ÖĞRENMEYE SINIRSIZ ÖZGÜRLÜK

COMP 482 - PROGRAMMING STUDIO
PROJECT 1

Image Compression Using Huffman Coding

Name: Dilara Şengünoğlu
ID: 420001001
Instructor: Muhittin Gökmen

Table of Contents

1. Abstract

In computer science and information theory, a Huffman code is a specific type of optimal prefix code commonly used for lossless data compression. In this project, I used the Huffman Coding method for image compression.

2. Problem Definition

In this project, I develop a program in Python to compress an image by using Huffman method, save the compressed file, and decompress the image from the compressed file.

There are 6 steps of this project:

Level 1: Huffman Encoding and Decoding

Level 2: Image Compression (Gray Level)

Level 3: Image Compression (Gray Level differences)

Level 4: Image Compression (Color)

Level 5: Image Compression (Color differences)

Level 6: GUI

In my solution, I implemented all the levels but 5.

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.

1. Code

```
"""

Author: Dilara Şengünoğlu
Date: 14 March 2022
Project: Image Compression Using Huffman Coding

"""

#Importing libraries
import streamlit as st
import re
import numpy as np
from PIL import Image
import glob


#Main method to run streamlit correctly
def main():
    selected_box = st.sidebar.selectbox('Choose one of the following',
                                        ('Gray Photo', 'RGB Colored Photo'))
    if selected_box == 'Gray Photo':
        gray_photo()
    if selected_box == 'RGB Colored Photo':
        colored_photo()

#Method for gray photo compression
def gray_photo():

    st.title('Image Compression Application for Gray Photo using Huffman
Coding')
    st.subheader('A simple app that shows gray scales image processing
algorithms.')

    st.image('image-1.jpg', use_column_width=True)

    if st.button('Click to see Original Image before huffman code works.'):

        original = Image.open('image-1.jpg')
        st.image(original, use_column_width=True)

        array = np.arange(0, 737280, 1, np.uint8)
        my_string = np.reshape(array, (1024, 720))
        print("Entered string is:", my_string)
        a = my_string
```

```
my_string = str(my_string.tolist())

letters = []
only_letters = []

for letter in my_string:
    if letter not in letters:
        frequency = my_string.count(letter)
        letters.append(frequency)
        letters.append(letter)
        only_letters.append(letter)

nodes = []
while len(letters) > 0:
    nodes.append(letters[0:2])
    letters = letters[2:]
nodes.sort()

huffman_tree = []
huffman_tree.append(nodes)

#Method to merge nodes
def merge_nodes(nodes):
    pos = 0
    newnode = []
    if len(nodes) > 1:
        nodes.sort()
        nodes[pos].append("1")
        nodes[pos+1].append("0")

        combined_node1 = (nodes[pos] [0] + nodes[pos+1] [0])
        combined_node2 = (nodes[pos] [1] + nodes[pos+1] [1])

        newnode.append(combined_node1)
        newnode.append(combined_node2)

        newnodes=[]
        newnodes.append(newnode)

        newnodes = newnodes + nodes[2:]
        nodes = newnodes
        huffman_tree.append(nodes)
        merge_nodes(nodes)

    return huffman_tree

newnodes = merge_nodes(nodes)

huffman_tree.sort(reverse=True)
```

```
        checklist = []
