Dependicies

```
!pip install anytree
!pip install bitarray
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: anytree in /usr/local/lib/python3.7/dist-packages (2.8.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages (fro
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: bitarray in /usr/local/lib/python3.7/dist-packages (2.6.0
```

Imports

```
from anytree.exporter import DotExporter
from PIL import Image
from anytree import Node, RenderTree, NodeMixin
import bitarray
from bitarray import bitarray
from dataclasses import dataclass
import os
import pickle
```

DataClass

```
# Node of a Huffman Tree
@dataclass
class Nodes(NodeMixin):
  def __init__(self,probability, symbol, left = None,right = None):
    self.probability = probability
    self.symbol = symbol
    self.parent = None
    self.left = left
    self.right = right
    if left is not None:
      left.parent = self
    if right is not None:
      right.parent = self
    #the tree direction ( 0/left or 1/right)
    self.code : str = ''
```

Functions

```
""" A supporting function in order to calculate the weights of symbols in specified data """
def CalculateWeights(data : dict):
  symbols = dict()
  for item in data:
    if symbols.get(item) == None:
      symbols[item] = 1
    else:
      symbols[item] += 1
  return symbols


""" A supporting function in order to print the codes of symbols by travelling a Huffman Tree
def CalculateCodes(the_codes : dict, node : Nodes, value = ''):

  # a huffman code for current node
  newValue = ''.join((value,node.code))

  if(node.left):
      CalculateCodes(the_codes, node.left, newValue)
  if(node.right):
      CalculateCodes(the_codes, node.right, newValue)

  if(not node.left and not node.right):
      the_codes[node.symbol] = bitarray(newValue)

  return the_codes
```

Huffman Encoding

```
def HuffmanEncodingTree(the_data):
    weightTable = CalculateWeights(the_data)

    #Empty Tree
    the_nodes = []

    # converting symbols and probabilities into huffman tree nodes
    for symbol in weightTable:
      the_nodes.append(Nodes(weightTable[symbol], symbol))

    while len(the_nodes) > 1:
        # sorting all the nodes in ascending order based on their probability
        the_nodes = sorted(the_nodes, key = lambda x: x.probability)
        # picking two smallest nodes
        right = the_nodes[0]
```

```
        left = the_nodes[1]
        left.code = '0'
        right.code = '1'
        # combining the 2 smallest nodes to create new node
        newNode = Nodes(left.probability + right.probability, left.symbol + right.symbol, lef


        the_nodes.remove(left)
        the_nodes.remove(right)
        the_nodes.append(newNode)

    return the_nodes[0]
```

Data

```
txt = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus ante turpis, iaculis
```

Generate

```
#@title
the_tree_root = HuffmanEncodingTree(txt)
the_codes = dict()
encoding = CalculateCodes(the_codes,the_tree_root)
```

Analysis

Writing encoding and checking filesize

```
with open('encoding.pickle', 'wb') as handle:
    pickle.dump(encoding, handle, protocol=pickle.HIGHEST_PROTOCOL)
encoding_file_size = os.path.getsize("encoding.pickle")
```

Reading back the encoding file

```
encoding_read_from_file = None
with open('encoding.pickle', 'rb') as handle:
    encoding_read_from_file = pickle.load(handle)
```

Compressing data file

```
output_bits = bitarray()
output_bits.encode(encoding_read_from_file, txt)
with open('data.bin', 'wb') as handle:
    pickle.dump(output_bits, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
output_bits_from_file = None
with open('data.bin', 'rb') as handle:
    output_bits_from_file = pickle.load(handle)
```

```
compressed_data_with_huffman_encoding = os.path.getsize("data.bin")
```

Reading encoding file and decoding

```
output_text = ''.join(output_bits_from_file.decode(encoding_read_from_file))
```

Writing the raw text file and checking filesize

```
with open('uncompressed.txt', 'wb') as f:
  f.write(txt.encode('ascii'))
total_uncompressed_filesize = os.path.getsize("uncompressed.txt")
```

Maths to check the compression ratio

```
total_compression_ratio = ((encoding_file_size + compressed_data_with_huffman_encoding)/ floa
print("{}% ratio".format(round(total_compression_ratio,2)))
```

```
81.43% ratio
```

