In [1]:
```
# to run this notebook, jupyter notebook is preferred

# just open this 'ipynb' file with jupyter notebook, preferably ins
talled through
# anaconda environment

# in top section, click on 'cells', then 'run all', and you are goo
d to go :)

# don't forget to put your input filename in 'encode_file' function
```

In [2]:
```
# installing the required packages

!pip install bitstring
!pip install Pillow
!pip install glob3
!pip install path.py
!pip install ffmpeg-python
```

```
Requirement already satisfied: bitstring in ./anaconda3/lib/python3.
8/site-packages (3.1.7)
Requirement already satisfied: Pillow in ./anaconda3/lib/python3.8/s
ite-packages (7.2.0)
Requirement already satisfied: glob3 in ./anaconda3/lib/python3.8/si
te-packages (0.0.1)
Requirement already satisfied: path.py in ./anaconda3/lib/python3.8/
site-packages (12.5.0)
Requirement already satisfied: path in ./anaconda3/lib/python3.8/sit
e-packages (from path.py) (13.1.0)
Requirement already satisfied: ffmpeg-python in ./anaconda3/lib/pyth
on3.8/site-packages (0.2.0)
Requirement already satisfied: future in ./anaconda3/lib/python3.8/s
ite-packages (from ffmpeg-python) (0.18.2)
```

In [3]:
```
# importing required modules

# module : to interact with the operating system
import os

# module :  for high level operations on files and collection of fi
les
# helps in automating the copying and removal of files and director
ies
import shutil

# module : gzip support for files
import gzip

# contextlib module :  provides utilities for common tasks involvin
g the 'with' statement
from contextlib import ExitStack

# bitstring module :  for the creation and analysis of binary data
from bitstring import BitArray,Bits

# module : python imaging library : support for image processing
from PIL import Image

# module : numerical python : fast and efficient processing for arr
ays
import numpy as np

# module : video processing support for python
import ffmpeg

# module: unix like pathname expansion
import glob
```

In [4]:
```
# encoding part
```

In [5]:
```
# outline
#  1. encoding_setup
#  2. compress_file
#  3. get_bitarray
#  4. generate_frames
#  5. generate_video
```

In [6]:
```
# function: to delete contents of the given folder
def rm_dir_content(dir):
    for filename in os.listdir(dir):
        file_path = os.path.join(dir, filename)
        try:
            # if it is a file or a symbolic link, delete or unlink
the file
            if os.path.isfile(file_path) or os.path.islink(file_pat
h):
                os.unlink(file_path)

            # if it is a directory, remove directory tree
            elif os.path.isdir(file_path):
                shutil.rmtree(file_path)
        except Exception as e:
            print('Failed to delete %s. Reason: %s' % (file_path,
e))


# making the setup
def encoding_setup():
    # directory to store intermediate frames to make video file
    inframes = './inframes'

    # if inframes does not exist, create it
    if not os.path.exists(inframes):
        os.makedirs(inframes)

    # delete anything present in inframes directory
    rm_dir_content(inframes)

    # delete the previous video file by same default name - OUTFIL
E.mp4
    os.remove('OUTFILE.mp4')
```

In [7]:
```
# function : compress a file using gzip compression
def compress_file(INPUT):
#   os.path.basename() : removes the leading path information of th
e file and leaves only
#   with the actual filename from the complete path
#   /home/user/file.txt -> file.txt
    print("Compressing " + os.path.basename(INPUT) + "...")

#   creates a stack of files so that we can define operations one a
fter the other
#   helpful in removing nested 'with' statements
    with ExitStack() as stack:
        f_in = stack.enter_context(open(INPUT, 'rb'))
        f_out = stack.enter_context(gzip.open(INPUT + ".gz", 'wb'))
        shutil.copyfileobj(f_in, f_out)

#       deleting unused objects
        del f_in
        del f_out

    print("Successfully compressed " + os.path.basename(INPUT))
```

In [8]:
```
# function : get binary representation of a file
def get_bitarray(INPUT):
    print("Converting " + os.path.basename(INPUT) + " to binary for
m...")

#   stores the hexdump of the file in a bitstring.BitArray object
    bitarray = BitArray(filename = INPUT)
#     print("Type: " ,type(bitarray))
#     print("Hexdump: ", bitarray)

    # removing the gzip file after deriving bitarray from it
    os.remove(INPUT + ".gz")

    print("Successfully converted " + os.path.basename(INPUT) + " t
o binary form")

#   returns the binary dump of file
#     print("Bindump: ", bitarray.bin)
    return bitarray.bin
```