        for level in huffman_tree:
            for node in level:
                if node not in checklist:
                    checklist.append(node)
                else:
                    level.remove(node)
        count = 0

        letter_binary = []
        if len(only_letters) == 1:
            lettercode = [only_letters[0], "0"]
            letter_binary.append(lettercode * len(my_string))
        else:
            for letter in only_letters:
                code = ""
                for node in checklist:
                    if len(node) > 2 and letter in node[1]:
                        code = code + node[2]
                lettercode = [letter, code]
                letter_binary.append(lettercode)
        print(letter_binary)
        print("Binary code:")
        for letter in letter_binary:
            print(letter[0], letter[1])

        bitstring = ""
        for character in my_string:
            for item in letter_binary:
                if character in item:
                    bitstring = bitstring + item[1]
        binary = "0b" + bitstring
        print("Your message as binary is:")

        uncompressed_file_size = len(my_string) * 7
        compressed_file_size = len(binary) - 2

        print("Your original file size was", uncompressed_file_size, "bits. The
compressed size is:", compressed_file_size)
        st.info("Your original file size was {}
bits.".format(uncompressed_file_size))
        st.info("The compressed size is: {}".format(compressed_file_size))

        bitstring = str(binary[2:])
        uncompressed_string = ""
        code = ""

        #Decoding part
```

```
        for digit in bitstring:
            code = code + digit
            pos = 0
            for letter in letter_binary:
                if code == letter[1]:
                    uncompressed_string = uncompressed_string +
letter_binary[pos][0]
                    code = ""
                pos += 1

        temp = re.findall(r'\d+', uncompressed_string)
        res = list(map(int, temp))
        res = np.array(res)
        res = res.astype(np.uint8)
        res = np.reshape(res, (1024, 720))
        data = Image.fromarray(res)
        data.save('output_grayscaled.png')
        st.info('Program works successfully! Decoded version of the image shown
below.')
        st.image(data)
        print("Program works successfully!")

#Method for colored photo compression
def colored_photo():

   st.title('Image Compression Application for RGB Colored Image Compression
using Huffman Coding')
   st.subheader('A simple app that shows RGB Colored Image Compression')

   if st.button('See Original Image before huffman code works:'):

        st.image('image-1.jpg', use_column_width=True)
        my_string = np.asarray(Image.open('image-1.jpg'), np.uint8)
        shape = my_string.shape
        a = my_string
        print("Entered string is: {}".format(my_string))
        my_string = str(my_string.tolist())

        letters = []
        only_letters = []

        for letter in my_string:
            if letter not in letters:
                frequency = my_string.count(letter)
                letters.append(frequency)
                letters.append(letter)
                only_letters.append(letter)

        nodes = []
```