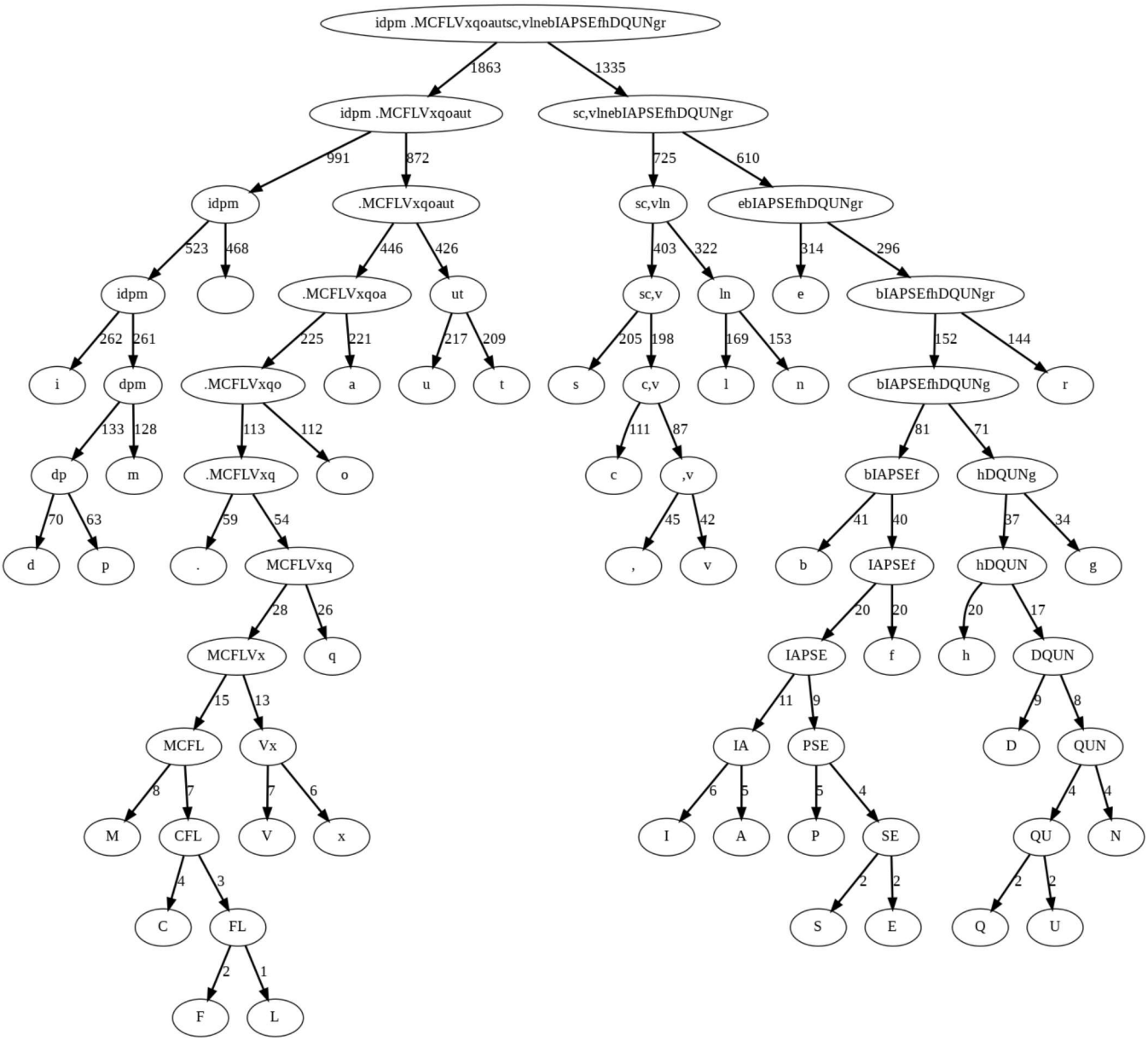
Print the text back

```
print(output_text)
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus ante turpis, iaculis vi
```

Graph

```
DotExporter(the_tree_root,nodenamefunc=lambda node: node.symbol, edgeattrfunc=lambda parent,
out = Image.open('weight.png')
display(out)
```


idpm .MCFLVxqoautsc,vlnebIAPSEfhDQUNgr
1863
1335
idpm .MCFLVxqoaut
sc,vlnebIAPSEfhDQUNgr
991
872
725
610
idpm
.MCFLVxqoaut
sc,vln
ebIAPSEfhDQUNgr
523
468
446
426
403
322
314
296
idpm
.MCFLVxqoa
ut
sc,v
ln
e
bIAPSEfhDQUNgr
262
261
225
221
217
209
205
198
169
153
152
144
i
dpm
.MCFLVxqo
a
u
t
s
c,v
l
n
bIAPSEfhDQUNg
r
133
128
113
112
111
87
81
71
dp
m
.MCFLVxq
o
c
,v
bIAPSEf
hDQUNg
70
63
59
54
45
42
41
40
37
34
d
p
.
MCFLVxq
,
v
b
IAPSEf
hDQUN
g
28
26
20
20
20
17
MCFLVx
q
IAPSE
f
h
DQUN
15
13
11
9
9
8
MCFL
Vx
IA
PSE
D
QUN
8
7
7
6
6
5
5
4
4
4
M
CFL
V
x
I
A
P
SE
QU
N
4
3
2
2
2
2
C
FL
S
E
Q
U
2
1
F
L

✓ 6s completed at 2:23 PM