Web Mining Lab Assignment-4

Aryan Vigyat

20BCE1452

Question 1

Apply run length encoding for the following string and compress it

In [ ]:

```
def encode(input_str):
    n = len(input_str)
    i = 0
    result = ""
    while i < n - 1:
        count = 1
        while (i < n - 1 and
                input_str[i] == input_str[i + 1]):
            count += 1
            i += 1
        result += input_str[i] + str(count)
        i += 1
    print("The Run Length Encoding of string", input_str, "=", result)

input_string = "eeeeeeerrrrrrrrrrrrrrrrttttttttttttttiiiiiiifffffeft"
encode(input_string)
```

Question 2

i. Apply Binary coding for term “Mercury” (apply for all doc ids)

In [ ]:

```
def to_binary(decimal_num):
    binary = bin(decimal_num)
    binary = binary[2:]
    return binary

print("The Binary Encoding for Planets is ", end=" ")
planets = [1, 2, 3, 7, 9, 10]
for planet in planets:
    print(to_binary(planet), end=" ")
```

```
The Binary Encoding for Planets is  1 10 11 111 1001 1010
```

ii. Apply Unary coding for term “Fiber”

In [ ]:

```
def to_unary(numbers):
    for number in numbers:
        unary_code = []
        for i in range(number):
            unary_code.append(1)
        unary_code.append(0)
        unary_code = [str(k) for k in unary_code]
        unary_code = "".join(unary_code)
        print(unary_code, end=" ")
```

```
elements = [1, 3, 5, 7, 19, 20]
print("The Unary Coding for Elements is ", end=" ")
to_unary(elements)
```

iii. Apply Elias Gamma Encoding for term “Airtel”

In [ ]:

```
from math import log
log2 = lambda x: log(x, 2)

def unary_encode(x):
    return (x-1)*'0'+'1'

def binary_encode(x, l = 1):
    s = '{0:0%db}' % l
    return s.format(x)

def elias_gamma(x):
    if(x == 0):
        return '0'
    n = 1 + int(log2(x))
    b = x - 2**(int(log2(x)))
    l = int(log2(x))
    return unary_encode(n) + binary_encode(b, l)

companies = [12, 17, 25, 148, 156, 159, 172]
print("The Elias Gamma Encoding for Companies is ", end=" ")
for company in companies:
    print(elias_gamma(company), end=" ")
```

```
The Elias Gamma Encoding for Companies is  0001100 000010001 000011001 00000001001
0100 000000010011100 000000010011111 000000010101100
```

v. Apply Elias Delta Encoding for term “Venus”

In [ ]:

```
from math import log
from math import floor

def Binary_Representation_Without_MSB(x):
    binary = "{0:b}".format(int(x))
    binary_without_MSB = binary[1:]
    return binary_without_MSB

def EliasGammaEncode(k):
    if (k == 0):
        return '0'
    N = 1 + floor(log(k, 2))
    Unary = (N-1)*'0'+'1'
    return Unary + Binary_Representation_Without_MSB(k)

def EliasDeltaEncode(x):
    Gamma = EliasGammaEncode(1 + floor(log(x, 2)))
    binary_without_MSB = Binary_Representation_Without_MSB(x)
    return Gamma+binary_without_MSB

print("The Elias Delta Encoding for Venus is ", end=" ")
Venus = [23, 45, 78, 122, 145]
for N in Venus:
    print(EliasDeltaEncode(N), end=" ")
```

```
The Elias Delta Encoding for Venus is  001010111 0011001101 00111001110 0011111101
0 00010000010001
```

iv. Apply Elias Delta Encoding and Decoding for “Mercury”