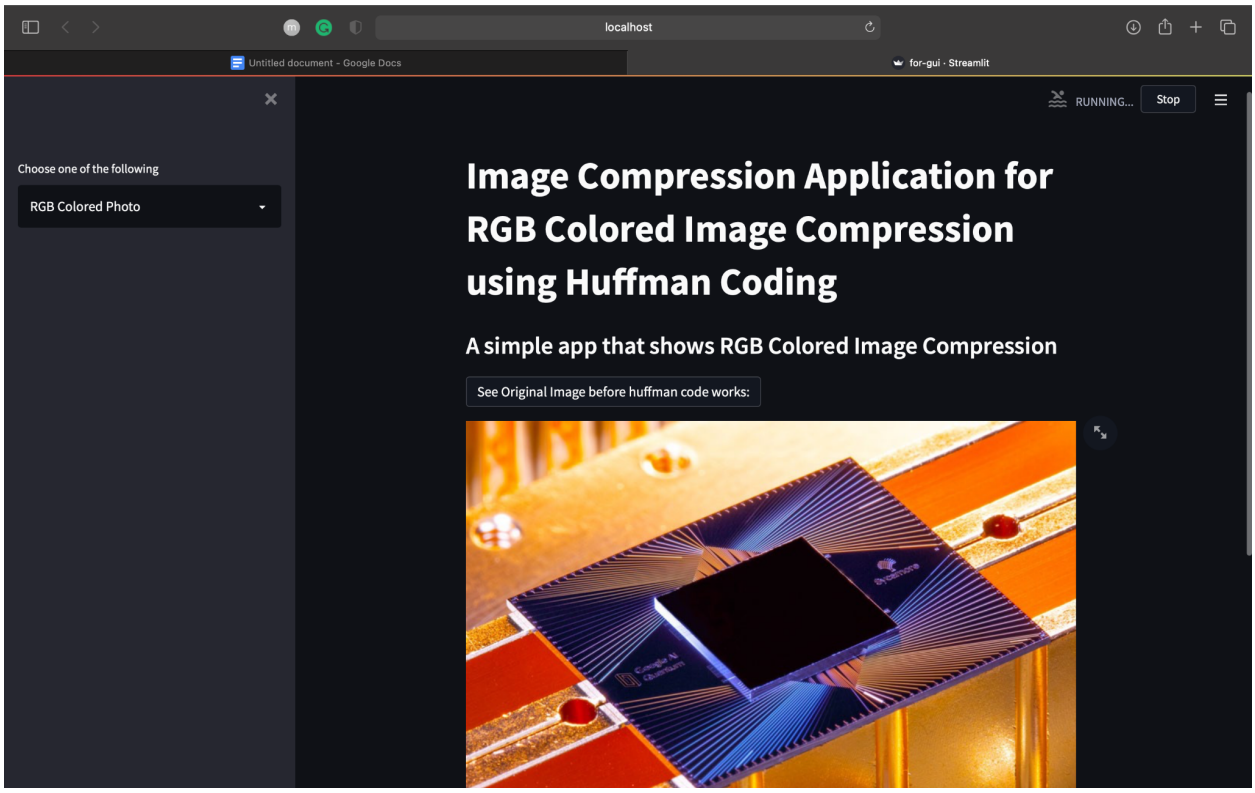
```
while len(letters) > 0:
    nodes.append(letters[0:2])
    letters = letters[2:]
nodes.sort()

huffman_tree = []
huffman_tree.append(nodes)

def merge_nodes(nodes):
    pos = 0
    newnode = []
    if len(nodes) > 1:
        nodes.sort()
        nodes[pos].append("1")
        nodes[pos + 1].append("0")

        combined_node1 = (nodes[pos][0] + nodes[pos + 1][0])
        combined_node2 = (nodes[pos][1] + nodes[pos + 1][1])

        newnode.append(combined_node1)
        newnode.append(combined_node2)

        newnodes = []
        newnodes.append(newnode)

        newnodes = newnodes + nodes[2:]
        nodes = newnodes
        huffman_tree.append(nodes)
        merge_nodes(nodes)

    return huffman_tree

newnodes = merge_nodes(nodes)

huffman_tree.sort(reverse=True)

checklist = []
for level in huffman_tree:
    for node in level:
        if node not in checklist:
            checklist.append(node)
        else:
            level.remove(node)
count = 0

letter_binary = []
if len(only_letters) == 1:
    lettercode = [only_letters[0], "0"]
    letter_binary.append(lettercode * len(my_string))
```

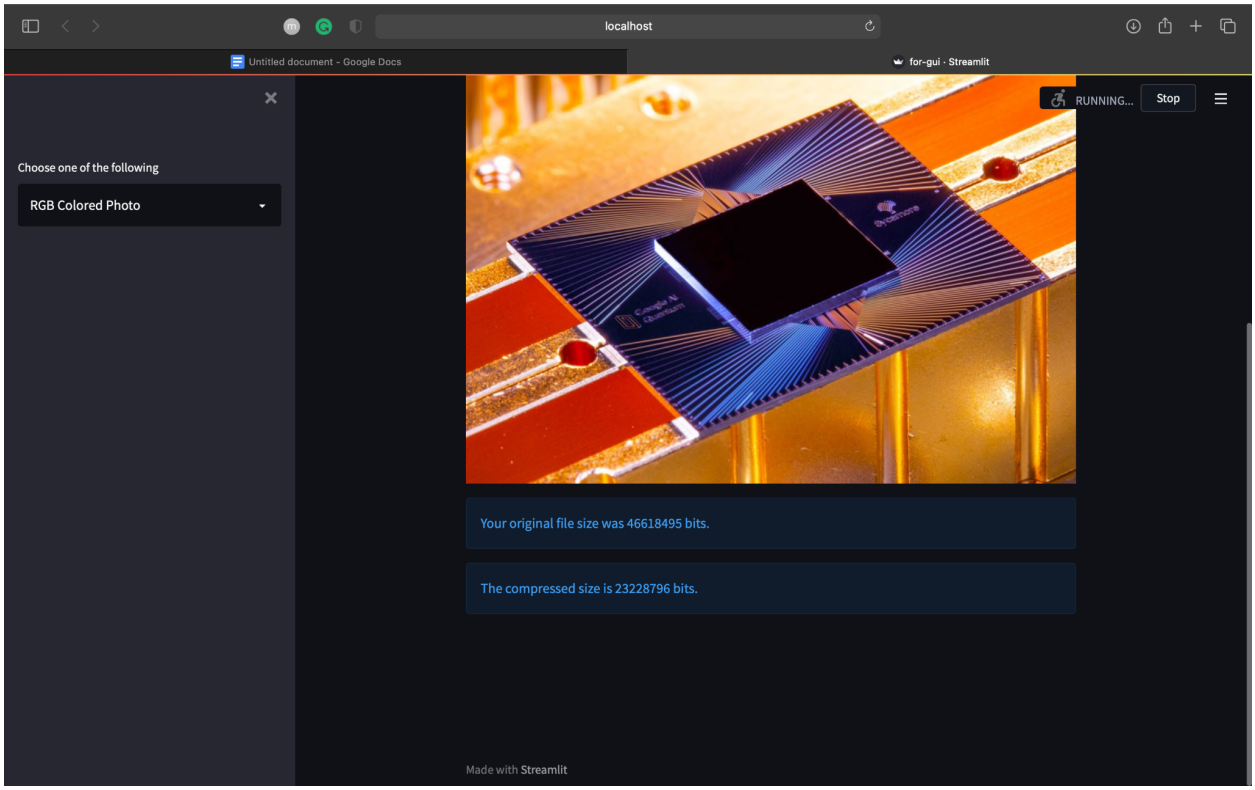
```
        else:
            for letter in only_letters:
                code = ""
                for node in checklist:
                    if len(node) > 2 and letter in node[1]:
                        code = code + node[2]
                lettercode = [letter, code]
                letter_binary.append(lettercode)
        print(letter_binary)
        print("Binary code:")
        for letter in letter_binary:
            print(letter[0], letter[1])

        bitstring = ""
        for character in my_string:
            for item in letter_binary:
                if character in item:
                    bitstring = bitstring + item[1]
        binary = "0b" + bitstring
        print("Your message as binary is:")

        uncompressed_file_size = len(my_string) * 7
        compressed_file_size = len(binary) - 2
        print("Your original file size was", uncompressed_file_size, "bits. The
compressed size is:", compressed_file_size)
        st.info("Your original file size was {}
bits.".format(uncompressed_file_size))
        st.info("The compressed size is: {}".format(compressed_file_size))

        bitstring = str(binary[2:])
        uncompressed_string = ""
        code = ""

        # Decoding
        for digit in bitstring:
            code = code + digit
            pos = 0
            for letter in letter_binary:
                if code == letter[1]:
                    uncompressed_string = uncompressed_string +
letter_binary[pos][0]
                    code = ""
                pos += 1

        temp = re.findall(r'\d+', uncompressed_string)
        res = list(map(int, temp))
        res = np.array(res)
        res = res.astype(np.uint8)
        res = np.reshape(res, shape)
```

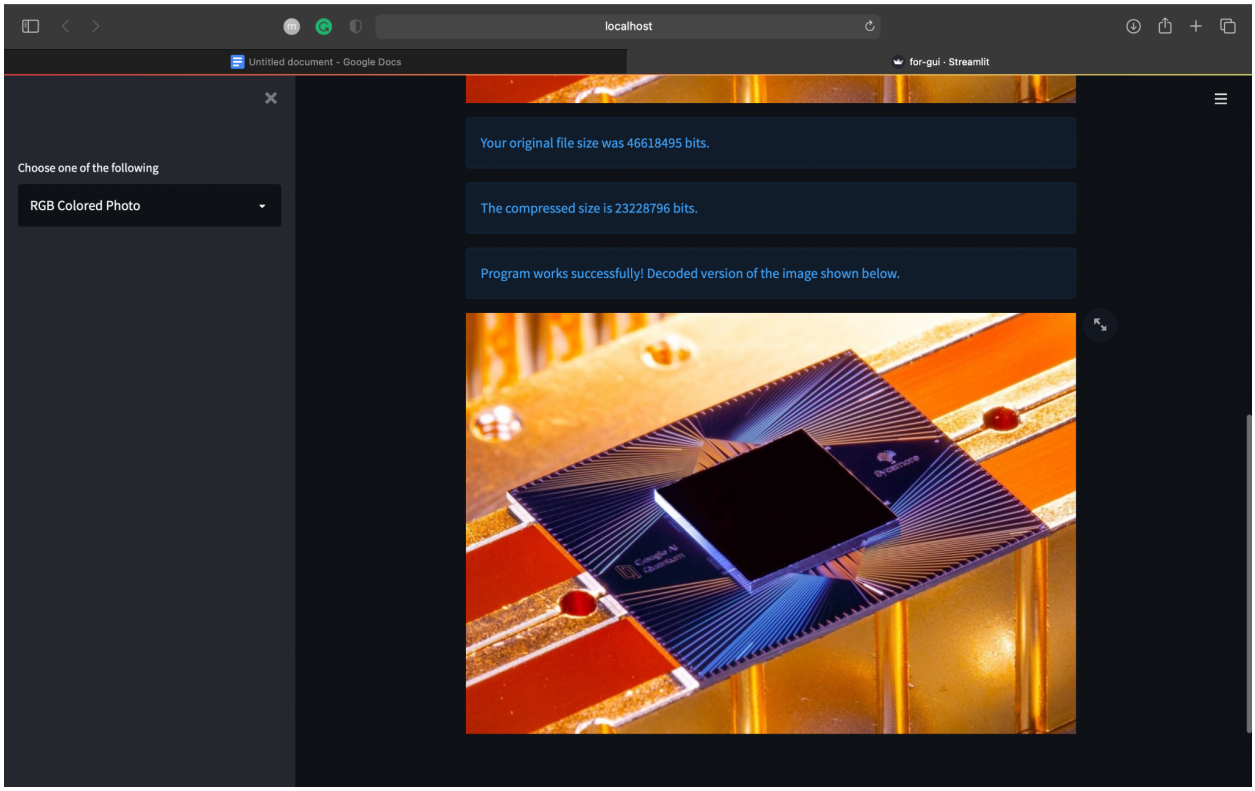
```
        print(res)
        data = Image.fromarray(res)
        data.save('output_colored.png')
        st.info('Program works successfully! Decoded version of the image shown
below.')
        st.image(data)
        if a.all() == res.all():
            print("Program works successfully!")



if __name__ == "__main__":
    main()
```

IMPORTANT: To run interface, run "**streamlit run for-gui.py**"

2. Relevant Attachments




localhost
Untitled document - Google Docs
for-gui · Streamlit
RUNNING...
Stop
Choose one of the following
RGB Colored Photo
Your original file size was 46618495 bits.
The compressed size is 23228796 bits.
Made with Streamlit


localhost
Untitled document - Google Docs
for-gui · Streamlit
Choose one of the following
RGB Colored Photo
Your original file size was 46618495 bits.
The compressed size is 23228796 bits.
Program works successfully! Decoded version of the image shown below.

3. Result

In computer science and information theory, a Huffman code is a specific type of optimal prefix code commonly used for lossless data compression. In this project, I used the Huffman Coding method for image compression. Streamlit platform provides a simple implementation and appearance to be used as an interface.