Web Mining

Jaydeep Dey - 20BCE1419

In [ ]:

```python
# Apply run length encoding for the following string and compress it

word = "eeeeeeeeffffffe rrrrrrttt"

character = word[0]
ans = []
count = 0
for i in range(len(word)):
    if(word[i] == character):
        count += 1
    else:
        ans.append(character)
        ans.append(str(count))
        character = word[i]
        count = 1
print("".join(ans))
```

```
e8f6e1r6
```

In [ ]:

```python
# 2.    Consider the following Inverted Index File with Terms, Occurrences and Posti

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

In [ ]:

```python
planets = [1, 2, 3, 7, 9, 10]
for planet in planets:
    print(bin(planet)[2:], end=" ")
```

```
1 10 11 111 1001 1010
```

In [ ]:

```python
# ii.   Apply Unary coding for term “Fiber”
fibre = [1, 3, 5, 7, 19, 20]
def unary(n):
    return "0" * n + "1"

for i in fibre:
    print(unary(i), end=" ")
```

```
01 0001 000001 00000001 00000000000000000001 000000000000000000001
```

In [ ]:

```python
# iii.  Apply Elias Gamma Encoding for term “Airtel”

airtel = [12, 17, 25, 148, 156, 159, 172]

from math import log2, floor

def elias_gamma(n):
    l = floor(log2(n))
    return "0" * l + "1" + bin(n)[3:]

for i in range(len(airtel)):
    print(elias_gamma(airtel[i]), end=" ")
```

```
0001100 000010001 000011001 000000010010100 000000010011100 000000010011111 00000001
0101100
```

In [ ]:

```
# elias delta encoding
# def elias_delta(n):
#     L = floor(log2(n))
#     return elias_gamma(L) + bin(n)[3:]
from math import log,floor
mercury = [1, 2, 3, 7, 9, 10]
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
mercury_encoded_list = []
for i in mercury:
    mercury_encoded_list.append(EliasDeltaEncode(i))
    print(EliasDeltaEncode(i), end=" ")
```

```
1 0100 0101 01111 00100001 00100010
```

In [ ]:

```
# Decoding mercury
import math

def Elias_Delta_Decoding(x):
        x = list(x)
        L=0
        while True:
                if not x[L] == '0':
                        break
                L= L + 1
        x=x[2*L+1:]
        x.insert(0,'1')
        x.reverse()
        n=0
        for i in range(len(x)):
                if x[i]=='1':
                        n=n+math.pow(2,i)
        return int(n)

for i in range(len(mercury_encoded_list)):
    print(Elias_Delta_Decoding(mercury_encoded_list[i]), end=" ")
```

```
1 2 3 7 9 10
```

In [ ]:

```
# v.    Apply Elias Delta Encoding for term “Venus”
venus = [23, 45, 78, 122, 145]

for i in venus:
    print(EliasDeltaEncode(i), end=" ")
```

```
001010111 0011001101 00111001110 00111111010 00010000010001
```

In [ ]:

```python
# vi.   Apply Elias Delta Decoding for “00101001”
print(Elias_Delta_Decoding("00101001"))
```

```
9
```

In [ ]:

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

sum_array = [1, 3, 9, 12]
normal_array = sum_to_normal(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=sum_to_normal(docgaps)
    for i in range(len(docgaps)):
        print(vbencode(docgaps[i]))
else:
    print(vbencode(docgaps))
```

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

Q3. Signature

In [ ]:

```python
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\jayde\AppData\Roaming\nltk_data...
```

```
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jayde\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_token
            continue
        else:
            if indx in d.keys():
                d[indx]+=" "+word_tokens[i]
            else:
                d[indx]=word_tokens[i]
```

In [ ]:

```
def generate_hash(word):
    hash=int(hashlib.sha256(word.encode()).hexdigest(),16)%(2**30)
    binary_hash=bin(hash)[2:].zfill(30)
    return binary_hash
```

In [ ]:

```
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
```

In [ ]:

```
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
```

In [ ]:

```
sentence="This is a text. A text has many words. Words are made from letters. The te
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
```

In [ ]:

```python
# Made

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 tex
t', '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
```

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 tex
t', '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
```