In [ ]:

```
mercury=[1, 2, 3, 7, 9, 1]
import math
from math import log
from math import floor

def Binary_Representation_Without_MSB(x):
    binary = "{0:b}".format(int(x))
    binary_without_MSB = binary[1:]
    return binary_without_MSB

def EliasGammaEncode(k):
    if (k == 0):
        return '0'
    N = 1 + floor(log(k, 2))
    Unary = (N-1)*'0'+'1'
    return Unary + Binary_Representation_Without_MSB(k)

def EliasDeltaEncode(x):
    Gamma = EliasGammaEncode(1 + floor(log(x, 2)))
    binary_without_MSB = Binary_Representation_Without_MSB(x)
    return Gamma+binary_without_MSB

print("The Elias Delta Encoding for Mercury is ", end=" ")
listdec=list()

for N in mercury:
    listdec.append(EliasDeltaEncode(N))
    print(EliasDeltaEncode(N), end=" ")
print(" ")

def decode_elias_delta(x):
    if(x=='1'):
        return 1
    count=0
    for i in x:
        if(i=='1'):
            break
        else:
            count=count+1
    y=x[count:count+1+count]
    exp=int(y,2)-1
    rest=int(x[2*count+1:],2)
    ans=pow(2,exp)+rest
    return ans
for i in listdec:
    print("The Decoded Value  is ", decode_elias_delta(i))
```

```
The Elias Delta Encoding for Mercury is  1 0100 0101 01111 00100001 1
The Decoded Value  is  1
The Decoded Value  is  2
The Decoded Value  is  3
The Decoded Value  is  7
The Decoded Value  is  9
The Decoded Value  is  1
```

vii. Variable Byte Encoding

In [ ]:

```
def cumulative_sum_to_normal(cumulative_sum_array):
    normal_array = [cumulative_sum_array[0]]
    for i in range(1, len(cumulative_sum_array)):
        normal_array.append(cumulative_sum_array[i] - cumulative_sum_array[i-1])
    return normal_array

cumulative_sum_array = [1, 3, 9, 12]
normal_array = cumulative_sum_to_normal(cumulative_sum_array)
def vbencode(x):
    binval=list()
    while x>0:
        rem=x%128
        x=x//128
        binval.insert(0,rem)
    templist=list()
    if(len(binval)==1):
        y=bin(binval[0])
        tempans=y[2:].zfill(8)
        tempans='1'+tempans[1:]
        templist.append(tempans)
    else:
        for i in range(len(binval)-1):
            y=bin(binval[i])
            templist.append(y[2:].zfill(8))
        y=bin(binval[len(binval)-1])
        tempans=y[2:].zfill(8)
        tempans='1'+tempans[1:]
        templist.append(tempans)
    return templist
docgaps=[34544, 34574, 35569]
if(len(docgaps)>1):
    docgaps=cumulative_sum_to_normal(docgaps)
    for i in range(len(docgaps)):
        print(vbencode(docgaps[i]))
else:
    print(vbencode(docgaps))
```

```
['00000010', '00001101', '11110000']
['10011110']
['00000111', '11100011']
```

vi. Apply Elias Delta Decoding for “00101001”

In [ ]:

```
import math

def decode_elias_delta(x):
    count=0
    for i in x:
        if(i=='1'):
            break
        else:
            count=count+1
    y=x[count:count+1+count]
    exp=int(y,2)-1
    rest=int(x[2*count+1:],2)
    ans=pow(2,exp)+rest
    return ans

binary_num = '00101001'
print("The Decoded Value of 00101001 is ", decode_elias_delta(binary_num))
```

```
The Decoded Value of 00101001 is  17
```

Question 3 : Signature Files

In [ ]:

```
import hashlib
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download("punkt")
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ayuar\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ayuar\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[ ]:

```
True
```

In [ ]:

```
d=dict()
def preprocess(doc,indx):
    doc=doc[indx].lower()
    word_tokens = word_tokenize(doc)
    stop_words = set(stopwords.words('english'))
    for i in range(0,len(word_tokens)):
        if(word_tokens[i]==',' or word_tokens[i].lower() in stop_words or word_toke
            continue
        else:
            if indx in d.keys():
                d[indx]+=" "+word_tokens[i]
            else:
                d[indx]=word_tokens[i]
def generate_hash(word):
    hash=int(hashlib.sha256(word.encode()).hexdigest(),16)%(2**30)
    binary_hash=bin(hash)[2:].zfill(30)
    return binary_hash

def divide_sentence(sentence,n):
    words=sentence.split()
    num_blocks=len(words)//n+(len(words)%n>0)
    blocks=[(" ").join(words[i*n:(i+1)*n]) for i in range(num_blocks)]
    return blocks
def orval(sentence):
    x=sentence.split(" ")
    res=generate_hash(x[0])
    for i in range(1,len(x)):
        temp=generate_hash(x[i].lower())
        z=str(temp)
        y=str(res)
        int_1 = int(z, 2)
        int_2 = int(y, 2)
        result = int_1 | int_2
        res=bin(result)[2:].zfill(30)
    return res
sentence="This is a text. A text has many words. Words are made from letters. The t
sentence = sentence.translate(str.maketrans('', '', string.punctuation))
ans=divide_sentence(sentence,4)
print(len(ans))
d1=dict()
```

```
for i in range(len(ans)):
    preprocess(ans,i)
for i,j in d.items():
    y=orval(j)
    d1[i]=y
```

```
7
```

Text

In [ ]:

```
n="Text"
n=n.lower()
ans1=generate_hash(n)
ans2=int(ans1,2)
print(ans)
for i,j in d1.items():
    y=int(str(ans1),2)
    x=int(str(j),2)
    res=y&x
    if(res==ans2):
        print("Found in Block {} consisting of {}".format(i+1,ans[i]))
```

```
['This is a text', 'A text has many', 'words Words are made', 'from letters The te
xt', 'is made of letters', 'Made many words letters', 'text Letters are text']
Found in Block 1 consisting of This is a text
Found in Block 2 consisting of A text has many
Found in Block 4 consisting of from letters The text
Found in Block 7 consisting of text Letters are text
```

Words

In [ ]:

```
n="Words"
n=n.lower()
ans1=generate_hash(n)
ans2=int(ans1,2)
print(ans)
for i,j in d1.items():
    y=int(str(ans1),2)
    x=int(str(j),2)
    res=y&x
    if(res==ans2):
        print("Found in Block {} consisting of {}".format(i+1,ans[i]))
```

```
['This is a text', 'A text has many', 'words Words are made', 'from letters The te
xt', 'is made of letters', 'Made many words letters', 'text Letters are text']
Found in Block 3 consisting of words Words are made
Found in Block 6 consisting of Made many words letters
```

Made

In [ ]:

```
n="Made"
n=n.lower()
ans1=generate_hash(n)
ans2=int(ans1,2)
print(ans)
for i,j in d1.items():
    y=int(str(ans1),2)
    x=int(str(j),2)
    res=y&x
    if(res==ans2):
        print("Found in Block {} consisting of {}".format(i+1,ans[i]))
```

```
['This is a text', 'A text has many', 'words Words are made', 'from letters The te
xt', 'is made of letters', 'Made many words letters', 'text Letters are text']
Found in Block 3 consisting of words Words are made
Found in Block 5 consisting of is made of letters
Found in Block 6 consisting of Made many words letters
```

Letters

In [ ]:

```python
n="Letters"
n=n.lower()
ans1=generate_hash(n)
ans2=int(ans1,2)
print(ans)
for i,j in d1.items():
    y=int(str(ans1),2)
    x=int(str(j),2)
    res=y&x
    if(res==ans2):
        print("Found in Block {} consisting of {}".format(i+1,ans[i]))
```

```
['This is a text', 'A text has many', 'words Words are made', 'from letters The te
xt', 'is made of letters', 'Made many words letters', 'text Letters are text']
Found in Block 4 consisting of from letters The text
Found in Block 5 consisting of is made of letters
Found in Block 6 consisting of Made many words letters
Found in Block 7 consisting of text Letters are text
```