Exercise 2 Report

Coursera Exercise 2 & 3 Shareable Lab Link:
https://hub.labs.coursera.org:443/connect/sharedsmgnmkdp?forceRefresh=false

All values in the table below are in **bytes.**

	Original Size	Rice (K = 4 bits)	Rice (K = 2 bits)	% Compression (K = 4 bits)	% Compression (K = 2 bits)
Sound1.wav	1,002,088	12,129,766	32,925,068	1,002,088	1,002,088
Sound2.wav	1,008,044	12,602,411	34,788,038	1,008,044	1,008,044

From the table above, we can see the file size of the bit length K = 2 bits is larger than the K = 4 bits. This is due to the fact that the smaller the bits, the larger the file size, which can be observed here. In general, a lower value of k makes smaller numbers cheaper to store and larger numbers more costly. A larger value of k, on the other hand, makes high numbers comparatively inexpensive to store while raising the storage overhead on all smaller values and making them more expensive to store. For the % compression with K = 4 and 2 bits, the file size reverts back to the original size, which means that the effectiveness of the rice coding algorithm has been effective.

We will firstly initialize the wav files. Afterwards, we will convert the sound frames to byte array.

```
In [3]: #initialise wav file
audio1 = wave.open(soundAudioFile1, mode = 'rb')
audio2 = wave.open(soundAudioFile2, mode = 'rb')
```

```
In [4]: #frames to byte array conversion
audioFrame1 = bytearray(list(audio1.readframes(audio1.getnframes())))
audioFrame2 = bytearray(list(audio2.readframes(audio2.getnframes())))
```

This will be the code for Rice Coding:

Rice Coding

```
In [5]: def rice_coding(frames, K):
    M = 2 ** K
    binary_string = ''

    for frame in frames:
        q = math.floor(frame/M)
        quotient_code = ''

        #finding the quotient, q
        for i in range(q):
            quotient_code += '1'
        quotient_code += '0'

        #finding the remainder, r
        r = frame % M
        remainder_code = bin(r)[2:].zfill(K)
        code_word = quotient_code + remainder_code
        binary_string += code_word
    return binary_string
```

We will now encode both audio files:

Encoding

Encode for Sound1

```
In [6]: binarySoundAudio1K4 = rice_coding(audioFrame1, 4)
        binarySoundAudio1K2 = rice_coding(audioFrame1, 2)
```

Encode for Sound2

```
In [7]: binarySoundAudio2K4 = rice_coding(audioFrame2, 4)
        binarySoundAudio2K2 = rice_coding(audioFrame2, 2)
```

Now let's write to Sound1 where K = 2

```
In [8]: file = open("Sound1_Enc_K2.ex2", "w")
        file.write(binarySoundAudio1K2)
        file.close()
        print('written to Sound1 where K = 2 successful')
```

```
written to Sound1 where K = 2 successful
```

Now let's write to Sound1 where K = 4

```
In [9]: file = open("Sound1_Enc_K4.ex2", "w")
        file.write(binarySoundAudio1K4)
        file.close()
        print('written to Sound1 where K = 4 successful')
```

```
written to Sound1 where K = 4 successful
```

Now let's write to Sound2 where K = 2

```
In [10]: file = open("Sound2_Enc_K2.ex2", "w")
         file.write(binarySoundAudio2K2)
         file.close()
         print('written to Sound2 where K = 2 successful')
```

```
written to Sound2 where K = 2 successful
```

Now let's write to Sound2 where K = 4

```
In [11]: file = open("Sound2_Enc_K4.ex2", "w")
         file.write(binarySoundAudio2K4)
         file.close()
         print('written to Sound2 where K = 4 successful')
```

```
written to Sound2 where K = 4 successful
```

Afterwards, we will decode both audio files:

Decoding

In [12]:
```
def acquire_quotient(frame):
    frame = str(frame)
    count = 0

    for i in frame:
        if i == '1':
            count += 1
        else:
            return count
    return count
```

In [13]:
```
def acquire_remainder(frame, K, q):
    frame = str(frame)
    frame = frame[q+1:]
    r = int(frame, 2)

    return r
```

In [14]:
```
def acquire_S(frame, K):
    M = 2 ** K
    q = acquire_quotient(frame)
    r = acquire_remainder(frame, K, q)
    S = q*M+r

    return S
```

In [15]:
```
def audio_decode(K, binaryAudioFile):

    binaryOpenAudioFile = open(binaryAudioFile, "r")
    binaryDataAudioFile = binaryOpenAudioFile.read()

    decodedFramesAudio = bytearray()
    i = 0
    string = ''
    while i < len(binaryDataAudioFile):
        if binaryDataAudioFile[i] == '1':
            string += '1'
            i+=1
        #if the first 0 is reached, add 0 and the following K numbers
        elif binaryDataAudioFile[i] == '0':
            string += (binaryDataAudioFile[i:i+(K+1)])
            i+=(K+1)
            #this will add and insert into bytearray
            decodedFramesAudio.append(acquire_S(string, K))
            string = ''
    return decodedFramesAudio
```

Now we need to see if we can recreate the original wav files:

Now we need to see whether the original wav frames could be reconstructed

Sound1 K = 2

In [18]:
```
decodedFramesSoundAudio1K2 == audioFrame1
```

Out[18]:
```
True
```

Sound1 K = 4

In [19]:
```
decodedFramesSoundAudio1K4 == audioFrame1
```

Out[19]:
```
True
```

Sound2 K = 2

In [20]:
```
decodedFramesSoundAudio2K2 == audioFrame2
```

Out[20]:
```
True
```

Sound2 K = 4

In [21]:
```
decodedFramesSoundAudio2K4 == audioFrame2
```

Out[21]:
```
True
```

Creating a new _Enc_Dec.wav file:

The final wav file is created

In [22]:
```
def create_decoded_sound_audio(K, decodedFramesAudio, resultingSoundAudioFile):

    getParamsFrom = resultingSoundAudioFile.split('_')[0] + '.wav'
    originalSoundAudioFile = wave.open(getParamsFrom, mode="rb")

    newSoundAudio = wave.open(resultingSoundAudioFile,"wb")
    newSoundAudio.setparams(originalSoundAudioFile.getparams())
    newSoundAudio.writeframes(decodedFramesAudio)

    newSoundAudio.close()
    originalSoundAudioFile.close()

    return 1
```

Sound1 K = 2

In [23]:
```
create_decoded_sound_audio(2, decodedFramesSoundAudio1K2, 'Sound1_Enc_Dec_K2.wav')
```
Out[23]: 1

Sound1 K = 4

In [24]:
```
create_decoded_sound_audio(4, decodedFramesSoundAudio1K4, 'Sound1_Enc_Dec_K4.wav')
```
Out[24]: 1

Sound2 K = 2

In [25]:
```
create_decoded_sound_audio(2, decodedFramesSoundAudio2K2, 'Sound2_Enc_Dec_K2.wav')
```
Out[25]: 1

Sound2 K = 4

In [26]:
```
create_decoded_sound_audio(4, decodedFramesSoundAudio2K4, 'Sound2_Enc_Dec_K4.wav')
```
Out[26]: 1