In [9]:
```
# function : generate frames from binstring.BitArray object
def generate_frames(bitarray):
#   RESOLUTION = (HEIGHT, WIDTH) : resolution of the video
    RESOLUTION = (480, 854)

    print("Generating frames...")

    index = 0
    frame_num = 0
    while(index < len(bitarray)):
#       generating a numpy array with the bitarray[index : index + resolution] slice
#       with data type as int
        pixels = np.fromiter(bitarray[index : index + (RESOLUTION[0] * RESOLUTION[1])], dtype = np.int)

        if(pixels.size < (RESOLUTION[0] * RESOLUTION[1])):
            pixels = np.concatenate((pixels, np.zeros((RESOLUTION[0] * RESOLUTION[1]) - pixels.size ,dtype = int)))

#       creating a new instance of 1-bit pixel image with the specified resolution and
#       with 1 pixel per byte. tuple denotes (width, height)
        image = Image.new("1", (RESOLUTION[1], RESOLUTION[0]))

        image.putdata(pixels)
        image.save("./inframes/" + "frame_" + str(frame_num) + ".png")
#         print("Generated frame: " + str(frame_num))

        del pixels
        del image
        frame_num += 1
        index += (RESOLUTION[0] * RESOLUTION[1])

    print("Successfully generated all frames")
```

In [10]:
```
# function : generate video file from frames
def generate_video(OUTPUT, FRAMERATE):
#     FRAMERATE = 24
    print("Generating video file...")

    (
        ffmpeg
        .input('./inframes/frame_%d.png')
        .filter('fps', fps=FRAMERATE, round='up')
        .output(OUTPUT)
        .run()
    )
#     os.system('ffmpeg -framerate 24 -i ./inframes/frame_%d.png output.mp4')

    shutil.rmtree('./inframes')
    print("Successfully generated video file")
```

In [ ]:

In [ ]:

In [ ]:

In [11]:
```
# decoding part
```

In [12]:
```
# outline
#  1. decoding_setup
#  2. get_bits_from_video
#     1. convert_video_to_frames
#     2. convet_image_to_bits
#  3. get_file_from_bits
```

In [13]:
```
# making the setup
def decoding_setup():
    # directory to store intermediate frames to make video file
    outframes = './outframes'

    # first removing the directory if it contains anything
    if(os.path.isdir(outframes)):
        shutil.rmtree(outframes)

    os.makedirs(outframes)
```

In [14]:
```
def convert_video_to_frames(INPUT, FRAMERATE):
    print("Converting video file to respective frames...")

    (
        ffmpeg
        .input(INPUT)
        .filter('fps', fps=FRAMERATE, round='down')
        .output('./outframes/frame_%d.png')
        .run()
    )

    print("Successfully generated all frames")
```

In [15]:
```
def convert_image_to_bits(imagepath):
    image = Image.open(imagepath)
    width, height = image.size
    bits = ""
    pixels = image.load()
    del image

    for j in range(height):
        for i in range(width):
            pixel = pixels[i, j]
            pixel_bin_rep = "0"

#           if white difference is smaller then black difference, t
hen
#           pixel_bin_rep must be "1"
            if (abs(pixel[0] - 255) < abs(pixel[0] - 0)
            and abs(pixel[1] - 255) < abs(pixel[1] - 0)
            and abs(pixel[2] - 255) < abs(pixel[2] - 0)):
                pixel_bin_rep = "1"

            bits += str(pixel_bin_rep)
    del pixels
    return bits
```

In [16]:
```
def get_bits_from_video(videopath, FRAMERATE):
    print("Getting bits from video file...")

    convert_video_to_frames(videopath, FRAMERATE)

    bits = ""

    for image in sorted(glob.glob("./outframes/*.png")):
        bits += convert_image_to_bits(image)

    shutil.rmtree('./outframes')
    print("Successfully retrieved bits from video file")

    return bits
```

In [17]:
```
def get_file_from_bits(bits, OUTPUT):
    print("Generating file from bits...")

    bitstring = Bits(bin = bits)
    bitstring = BitArray(bitstring)


    with open(OUTPUT, 'wb') as outfile:
        bitstring.tofile(outfile)

    del bitstring

    print("Successfully retrieved the file")
```

In [ ]:

In [ ]:

In [ ]:

In [18]:
```
# actual function to encode files
def encode_file(INPUT, OUTPUT = 'OUTFILE.mp4'):
    FRAMERATE = 24

    encoding_setup()
    compress_file(INPUT)
    bitarray = get_bitarray(INPUT)
    generate_frames(bitarray)
    generate_video(OUTPUT, FRAMERATE)
```

In [19]:
```
#actual function to decode files
def decode_file(videopath, OUTPUT = 'OUTFILE.gz'):
    FRAMERATE = 24

    decoding_setup()
    bits = get_bits_from_video(videopath, FRAMERATE)
    get_file_from_bits(bits, OUTPUT)
```

In [20]:
```
encode_file('FINAL450.pdf')
```

```
Compressing FINAL450.pdf...
Successfully compressed FINAL450.pdf
Converting FINAL450.pdf to binary form...
Successfully converted FINAL450.pdf to binary form
Generating frames...
Successfully generated all frames
Generating video file...
Successfully generated video file
```

In [21]:
```
decode_file('OUTFILE.mp4', 'OUTFILE.pdf.gz')
```

```
Getting bits from video file...
Converting video file to respective frames...
Successfully generated all frames
Successfully retrieved bits from video file
Generating file from bits...
Successfully retrieved the file
```

In [ ]:

In [ ]: