Week 7 Live Coding Solutions

Week 7 Live Coding Solutions

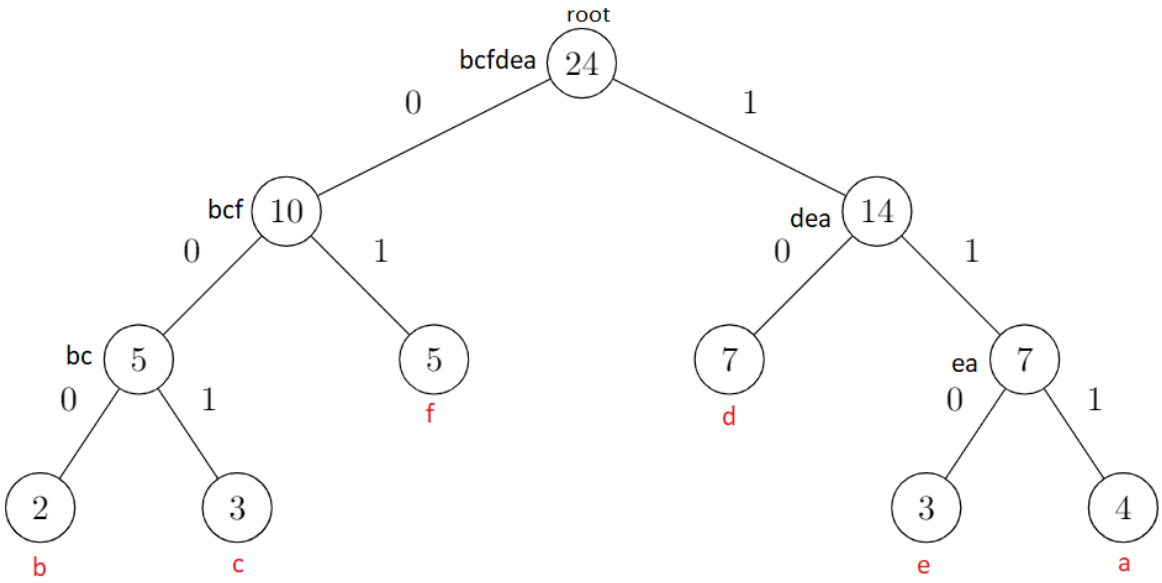
- Week-7 Live coding problem 1
  - Solution
  - Public Test case
  - Private Test case
- Week-7 Live coding problem 2
  - Solution
  - Public Test Case
  - Private test case
- Week-7 Live coding problem 3
  - Solution
  - Public Test case
  - Private Test case

Week-7 Live coding problem 1

Write a function **decode(root, ciphertext)** that accepts a variable `root` which contains the reference of the root node of Huffman tree and an encoded message `ciphertext` in the form of a string (using 0 and 1). The function returns the decoded message in the form of a string.

Structure of node in given Huffman tree

```
class Node:
    def __init__(self,frequency,symbol = None,left = None,right = None):
        self.frequency = frequency
        self.symbol = symbol
        self.left = left
        self.right = right
```



- Leaf node contains the final symbol (in red color)

Sample Input

```
0001110001110000011011001 #Encoded message
```

Output

```
bababcdef # Decoded message
```

Solution

Solution code

```
# Solution
def decode(root,ciphertext):
    message =''
    temp = root
    for i in ciphertext:
        if i == '0':
            temp = temp.left
        if i == '1':
            temp = temp.right
        if temp.left == None and temp.right == None:
            message += temp.symbol
            temp = root
    return message
```

Suffix Code(Hidden)

```
class Node:
    def __init__(self,frequency,symbol=None,left=None,right=None):
        self.frequency = frequency
        self.symbol = symbol
        self.left = left
        self.right = right

def Huffman(s):
    char = list(s)
    freqlist=[]
    unique = set(char)
    for c in unique:
        freqlist.append((char.count(c),c))
    nodes = []
    for nd in sorted(freqlist):
        nodes.append((nd,Node(nd[0],nd[1])))
    while len(nodes) > 1:
        nodes.sort()
        L = nodes[0][1]
        R = nodes[1][1]
        newnode = Node(L.frequency + R.frequency,L.symbol+R.symbol,L,R)
        nodes.pop(0)
        nodes.pop(0)
        nodes.append(((L.frequency+R.frequency,L.symbol+R.symbol),newnode))
    return newnode

# huffman code
'''a 111
b 000
c 001
d 10
e 110
f 01'''

s = 'aaaacccbbddddddeeeffffff'
cipher = input()
res = Huffman(s)
print(decode(res,cipher))
```

Public Test case

Input 1

```
000111000111000000011011001
```

Output

```
bababcdef
```

Input 2

```
11100000110110011110000011011001
```

Output

```
abcdefabcdef
```

Input 3

```
01110100010001111111000000011011001
```

Output

```
fedcbaabcdef
```

Private Test case

Input 1

```
011101000100011111100000110110011110000011011001111000000110110011
```

Output

```
fedcbaabcdefabcdefabcdef
```

Input 2

```
11111111111111111111111111111111
```

Output

```
aaaaaaaaaaa
```

Input 3

```
10011001100110011001001
```

Output

```
dfdfdfdfdfdf
```

Week-7 Live coding problem 2

Write a method `MaxValueSelection(items, C)` that accepts a dictionary `items` where each key of the dictionary represents the item name and the corresponding value is a tuple `(number of units, value of all units)` and function accept one more variable `C` which represents the maximum capacity of units you can select from all items to get maximum value.

Sample input

```
{1:(10,60),2:(20,100),3:(30,120)}
50
```

Output

```
240.0
```

Solution

Solution Code

```
def MaxValueSelection(items, C):
    itemlist = []
    for i,j in items.items():
        itemlist.append((j[1]/j[0],i,j[0]))
    itemlist.sort(reverse=True)
    maxvalue = 0
    for itm in itemlist:
        if C > itm[2]:
            maxvalue += itm[0]*itm[2]
            C = C - itm[2]
        else:
            maxvalue += C*itm[0]
            C = 0
            break
    return maxvalue
```

Suffix code(Visible)

```
items = eval(input())
C = int(input())
print(round(MaxValueSelection(items, C),2))
```

Public Test Case

Input 1

```
{1:(10,60),2:(20,100),3:(30,120)}
50
```

Output

```
240.0
```

Input 2

```
{1:(6,110),2:(7,120),3:(3,2)}
10
```

Output

```
178.57
```

Input 3

```
{1:(4,400),2:(9,1800),3:(10,3500),4:(5,4000),5:(2,1000),6:(1,200)}
15
```

Output

```
7800.0
```

Input 4

```
{1:(4,400),2:(9,1800),3:(10,3500),4:(20,4000),5:(2,1000),6:(1,200)}
20
```

Output

```
6100.0
```

Input 5

```
{1:(4,1600),2:(9,2700),3:(10,3500),4:(5,4000),5:(2,1000),6:(2,1200),7:
(2,1350),8:(9,1800),9:(10,2300),10:(5,1530),11:(2,100),12:(1,120),13:
(2,1600),14:(3,2700),15:(7,3500),16:(8,4000),17:(1,1000),18:(6,1200),19:
(1,1350),20:(10,1800),21:(2,2300),22:(10,1530),23:(4,100),24:(1,125)}
1
```

Output

```
1350.0
```

Private test case

Input 1

```
{1:(4,400),2:(9,1800),3:(10,3500),4:(5,4000),5:(2,1000),6:(1,200)}
8
```

Output

```
5350.0
```

Input 2

```
{1:(4,1600),2:(9,2700),3:(10,3500),4:(5,4000),5:(2,1000),6:(2,1200),7:
(2,1350),8:(9,1800),9:(10,2300),10:(5,1530),11:(2,100),12:(1,120)}
25
```

