problem_3_huffman_coding

March 11, 2020

0.0.1 Analyze:

I need to calculate the frequency of each character in the string, so I used Counter . and then I need a sorted list for Huffman coding. and I chose the latest frequency tow item from the list. I need to create a Huffman tree to encoding my data. so I create a Node class and Tree class. I made each unique character as a leaf node.

1. "huffman_encoding" takes O(n), because I using a while loop to create tree and "huffman_decoding" takes O(n)
2. The tree is expending data structures. I think the space complexity is O(n)

[474]:
```
import sys
from collections import Counter

class HuffManTreeNode(object):
    # define a Tree without value and store data by leaf node
    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right

    def set_left_child(self,left):
        self.left = left

    def get_left_child(self):
        return self.left

    def set_right_child(self,right):
        self.right = right

    def get_right_child(self):
        return self.right
```

[475]:
```
class HuffManTree(object):
    def __init__(self, data=None):
        self.frequencies = sorted(Counter(data).most_common(), key=lambda x:␣
 ↪x[1])
```

```
    def get_tree_root(self):
        while len(self.frequencies) > 1:
            # create subtree from the lowest frequencies node to the highest␣
↪frequencies node
            (key1, freq1) = self.frequencies[0]
            (key2, freq2) = self.frequencies[1]
            self.frequencies = self.frequencies[2:]

            # The keys are string type in leaf node, otherwise are node type
            left_node = key1
            right_node = key2
            node = HuffManTreeNode(left_node, right_node)

            self.frequencies.append((node, freq1+freq2))
            self.frequencies = sorted(self.frequencies, key=lambda x: x[1])

        return self.frequencies[0][0]
```

[476]:
```
def huffman_encoding_recursion(node, code=''):
    """
    Encoding characters using recursion
    Args:
      node(HuffManTreeNode):  leaf to save data and non-leaf node assign 0 and␣
↪1 to edges
      code(str): Assign 0 to the left edge and 1 to the right edge

    Returns:
      encode_map(dict) : huffman coding map table
    """
    if type(node) is str:
        return {node: code}

    encode_map = dict()

    left_node = node.get_left_child()
    right_node = node.get_right_child()

    encode_map.update(huffman_encoding_recursion(left_node, code + '0'))
    encode_map.update(huffman_encoding_recursion(right_node, code + '1'))
    return encode_map
```

[477]:
```
def huffman_encoding(data):
    if not data:
        return

    huffman_tree = HuffManTree(data)
```

```
    root_node = huffman_tree.get_tree_root()
    encode_map = huffman_encoding_recursion(root_node)

    encoded_list = [encode_map[c] for c in data]
    encoded_string = ''.join(encoded_list)

    return(encoded_string, root_node)
```

[478]:
```
def huffman_decoding(data,root):
    node = root
    decoded_data = ''

    for b in data:
        if b == '0' and type(node) is HuffManTreeNode:
            node = node.get_left_child()

        elif b == '1' and type(node) is HuffManTreeNode:
            node = node.get_right_child()

        if type(node) is str:
            decoded_data += node
            node = root

    return decoded_data
```

[479]:
```
if __name__ == "__main__":
    codes = {}

    a_great_sentence = "The bird is the word"

    print ("The size of the data is: {}\n".format(sys.
 ↪getsizeof(a_great_sentence)))
    print ("The content of the data is: {}\n".format(a_great_sentence))

    encoded_data, tree = huffman_encoding(a_great_sentence)

    print ("The size of the encoded data is: {}\n".format(sys.
 ↪getsizeof(int(encoded_data, base=2))))
    print ("The content of the encoded data is: {}\n".format(encoded_data))

    decoded_data = huffman_decoding(encoded_data, tree)

    print ("The size of the decoded data is: {}\n".format(sys.
 ↪getsizeof(decoded_data)))
    print ("The content of the encoded data is: {}\n".format(decoded_data))
```

```
The size of the data is: 69

The content of the data is: The bird is the word

The size of the encoded data is: 36

The content of the encoded data is:
011011101111110011100000101011000010001101001111011111101010101100101O

The size of the decoded data is: 69

The content of the encoded data is: The bird is the word
```

