# Programming language used: Python3

#

# This coding question is to enumerate all uppercase/lowercase permutation

# for any letter specified in $rule\_char\_set

#

# Try to provide an optimal solution

#

# Input:

# word = “medium-one”

# rule\_char\_set = “io”

#

# Output:

# solutions = [“medium-one”, “med**I**um-one”, “medium-**O**ne”, “med**I**um-**O**ne”]

#

# If the character in $rule\_char\_set appears more than once in $word, treat # them as independent variable. For example:

#

# Input:

# word = “medium-one”

# rule\_char\_set = “m”

# Output:

# solutions = [“medium-one”, “Medium-one”, ‘mediuM-one”, “MediuM-one”]

#

Code:

class Solution:

def permute(self, word, rule):

word=word.lower()

rule=rule.lower()

l=[]

sol=[]

for i in range(len(word)):

if((word[i] in rule) and (word[i]>='a' and word[i]<='z')):

l.append(i)

x=1<<(len(l))

for i in range(0,x):

s=list(word)

a=bin(i)[2:].zfill(len(l))

for j in range(0,len(a)):

if(a[j]=='1'):

s[l[j]]=s[l[j]].upper()

sol.append(''.join(s))

return sol

word="medium-one"

rule="m"

a=Solution()

permute\_ul = a.permute

print (permute\_ul(word, rule))

print (permute\_ul('medium-one', 'me'))

print (permute\_ul('m', 'me'))

print (permute\_ul('ab-cd', 'ad'))

print (permute\_ul('ab-cd', 'a-'))

print (permute\_ul('a3b', 'ab'))

print (permute\_ul('a3bcdddd', 'acd'))

UNIT TEST:

Test case 1:

Word: "medium-one"

Rule: “m”

Output:

['medium-one', 'mediuM-one', 'Medium-one', 'MediuM-one']

Test case 2:

Word: 'medium-one'

Rule: 'me'

Output:

['medium-one', 'medium-onE', 'mediuM-one', 'mediuM-onE', 'mEdium-one', 'mEdium-onE', 'mEdiuM-one', 'mEdiuM-onE', 'Medium-one', 'Medium-onE', 'MediuM-one', 'MediuM-onE', 'MEdium-one', 'MEdium-onE', 'MEdiuM-one', 'MEdiuM-onE']

Test case 3:

Word: 'm'

Rule: 'me'

Output:

['m', 'M']

Test case 4:

Word: 'ab-cd'

Rule: 'ad'

Output:

['ab-cd', 'ab-cD', 'Ab-cd', 'Ab-cD']

Test Case 5:

Word: 'ab-cd'

Rule: 'a-‘

Output:

['ab-cd', 'Ab-cd']

Test Case 6:

Word: 'a3b'

Rule: 'ab'

Output:

['a3b', 'a3B', 'A3b', 'A3B']

Test Case 7:

Word: 'a3bcdddd'

Rule: 'acd'

Output:

['a3bcdddd', 'a3bcdddD', 'a3bcddDd', 'a3bcddDD', 'a3bcdDdd', 'a3bcdDdD', 'a3bcdDDd', 'a3bcdDDD', 'a3bcDddd', 'a3bcDddD', 'a3bcDdDd', 'a3bcDdDD', 'a3bcDDdd', 'a3bcDDdD', 'a3bcDDDd', 'a3bcDDDD', 'a3bCdddd', 'a3bCdddD', 'a3bCddDd', 'a3bCddDD', 'a3bCdDdd', 'a3bCdDdD', 'a3bCdDDd', 'a3bCdDDD', 'a3bCDddd', 'a3bCDddD', 'a3bCDdDd', 'a3bCDdDD', 'a3bCDDdd', 'a3bCDDdD', 'a3bCDDDd', 'a3bCDDDD', 'A3bcdddd', 'A3bcdddD', 'A3bcddDd', 'A3bcddDD', 'A3bcdDdd', 'A3bcdDdD', 'A3bcdDDd', 'A3bcdDDD', 'A3bcDddd', 'A3bcDddD', 'A3bcDdDd', 'A3bcDdDD', 'A3bcDDdd', 'A3bcDDdD', 'A3bcDDDd', 'A3bcDDDD', 'A3bCdddd', 'A3bCdddD', 'A3bCddDd', 'A3bCddDD', 'A3bCdDdd', 'A3bCdDdD', 'A3bCdDDd', 'A3bCdDDD', 'A3bCDddd', 'A3bCDddD', 'A3bCDdDd', 'A3bCDdDD', 'A3bCDDdd', 'A3bCDDdD', 'A3bCDDDd', 'A3bCDDDD']

# Question 2:

# Find the median numbers of two very large array of sorted integers.

# Assuming both arrays have the same large length = N.

# The result should be the same as if you concatenated the two arrays into a # single array, sorted it and then returned the middle two numbers (This is # O(n log n) and thus not an acceptable solution)

#

# For example:

# Input: array\_int1 = [ 10, 20, 30, 40, 51, 61, 71]

# array\_int2 = [ 15, 25, 31, 86, 600, 700, 900]

#

# Output: median numbers = 40, 51

#

#

Code:

#function to find median of a given list a

def localMedian(a):

n=len(a)

if(n%2==0):

return [a[int(n/2)-1],a[int(n/2)]]

else:

return [a[int(n/2)]]

#function to find median of 2 lists given

def getMedian(a1,a2,n):

if(n<=0):

return []

if(n==1):

return [a1[0],a2[0]]

if(n==2):

return [max(a1[0],a2[0]),min(a1[1],a2[1])]

m1=localMedian(a1)

m2=localMedian(a2)

#Median of both lists are equal

if(sum(m1)/len(m1) == sum(m2)/len(m1)):

if(n%2==0):

return [max(m1[0],m2[0]),max(m1[1],m2[1])]

return [m1[0],m2[0]]

#Median of list a1 is less than median of list a2 - if a1=[..,m1,m2,...n] and a2=[1....m1,m2,...] then take find median of a1=[m1..n] and a2=[1..m2]

elif(sum(m1)/len(m1) < sum(m2)/len(m2)):

if(n%2==0):

return getMedian(a1[int(n/2)-1:],a2[0:int(n/2)+1],int(n/2)+1)

else:

return getMedian(a1[int(n/2):],a2[0:int(n/2)+1],int(n/2)+1)

#Median of list a2 is less than median of list a1

else:

if(n%2==0):

return getMedian(a1[0:int(n/2)+1],a2[int(n/2)-1:],int(n/2)+1)

else:

return getMedian(a1[0:int(n/2)+1],a2[int(n/2):],int(n/2)+1)

#helper function - returns median of two given lists

def getMedian\_helper(array\_int1, array\_int2):

median = []

if(len(array\_int1)==len(array\_int2)):

median=getMedian(array\_int1,array\_int2,len(array\_int1))

median.sort()

return median

#assert to check if the median returned is correct

assert(getMedian\_helper([],[]) == [])

assert(getMedian\_helper([],[1]) == [])

assert(getMedian\_helper([1],[]) == [])

assert(getMedian\_helper([1],[1]) == [1,1])

assert(getMedian\_helper([1,2],[3,4]) == [2,3])

assert(getMedian\_helper([1,1,1],[1,1,1]) == [1,1])

assert(getMedian\_helper([1,2,3],[1,2]) == [])

assert(getMedian\_helper([1,2,3],[4,5,6]) == [3,4])

assert(getMedian\_helper([1,3,5],[2,4,6]) == [3,4])

assert(getMedian\_helper([1,2,3],[3,4,5]) == [3,3])

assert(getMedian\_helper([1,2,4,10,20,34],[30,32,34,40,45,50]) == [30,32])

assert(getMedian\_helper([30,32,34,40,45,50],[1,2,4,10,20,34]) == [30,32])

assert(getMedian\_helper([1, 10, 20, 30, 40, 50], [11,12,13,14,15,16]) == [14,15])

assert(getMedian\_helper([1, 10, 15, 30, 40, 50], [11,12,13,14,16,17]) == [14,15])

UNIT TEST:

Test case 1:

[],[]

Median:

[]

Test case 2://Lists with unequal length

[],[1]

Median:

[]

Test case 3:

[1],[]

Median:

[]

Test case 4:

[1],[1]

Median

[1],[1]

Test case 5:

[1,2],[3,4]

Median:

[2,3]

Test case 6:

[1,1,1],[1,1,1]

Median:

[1,1]

Test case 7:

[1,2,3],[1,2]

Median:

[]

Test case 8:

[1,2,3],[4,5,6]

Median:

[3,4]

Test case 9:

[1,3,5],[2,4,6]

Median:

[3,4]

Test case 10:

[1,3,5],[2,4,6]

Median:

[3,4]

Test case 11:

[1,2,3],[3,4,5]

Median:

[3,3]

Test case 12:

[1,2,4,10,20,34],[30,32,34,40,45,50]

Median:

[30,32]

Test case 13:

[30,32,34,40,45,50],[1,2,4,10,20,34]

Median:

[30,32]

Test case 13:

[1, 10, 20, 30, 40, 50], [11,12,13,14,15,16]

Median:

[14,15]

Test case 13:

[1, 10, 15, 30, 40, 50], [11,12,13,14,16,17]

Median:

[14,15]

When the number of integers in the list exceeds the size of memory, chunks of data can be loaded and processed as per the logic given above.