Output

```
12650.0
```

Input 3

```
{1:(4,1600),2:(9,2700),3:(10,3500),4:(5,4000),5:(2,1000),6:(2,1200),7:
(2,1350),8:(9,1800),9:(10,2300),10:(5,1530),11:(2,100),12:(1,120)}
10
```

Output

```
7050.0
```

Input 4

```
{1:(4,1600),2:(9,2700),3:(10,3500),4:(5,4000),5:(2,1000),6:(2,1200),7:
(2,1350),8:(9,1800),9:(10,2300),10:(5,1530),11:(2,100),12:(1,120),13:
(2,1600),14:(3,2700),15:(7,3500),16:(8,4000),17:(1,1000),18:(6,1200),19:
(1,1350),20:(10,1800),21:(2,2300),22:(10,1530),23:(4,100),24:(1,125)}
30
```

Output

```
21500.0
```

Input 5

```
{1:(4,1600),2:(9,2700),3:(10,3500),4:(5,4000),5:(2,1000),6:(2,1200),7:
(2,1350),8:(9,1800),9:(10,2300),10:(5,1530),11:(2,100),12:(1,120),13:
(2,1600),14:(3,2700),15:(7,3500),16:(8,4000),17:(1,1000),18:(6,1200),19:
(1,1350),20:(10,1800),21:(2,2300),22:(10,1530),23:(4,100),24:(1,125)}
2
```

Output

```
2500.0
```

Week-7 Live coding problem 3

Write a function `IsCodeValid(hfcode, message)` that accept a dictionary `hfcode` in which key represents the character and corresponding value represents the Huffman code for that character and function accept one more string `message` (encoded message generated using Huffman codes). The function returns `True` if `message` is valid, otherwise return `False`.

Sample Input

```
{'a':'000', 'b':'0010', 'c':'0011', 'd':'01', 'e':'10', 'f':'11'} #huffman code
101010110101000000010011 #Encoded message
```

Output

```
False
```

Sample Input

```
{'a':'000', 'b':'0010', 'c':'0011', 'd':'01', 'e':'10', 'f':'11'}
111010111001011100011
```

Output

```
True
```

Solution

```
def IsCodeValid(hfcode,message):
    emsg = ''
    huffcode ={}
    maxlength=0
    for i,j in hfcode.items():
        huffcode[j]=i
        if len(j) > maxlength:
            maxlength=len(j)
    cd = ''
    for b in message:
        cd += b
        if len(cd) > maxlength:
            return False
        if cd in huffcode:
            emsg += huffcode[cd]
            cd = ''
    if cd == '':
        return True
    else:
        return False
```

Suffix Code(visible)

```
hfcode = eval(input())
message = input()
print(IsCodeValid(hfcode,message))
```

Public Test case

Input 1

```
{'a':'000', 'b':'0010', 'c':'0011', 'd':'01', 'e':'10', 'f':'11'}
10101011010100000010011
```

Output

```
False
```

Input 2

```
{'a':'000', 'b':'0010', 'c':'0011', 'd':'01', 'e':'10', 'f':'11'}
111010111001011100011
```

Output

```
True
```

Input 3

```
{'a':'111', 'b':'000', 'c':'001', 'd':'10', 'e':'110', 'f':'01'}
11100000110110010111010001000111
```

Output

```
True
```

Private Test case

Input 1

```
{'a':'111', 'b':'000', 'c':'001', 'd':'10', 'e':'110', 'f':'01'}
11100000110110010111010000010001111
```

Output

```
False
```

Input 2

```
{'a':'111', 'b':'000', 'c':'001', 'd':'10', 'e':'110', 'f':'01'}
1101110111111101010111110110
```

Output

```
True
```

Input 3

```
{'a':'111', 'b':'000', 'c':'001', 'd':'10', 'e':'110', 'f':'01'}
111011
```

Output

```
False
```