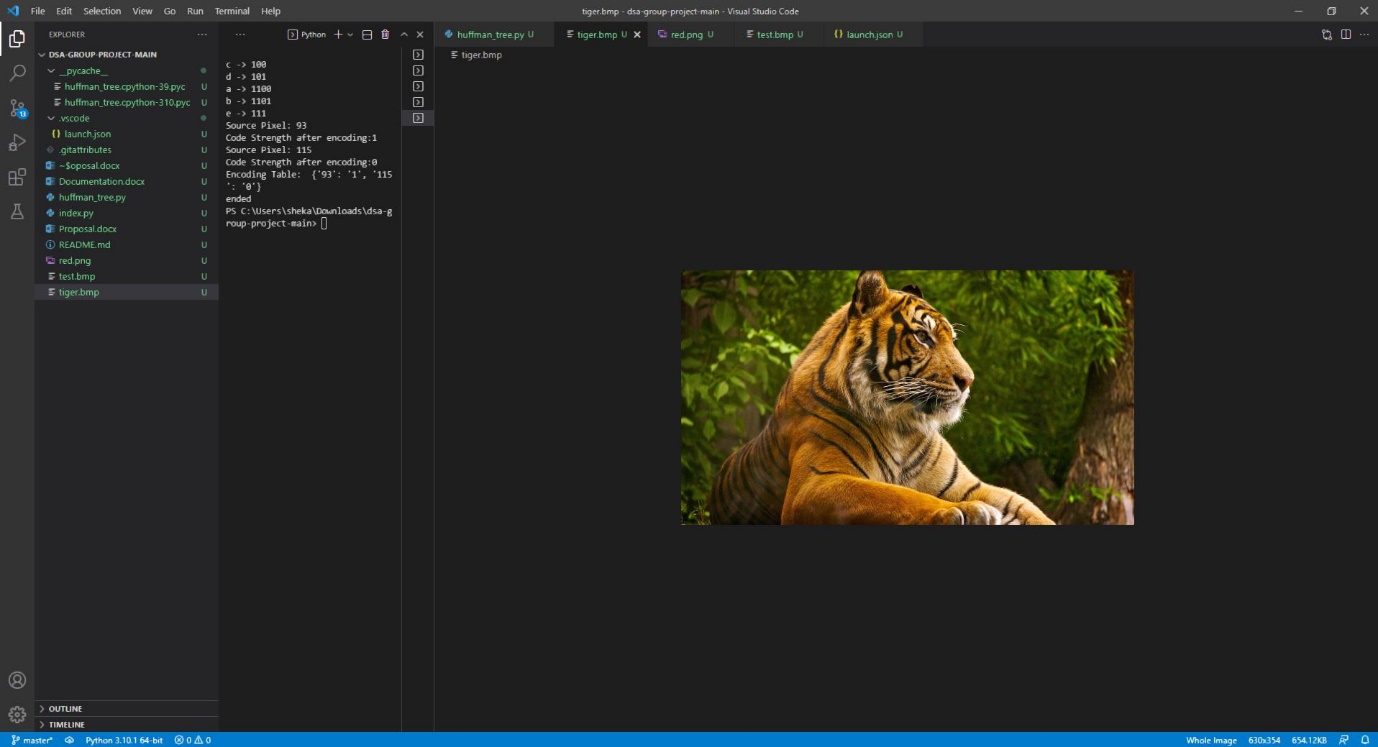
huffman\_encoding(g\_img)

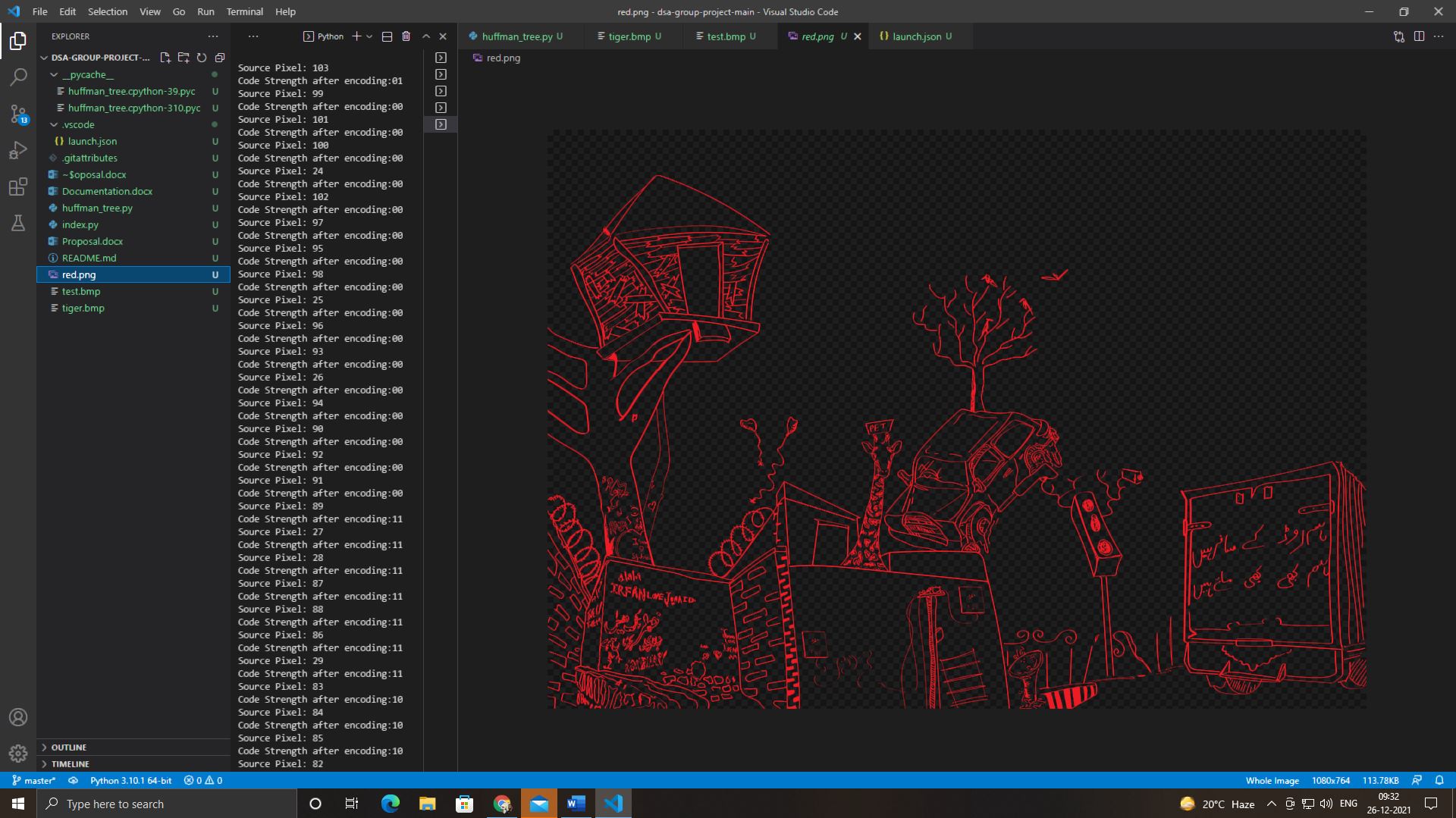
print("ended")

**Output**









**Conclusion**

The experiment shows that the higher data redundancy helps to achieve more compression. The above presented a new compression and decompression technique based on Huffman coding and decoding for scan testing to reduce test data volume, test application time. We conclude that Huffman coding is efficient technique for image compression and decompression to some extent. The compression ratio achieved by using Discrete Cosine Transform in Huffman coding can be further increased by using Discrete Fourier Transform.

**REFERENCES**

1. A.M Raid, W.M.Khedr, M. A. El-dosuky,Wesam Ahmed,"Jpeg Image Compression Using Discrete Cosine Transform -A Survey",(IJCSES) Vol.5, No.2, April 2014.
2. Jagadish h. Pujar,lohit m. Kadlaskar," A New Lossless method of image compression and decompression using huffman coding techniques",Journal of Theoretical and Applied Information Technology.
3. Mridul Kumar Mathur,Seema Loonker,Dr. Dheeraj Saxena," Lossless Huffman Coding Technique For Image Compression Andreconstruction Using Binary Tree",IJCTA,Jan-Feb 2012,Vol3(1),76-79.
4. Aarti,"Performance Analysis of Huffman Coding Algorithm",IJARCSSE, Volume3,Issue 5,May 2013.
5. <https://commons.wikimedia.org/wiki/File:Huffmantree_ru_animated.gif>
6. <https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>