Week 7 Practice Programming Assignment

Problem 1

Write a function **Huffman(s)** that accepts a string `s` of characters `a,b,c,d,e and f` without any space. The function should generate the prefix code for each character based on its frequency in string `s` and return a dictionary where the key of the dictionary represents the character and the corresponding value represents the Huffman code for that character.

Consider the following points to create a unique Huffman tree to generate codes:

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

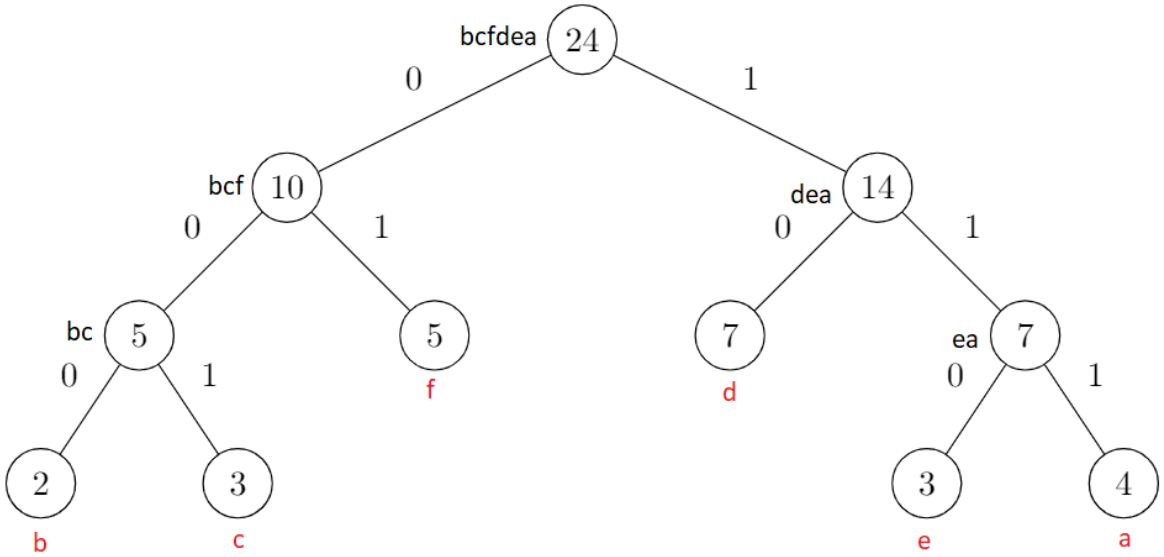
Select two smallest frequency nodes each time. If more than two nodes have the same smallest frequency, then select nodes in the lexicographical order of their symbol. Assume that `x` and `y` are the two smallest nodes, then:-

- If `x.frequency < y.frequency` then `x` will always come on the left and `y` will always come on the right of the parent node.
- If `x.frequency = y.frequency` then the symbol of node, which comes first in lexicographical order, will become the left child, and others will become the right child of the parent node.
- If `x` is a left node and `y` is a right node, then the parent node will be identified by `x.symbol + y.symbol` for further creation of the tree.

Example:-

```
s = 'aaaacccbbddddddeeeffffff'
Frequency of each character
a - 4
b - 2
c - 3
d - 7
e - 3
f - 5
```

Generated Huffman tree



Sample Input

```
aaaacccbbddddddeeeffffff
```

Output

```
a 111
b 000
c 001
d 10
e 110
f 01
```

Solution

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

# Solution

def Huffman(s):
    huffcode = {}
    char = list(s)
    freqlist = []
    unique_char = set(char)
    for c in unique_char:
        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))

    for ch in unique_char:
        temp = newnode
        code = ''
        while ch != temp.symbol:
            if ch in temp.left.symbol:
                code += '0'
                temp = temp.left
            else:
                code += '1'
                temp = temp.right
        huffcode[ch] = code
    return huffcode


# Suffix code(visible)
s = input()
res = Huffman(s)
for char in sorted(res):
    print(char,res[char])
```

Public Test case

Input 1

```
aaaacccbbddddddeeeffffff
```

Output 1

```
a 111
b 000
c 001
d 10
e 110
f 01
```

Input 2

```
aabbccddeeff
```

Output 2

```
a 100
b 101
c 110
d 111
e 00
f 01
```

Input 3

```
aaaaaabbbbbccccdddeef
```

Output 3

```
a 10
b 01
c 00
d 110
e 1111
f 1110
```

Private Test Case

Input 1

```
abbcccddddeeeeeffffff
```

Output 1

```
a 1000
b 1001
c 101
d 00
e 01
f 11
```

Input 2

```
abcdef
```

Output 2

```
a 100
b 101
c 110
d 111
e 00
f 01
```

Input 3

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaffffeeeeeeeeeccccccccccccbbbbbbbbbbbbbbddddddddddddddddd
```

Output 3

```
a 0
b 101
c 100
d 111
e 1101
f 1100
```

Input 4

```
ddddddddddddddddbbbbbbbbbbbbbcccccccccccceeeeeeeeeffffaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Output 4

```
a 0
b 101
c 100
d 111
e 1101
f 1100
```

Input 5

```
aaaaffffbbbeeeccdd
```

Output 5

```
a 00
b 110
c 010
d 011
e 111
f 10
```

Problem 2

A manufacturing plant has `N` independent working lines. Each job requires different amount of time to be completed and due time. Optimize the scheduling of jobs for every line so that every job is will be completed within due time or minimum lateness possible.

Write a function `minimizeLateness(N, jobs)` to return the sequence of job ids to be scheduled in list of list. `jobs` is the tuple of `(job_id, time_required, due_time)`, where `job_id`, `time_required` and `due_time` are the unique number assigned to job, relative time required to complete the job and absolute time before which the job should have completed respectively.

Note:

`job_id` is given from 0 to m-1, for m jobs.

$$Lateness = \sum_{id=0}^{m} L(i)$$

$$L(i) = \begin{Bmatrix} if\ \ TimeDelivered(i) > TimeDue(i),\ \ TimeDelivered(i) - TimeDue(i) \\ else, 0 \end{Bmatrix}$$

For each test case there will be a upper bound of lateness on the second line of input. The lateness of your optimum schedule should be less than or equal to the upper bound of lateness.

Sample Input

```
N = 2
36 # upper bound of lateness
jobs = [ (0, 10, 15),
         (1, 9, 15),
         (2, 2, 10),
         (3, 6, 14),
         (4, 5, 6),
         (5, 5, 7),
         (6, 2, 6),
         (7, 7, 11),
         (8, 3, 5),
         (9, 4, 9) ]
```

Sample Output

```
[[8, 5, 2, 7, 1], [6, 4, 9, 3, 0]] # upper bound of lateness is 36 for N=2
```

Solution

```python
def minimizeLateness(N, jobs):
    sortedL = sorted(jobs, key=lambda x:(x[2], x[1], x[0]))

    optimum = [ [] for i in range(N) ]
    usedtime = { i:0 for i in range(N) }
    i = 0
    while i < len(sortedL):
        freeline = min(usedtime, key=lambda x:usedtime[x])
        optimum[freeline].append(sortedL[i][0])
        usedtime[freeline] += sortedL[i][1]
        i += 1
    return optimum

# suffix (invisible)
def lateness(optimum, jobs):
    jobdict = {job[0]:(job[0],job[1],job[2]) for job in jobs}
    time = 0
    late = 0
    for optID in optimum:
        time += jobs[optID][1]
        overtime = time - jobdict[optID][2]
        late += overtime if overtime >= 0 else 0
    return late

N = int(input())
m = int(input())
jobs = []
while True:
    line = input().strip()
    if line == '':
        break
    t = line.split(' ')
    jobs.append((int(t[0]), int(t[1]), int(t[2])))

optimum = minimizeLateness(N, jobs)
extratime = 0
for i in range(N):
    extratime += lateness(optimum[i], jobs)
if extratime <= m:
    print('Passed')
else:
    print('Improve your algorithm')
```

Test cases

Public Test case 1

Input

```
2
36
0 10 15
1 9 15
2 2 10
```

```
3 6 14
4 5 6
5 5 7
6 2 6
7 7 11
8 3 5
9 4 9
```

Output

```
Passed
```

Public Test case 2

Input

```
2
34
0 1 8
1 1 6
2 3 6
3 7 10
4 5 6
5 7 13
6 3 5
7 2 10
8 9 11
9 1 3
10 8 13
```

Output

```
Passed
```

Public Test case 3

Input

```
3
0
0 7 14
1 8 16
2 7 15
3 7 10
4 3 7
```

Output

```
Passed
```

Private Test case 1

Input

```
2
3
0 1 6
1 4 8
2 9 15
3 9 13
4 9 16
```

Output

```
Passed
```

Privat Test case 2

Input

```
4
2
0 2 9
1 1 2
2 2 8
3 3 10
4 9 11
5 6 12
6 9 11
7 2 8
8 4 11
9 9 11
```

Output

```
Passed
```

Private Test case 3

Input

```
4
2
0 1 8
1 1 6
```

```
2 3 6
3 7 10
4 5 6
5 7 13
6 3 5
7 2 10
8 9 11
9 1 3
10 8 13
```

Output

```
Passed
```

Private Test case 4

Input

```
3
11
0 1 8
1 1 6
2 3 6
3 7 10
4 5 6
5 7 13
6 3 5
7 2 10
8 9 11
9 1 3
10 8 13
```

Output

```
Passed
```

Private Test case 5

Input

```
4
241
0 12 17
1 3 14
2 18 30
3 21 31
4 16 25
5 19 31
6 5 17
7 1 9
```

```
8 27 31
9 11 17
10 8 16
11 30 37
12 3 7
13 16 29
14 11 16
15 23 27
16 6 21
17 22 36
18 19 29
19 20 34
```

Output

```
Passed
```