7 Jun 2019, 6 Nov 2019, 9 Nov 2019, 16 Apr 2020

Write a function min_no_of_turns(L) to return the minimal number of turns to finish the game as an integer. The input L is a tuple of collection of the integers provided in the beginning of the game.

In [1]:
```
def min_no_of_turns(order):

    keyList = []
    count = 0
    length = len(order)

    current = 0
    future = 1

    for elt in order:

        keyList.append(elt)

    keyList.sort()

    while future < length:

        if keyList[current] == keyList[future] :

            count += 1
            future += 1


        elif keyList[current] + 1 == keyList[future] :

            current = future
            future += 1


        else:

            count += 1
            current = future
            future += 1


    if current == 0:

        count += 1

    if future == length and current == length - 1:

        count += 1


    return count
```

In [2]:
```
min_no_of_turns ([0 for i in range(1000000)])
```

Out[2]:
```
1000000
```

Out[2]:
```
1000000
```

In [3]:
```
min_no_of_turns ((1, 8, 3, 6, 5, 7, 2, 1))
```

Out[3]: 3

Out[3]: 3

In [4]:
```
min_no_of_turns ((1, 8, 3, 6, 5, 7, 2, 1, 4))
```

Out[4]: 2

Out[4]: 2

In [5]:
```
tup3 = (6,5,4,3,2,1,11,12,13,14,15,16,6,5,16,16)
min_no_of_turns (tup3)
```

Out[5]: 5

Out[5]: 5

In [6]:
```
min_no_of_turns ([i for i in range(1000000)])
```

Out[6]: 1

Out[6]: 1

Give a sequence seq, namely, a string, a list or a tuple (excluding dictionaries and sets), write a function isElementUnique(seq) to check if all the elements inside seq are unique (no duplicate).

In [7]:
```
def isElementUnique(x):

    flag = True

    if isinstance(x, str) == True:

        stringDictionary = {}
        for elt in x:
            if stringDictionary.get(elt) == None:
                 stringDictionary[elt] = '1'
            else:
                flag = False
                break
        return flag

    else:

            dictionary = {}

            for elt in x:

                if dictionary.get(elt) == None:

                    dictionary[elt] = '1'

                else:

                    flag = False
                    break

            return flag
```

In [8]:
```
print(isElementUnique('minions'))
```

```
False
False
```

In [9]:
```
print(isElementUnique('abcdefghijklmnopqrstuvwxyz'))
```

```
True
True
```

In [10]:
```
print(isElementUnique([1,2,3,4,5,6,7]))
```

```
True
True
```

In [11]:
```
print(isElementUnique(['a', 'b', 3 , True, 999, 'a']))
```

```
False
False
```

In [12]:
```
print(isElementUnique((1, 2, 999, 4, 0, 6, (1,2,), 999)))
```

```
False
False
```

In [13]:
```
print(isElementUnique(['aaa', 'bbb', (1,1), 1]))
```

```
True
True
```

Write a function superFibonacciSeqR(t2,n) to return the list of the first n terms of an SFS with its second term as t2 with proper recursions for n > 0. No iteration is allowed in this part.

In [14]:
```
def superFibonacciSeqR1(t2,n,result):

    if n == 1:
        #result.append(1)
        return 1
    elif n == 2:
        #result.append(t2)
        return t2
    elif n == 3:
        result.append(t2+1)
        return t2 + 1
    else:
        x = 2*superFibonacciSeqR1(t2,n-1,result)
        result.append(x)
        return x
```

In [15]:
```
def superFibonacciSeqR(t2,n):

    result = []
    superFibonacciSeqR1(t2,n,result)
    result.insert(0,t2)
    result.insert(0,1)
    return result
```

In [16]:
```
superFibonacciSeqR(10,10)
```

Out[16]:
```
[1, 10, 11, 22, 44, 88, 176, 352, 704, 1408]
```

Out[16]: [1, 10, 11, 22, 44, 88, 176, 352, 704, 1408]

In [17]:
```
sfs1 = superFibonacciSeqR(11,994)
```

In [18]:
```
print(sfs1[-1])
```

```
251134829809281403347287120873437924350329252743484439244628997274301027607406903709343370034928716748655001465051518787153237176334136103968388536906997846967216432222442913720806436056149323637764551144212026757427701748454658614667942436236181162060262417445778332054541324179358384066497007845376
251134829809281403347287120873437924350329252743484439244628997274301027607406903709343370034928716748655001465051518787153237176334136103968388536906997846967216432222442913720806436056149323637764551144212026757427701748454658614667942436236181162060262417445778332054541324179358384066497007845376
```

In [19]:
```
def superFibonacciSeqI1(t2,bound,result):

    result.append(1)
    result.append(t2)
    x = t2 + 1

    while x <= bound:
        result.append(x)
        x *= 2
```

In [20]:
```
def superFibonacciSeqI(t2,bound):

    result = []
    superFibonacciSeqI1(t2,bound,result)
    return result
```

In [21]:
```
superFibonacciSeqI(4,100)
```

Out[21]: [1, 4, 5, 10, 20, 40, 80]

Out[21]: [1, 4, 5, 10, 20, 40, 80]

In [22]:
```
superFibonacciSeqI(4,160)
```

Out[22]: [1, 4, 5, 10, 20, 40, 80, 160]

Out[22]: [1, 4, 5, 10, 20, 40, 80, 160]

In [23]:
```
longSFS = superFibonacciSeqI(20,10**4321)
print(len(longSFS))
```

```
14352
14352
```

In [24]:
```
def getPartition(array, result, n, length):

    if len(array) == n:
        array.sort()
        array.insert(0,0)
        array.append(length)
        if array not in result:
            result.append(array)
        return
    else:
        for elt in array:
            copyArray = array.copy()
            copyArray.remove(elt)
```

```
            getPartition(copyArray, result, n, length)
```

In [25]:
```
def getOpResult(array,leftdigits):

    numbers = []
    length = len(array)
    for i in range(1,length):
        x = array[i-1]
        y = array[i]
        numbers.append(int(leftdigits[x:y]))

    return numbers
```

In [26]:
```
def opList(n,opString,result,op,final):

    stringArray = []

    if n == 0:

        final.append(result)
        return

    else:

        copyResult = result.copy()
        if op != None:
            copyResult.append(op)

        for elt in opString:

            opList(n-1,opString,copyResult,elt,final)
```

In [27]:
```
final = []
opList(4,'+-',[],None,final)
print(final)
```

```
[['+', '+', '+'], ['+', '+', '+'], ['+', '+', '-'], ['+', '+', '-'], ['+', '-',
'+'], ['+', '-', '+'], ['+', '-', '-'], ['+', '-', '-'], ['-', '+', '+'], ['-',
'+', '+'], ['-', '+', '-'], ['-', '+', '-'], ['-', '-', '+'], ['-', '-', '+'], ['-
', '-', '-'], ['-', '-', '-']]
[['+', '+', '+'], ['+', '+', '+'], ['+', '+', '-'], ['+', '+', '-'], ['+', '-',
'+'], ['+', '-', '+'], ['+', '-', '-'], ['+', '-', '-'], ['-', '+', '+'], ['-',
'+', '+'], ['-', '+', '-'], ['-', '+', '-'], ['-', '-', '+'], ['-', '-', '+'], ['-
', '-', '-'], ['-', '-', '-']]
```

In [28]:
```
def opArray(array, opArray):

    length = len(array)
    result = 0

    for i in range(0,length):

        if opArray[i] == '+':

            result += array[i]

        elif opArray[i] == '-':

            result -= array[i]
```

```
        elif opArray[i] == '*' and i != 0:

            result *= array[i]

        elif opArray[i] == '*' and i == 0:

            result += array[0]

    return result
```

In [29]:
```
def stringBuilder(array1, array2):

    result = []

    length = len(array1)

    for i in range(length):

        if i == 0 and array2[i] == '*':
            result.append('+')
            result.append(array1[i])
        else:
            result.append(array2[i])
            result.append(array1[i])


    return result
```

In [30]:
```
def sumTo(leftdigits, op, n):

    length = len(leftdigits)
    array = []
    final= []

    for i in range (1,length):
        array.append(i)

    for i in range(1,length):

        result = []
        getPartition(array, result, i,length)
        for elt in result:
            partition = getOpResult(elt,leftdigits)
            operations = []
            opList(len(partition)+1,op,[],None,operations)
            for opSeq in operations:
                if opArray(partition, opSeq) == n:

                    finalString = stringBuilder(partition, opSeq)
                    myfinalString = ' '.join(map(str,finalString))
                    if myfinalString not in final:
                        final.append(myfinalString)


    return final
```

In [31]:
```
 sumTo('199','+-',100)
```

Out[31]:
```
['+ 1 + 99']
```

Out[31]:
['+ 1 + 99']

In [32]:
```
sumTo('123456789','+-',100)
```

Out[32]:
```
['+ 123 - 45 - 67 + 89',
 '+ 123 + 45 - 67 + 8 - 9',
 '+ 123 + 4 - 5 + 67 - 89',
 '+ 123 - 4 - 5 - 6 - 7 + 8 - 9',
 '+ 12 + 3 - 4 + 5 + 67 + 8 + 9',
 '+ 12 + 3 + 4 + 5 - 6 - 7 + 89',
 '+ 12 - 3 - 4 + 5 - 6 + 7 + 89',
 '+ 1 + 23 - 4 + 56 + 7 + 8 + 9',
 '+ 1 + 23 - 4 + 5 + 6 + 78 - 9',
 '+ 1 + 2 + 34 - 5 + 67 - 8 + 9',
 '+ 1 + 2 + 3 - 4 + 5 + 6 + 78 + 9',
 '- 1 + 2 - 3 + 4 + 5 + 6 + 78 + 9']
```

Out[32]:
```
['+ 123 - 45 - 67 + 89',
 '+ 123 + 45 - 67 + 8 - 9',
 '+ 123 + 4 - 5 + 67 - 89',
 '+ 123 - 4 - 5 - 6 - 7 + 8 - 9',
 '+ 12 + 3 - 4 + 5 + 67 + 8 + 9',
 '+ 12 + 3 + 4 + 5 - 6 - 7 + 89',
 '+ 12 - 3 - 4 + 5 - 6 + 7 + 89',
 '+ 1 + 23 - 4 + 56 + 7 + 8 + 9',
 '+ 1 + 23 - 4 + 5 + 6 + 78 - 9',
 '+ 1 + 2 + 34 - 5 + 67 - 8 + 9',
 '+ 1 + 2 + 3 - 4 + 5 + 6 + 78 + 9',
 '- 1 + 2 - 3 + 4 + 5 + 6 + 78 + 9']
```

In [33]:
```
sumTo('111111','+-*',100)
```

Out[33]:
```
['+ 111 - 11 * 1',
 '+ 111 * 1 - 11',
 '- 11 + 111 * 1',
 '- 11 * 1 + 111',
 '+ 1 * 111 - 11',
 '- 1 * 11 + 111']
```

Out[33]:
```
['+ 111 - 11 * 1',
 '+ 111 * 1 - 11',
 '- 11 + 111 * 1',
 '- 11 * 1 + 111',
 '+ 1 * 111 - 11',
 '- 1 * 11 + 111']
```

Given a string s, write a function extractParenthesesI(s)to extract all the parentheses and return a string of parentheses using iterations only. You do not have to care if the string s is a correct arithmetic expression or not.

In [34]:
```
def extractParenthesesI(pString):

    result = ''

    for elt in pString:

        if elt == '(':

            result += elt

        elif elt == ')':

            result += elt

    return result
```

In [35]:
```
print(extractParenthesesI('(1+Y)*(3+(X-5))'))
```

```
()(())
()(())
```

In [36]:
```
def extractParenthesesR(pString):

    if len(pString) == 0:

        return ''

    elif pString[0] == '(':

        return  '(' + extractParenthesesR(pString[1:])

    elif pString[0] == ')':

        return  ')' + extractParenthesesR(pString[1:])

    else:

        return extractParenthesesR(pString[1:])
```

In [37]:
```
print(extractParenthesesR('(1+Y)*(3+(X-5))'))
```

```
()(())
()(())
```

Follow the normal mathematic conventions, write a function cbp(s) (cbp = check balanced parentheses) to check if an expression has the correct parentheses balanced.

In [38]:
```
def cbp(pString):

    checkP = []

    for elt in pString:

        if elt == '(':
            check = []
            check.append(elt)
            check.append(False)
            checkP.append(check)
        elif elt == ')':
            check = []
            check.append(elt)
            check.append(False)
            checkP.append(check)

    length = len(checkP)

    for i in range(length):

        if checkP[i][0] == '(':

            flag = False

            for j in range(i,length):

                if checkP[j][0] == ')' and flag == False and checkP[j][1] == False:
```

```
                    flag = True
                    checkP[j][1] = True
                    checkP[i][1] = True

    for elt in checkP:

        if elt[1] == False:
            return False

    return True
```

In [39]:
```
print(cbp('(1+2)*(3+(4-5))'))
```

```
True
True
```

In [40]:
```
print(cbp('(1+2)*(3+(4-5)))'))
```

```
False
False
```

In [41]:
```
print(cbp('(1+2)-Y*X+(3+(4-Z)'))
```

```
False
False
```

Now, in your packet, you have
mandyouwanttoenjoythemostexpensiveburgeryoucanbuyfromourcafé!Writeafunctio
returnatuplethatcontainsthemostexpensiveburgerthatyoucanmakeandthechangeyou
m. If you order the ingredient by their prices in ascending order, that each price is at least doubled as the previous price. Namely, the price for 'P' is at least 6 because the previous price for 'V' is 3. However, you cannot assume the dictionary given is sorted.

In [42]:
```
priceList0 = {'C':1,'V':3,'P':11,'A':31}
priceList1 = {'C':1,'W':2,'I':4,'T':9,'O':20,'V':41,'S':85}
```

In [43]:
```
def buyMaxBurger(priceList,m):

    if m == 0:
        return ('',0)

    result = ''
    valueArray = []

    for item in priceList.items():

        valueArray.append(item[1])

    valueArray.sort()
    valueDict = {}
    length = len(valueArray)
    expense = m
    startIndex = length-1


    for elt in priceList.items():

        valueDict[elt[1]] = elt[0]


    for i in range(length):
```

```
        if valueArray[i] > m:

            startIndex = i-1
            break

    while startIndex > -1:

        if expense - valueArray[startIndex]  > 0:
            expense -= valueArray[startIndex]
            result += valueDict[valueArray[startIndex]]

        startIndex -= 1

    return ('B'+result+'B',expense)
```

In [44]:
```
print(buyMaxBurger(priceList0,26))
```

```
('BPVCB', 11)
('BPVCB', 11)
```

In [45]:
```
print(buyMaxBurger(priceList1,55))
```

```
('BVTIB', 1)
('BVTIB', 1)
```