[480]:
```
    codes = {}

    a_great_sentence = "Lorem ipsum dolor sit amet, consectetur adipiscing␣
↪elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut␣
↪enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut␣
↪aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in␣
↪voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint␣
↪occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit␣
↪anim id est laborum."

    print ("The size of the data is: {}\n".format(sys.
↪getsizeof(a_great_sentence)))
    print ("The content of the data is: {}\n".format(a_great_sentence))

    encoded_data, tree = huffman_encoding(a_great_sentence)

    print ("The size of the encoded data is: {}\n".format(sys.
↪getsizeof(int(encoded_data, base=2))))
    print ("The content of the encoded data is: {}\n".format(encoded_data))

    decoded_data = huffman_decoding(encoded_data, tree)

    print ("The size of the decoded data is: {}\n".format(sys.
↪getsizeof(decoded_data)))
    print ("The content of the encoded data is: {}\n".format(decoded_data))
```

```
The size of the data is: 494

The content of the data is: Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
```

```
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

The size of the encoded data is: 272

The content of the encoded data is: 0110110101000010011111011111000100001111000111
1101111101110110000001100001001101110000110101101001101111111101001101111101011
0100001011110011111011010101111101001110100110100111101001000010011110010110001
0
```


```
1010110001110111100010011010011011111011100111111101110111011000110111100101111
1
1001011110001110111010101111101110000110000100110001010110110001111010011110101
1
1010110101100111101011000011001011001010000100111111011111010110111011000000110
0
0010011111101011110010110001010110011101001000100100000101111001000000110011011
0
1110101101111010100110111110100111101110101110010101001101111100110011111101010
0
1100110111011011111000000101110011110011001011000111001010010001111110111011101
1
1010011110100101100011010100110100011000010111001110001000110011011110110100011
0
0001100101100101000010000111100110010100111100001110011110101101001000100100000
1
0111001000011101111011010011011111001110101101000101111011110001110110001101011
0
1000010111100111100000101111001101000000011001100000011100111100110100101111010
1
1111100010100011101001111110111011000000110000100110001010111001001111000010100
1
1110110000111110101111011111010000110101100010101110011001110000001011100001101
0
1001101011111100110011111110001001101011011111110011100111111101011000100010001
0
1101111101110110000001100001001111110111011011111001101010111011000100110011010
0
1010111000100011001110000011001010000110011010011101000000001100110110001101001
0
1101111000011010111011101001101110000101011010110100010110101101001111110110100
0
1101010110101100111000010011110110011010100110101100101100001011100000101001000
0
1111011111010110100110111110111000111010110101100010101110101100111000100001100
1
1100000010111001110100001101010110101001101100011001110111011111111100111101000
1
1010110101101011110000001000100110101101001010100110111110001111011101111111001
0
1011000011001011001010000100011110111000000
```


```
The size of the decoded data is: 494

The content of the encoded data is: Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
id est laborum.
```

[483]:
```
    codes = {}

    a_great_sentence = "><?skdownnf()_()_*&^~%%%$$$###@@@_)(*^%$$@!
↪~><MNNB)7783292jdj "

    print ("The size of the data is: {}\n".format(sys.
↪getsizeof(a_great_sentence)))
```

```
    print ("The content of the data is: {}\n".format(a_great_sentence))

    encoded_data, tree = huffman_encoding(a_great_sentence)

    print ("The size of the encoded data is: {}\n".format(sys.
↪getsizeof(int(encoded_data, base=2))))
    print ("The content of the encoded data is: {}\n".format(encoded_data))

    decoded_data = huffman_decoding(encoded_data, tree)

    print ("The size of the decoded data is: {}\n".format(sys.
↪getsizeof(decoded_data)))
    print ("The content of the encoded data is: {}\n".format(decoded_data))
```

```
The size of the data is: 111

The content of the data is:
><?skdownnf()_()_*&^^%%%$$$###@@@_)(*^%$$@!~><MNNB)7783292jdj

The size of the encoded data is: 64

The content of the encoded data is: 0011101110101100101101101110011111011111100001
0100001000011000111111010000001111101000000100011100100001000101010101010111101110
1011100010001000100110011001100000001001111110001000101011110111001101100111101000
0011101110110101100101001011011001001001110011110111111100101001111011010010101011
11111010100110

The size of the decoded data is: 111

The content of the encoded data is:
><?skdownnf()_()_*&^^%%%$$$###@@@_)(*^%$$@!~><MNNB)7783292jdj
```

Resources Huffman Visualization

[ ]: