Problem23

April 17, 2022

[ ]:

1 Problem 23

A perfect number is a number for which the sum of its proper divisors is exactly equal to the number. For example, the sum of the proper divisors of 28 would be $1 + 2 + 4 + 7 + 14 = 28$, which means that 28 is a perfect number.

A number n is called deficient if the sum of its proper divisors is less than n and it is called abundant if this sum exceeds n.

As 12 is the smallest abundant number, $1 + 2 + 3 + 4 + 6 = 16$, the smallest number that can be written as the sum of two abundant numbers is 24. By mathematical analysis, it can be shown that all integers greater than 28123 can be written as the sum of two abundant numbers. However, this upper limit cannot be reduced any further by analysis even though it is known that the greatest number that cannot be expressed as the sum of two abundant numbers is less than this limit.

Find the sum of all the positive integers which cannot be written as the sum of two abundant numbers.

Create a class called 'abundantnum', this creates an array based on the prime sieve class (so a bit array)

[ ]:
```
from bitarray import bitarray
import copy


class AbundantNum:
    # The sieve itself is stored as a boolean array where if array[n]=false␣
 ↪then n is not abundant
        # if array[n]=true then n is abundant
    # size is how big the sieve should be (i.e. the highest number to be␣
 ↪checked for 'abundantness'

    def __init__(self, size):
        self.sieveArray = bitarray(size)
        self.sieveArray.setall(False)
        self.maxNum = size
        self.currentNum = 1
```

```
        for n in range(size):
            m = 1
            divList = [] #list of divisors for each number
            divLimit = n/2
            while m <= divLimit:
                if n % m == 0:
                    divList.append(m)
                m = m + 1
            if sum(divList) > n:
                #if n % 2 != 0: print(n) #print odd abundant numbers
                self.sieveArray[n] = True
        print("Array of abundant numbers up to ", size, " created.")

        self.firstAbundant()

    def isAbundant(self, num):
        return self.sieveArray[num]

    def printSieve(self):
        return self.sieveArray.to01()

    def myIndex(self, value, start, end):
        try:
            return self.sieveArray.index(value, start, end)
        except ValueError:
            print("Oops")
            return 0

    def setNum(self, value):
        self.currentNum = value
        if self.isAbundant(self.currentNum):
            return self.currentNum
        else:
            print("Error in setting number")

    def nextAbundant(self):
        self.currentNum = self.myIndex(1, self.currentNum + 1, self.maxNum)
        return self.currentNum

    def firstAbundant(self):
        self.currentNum = self.myIndex(1, 0, self.maxNum)
        return self.currentNum
```

[ ]:
```
numLimit = 5000

AN = AbundantNum(numLimit)
```

```
AN2 = copy.deepcopy(AN)

notAbundantSumList = []

# a + b = c
a = AN.firstAbundant()
b = AN2.firstAbundant()
c = 1
sumofabundant = False

while c <= numLimit:
    a = AN.firstAbundant()
    b = AN2.firstAbundant()
    notAbundantSumList.append(c)
    sumofabundant = False
    while a + b <= c and sumofabundant == False:
        while a + b <= c and sumofabundant == False:
            if a + b == c:
                notAbundantSumList.remove(c)
                sumofabundant = True
            b = AN2.nextAbundant()
        a = AN.nextAbundant()
        b = AN2.setNum(a)
    c = c + 1

print(sum(notAbundantSumList))
print(notAbundantSumList)
```

The first odd abundant number is 945, so no odd sums of abundant numbers below 957. 1500 = 7.5s 5000 = 3min 9s

Take a slightly different approach? c - b = a, just subtract b and *check* whether a is abundant, rather than trying all possible values.

[ ]:
```
# c - a = b
numLimit = 28123

AN = AbundantNum(numLimit) #creates an object that contains all the abundant␣
 ↪numbers upto the numLimit
notAbundantSumList = [] #list to contain all numbers that are not the sum of␣
 ↪abundant numbers

c = 1
sumofabundant = False

while c <= numLimit:
    a = AN.firstAbundant()
```

```
    notAbundantSumList.append(c) #each number added to list, then removed if␣
↪found to be the sum of abundant numbers
    sumofabundant = False
    while a < c - 11 and sumofabundant == False:    #term in c-11 is there to␣
↪prevent a from being advanced to a number
                                                    #beyond the size of AN and␣
↪so throwing an error.
        b = c - a
        if AN.isAbundant(b):
            notAbundantSumList.remove(c)
            sumofabundant = True
        a = AN.nextAbundant()
    c = c + 1

print(sum(notAbundantSumList))
```

```
Array of abundant numbers up to  28123  created.
4179871
```

1500 = 0.3s 5000 = 1.9s 28123 = 31.3s (~6x faster than first attempt - most of this time is creating the AN object)

Gives the correct answer 4179871