Algorithms - Series Two(iii)

Highly Composite Numbers

The self-taught Indian mathematician Ramanujan was interested in numbers which are, in a sense, the opposite of primes. A Highly Composite Number is one that has more divisors than any number lower than itself. For instance, 12 is an HCN because it has 6 divisors:

```
12 divides by: 1, 2, 3, 4, 6, 12
```

The next HCN is 24, with 8 divisors.

```
24 divides by: 1, 2, 3, 4, 6, 8, 12, 24
```

Ramanujan then looked at HCNs in terms of their Prime Factors. For instance:

$$12 = 2^3 \times 3^1$$

The following code is not particularly interesting from an algorithmic point of view, but it does allow us too look at this pattern of Ramanujan.

```
# ramanujan_hcn.py
# a look at Ramanujan's discoveries about HCNs.

def main():
    print("*** Ramanujan's Highly Composite Numbers ***")
    print("Enter a number and I will show you the next 5 HCNs.")
    print("Enter 0 to quit.")
    while 1:
        try:
            a = (int(input(">> ")))
            if a != 0:
                next_5(a+1)
            else:
                return
        except ValueError:
            print("Huh?")

def next_5(x):
    """Takes a number and looks for next 5 HCNs, printing as it goes."""
    a = 0
    y = 2
    while a < 5:
        y = is_hcn(x, y)
        if y:
            print(pretty(x, y, factor(x)))
            a += 1
        x += 1

def is_hcn(x, y = 2):
    """Brute-force check to see if x is an HCN."""
    divs = divisors(x)
    for i in range(y, x):
        if not divs > divisors(i):
            return False
    return divs

def divisors(x):
    """Takes a number and returns the number of divisors."""
    a = 0
    for i in range(1, x+1):
        if x % i == 0:
            a += 1
    return a
```

```
def pretty(x, d, f):
    """Takes a dictionary and prints it out."""
    l= [str(x) + ": " + str(d)+ " divisors \nPrime Factors: "]
    for x in iter(f):
        l.append(str(x) + '^' + str(f[x]))
    string = l[0]
    for item in l[1:-1]:
        string = string + item + " x "
    return string + l[-1]

def factor(n):
    """Takes an integer and returns a dictionary of prime factors."""
    i = 2
    factors = {}
    while n != 1:
        while n % i == 0:
            n = n // i
            if i not in factors:
                factors[i] = 1
            else:
                factors[i] += 1
        i += 1
    return factors

if __name__ == "__main__":
    main()
```

Here's some sample output:

```
>> 36
48: 10 divisors
Prime Factors: 2^4 x 3^1
60: 12 divisors
Prime Factors: 2^2 x 3^1 x 5^1
120: 16 divisors
Prime Factors: 2^3 x 3^1 x 5^1
180: 18 divisors
Prime Factors: 2^2 x 3^2 x 5^1
240: 20 divisors
Prime Factors: 2^4 x 3^1 x 5^1
>>
```

Task:

Can you spot the pattern that Ramanujan noticed?

Hints:
Look at the exponents of the prime factors.
4 and 36 are exceptions to the pattern.

Extension:
Once you have seen the pattern in the Prime Factors and their exponents, can you write a program that finds HCNs using these facts?

Roundness

A related topic that Ramanujan and Hardy worked on together is the notion of roundness[1]. By roundness, we mean how many times each prime divisor occurs in the factorisation of a number. Some numbers are very much "rounder" than others!

		roundness
999,993	is 3^1 x 333,33^1	2
1,000,000	is 2^6 x 3^6	12

We can quite easily add a function to our program that uses the dictionary produced by factor() to calculate roundness:

[1] There is good material on this and other topics in "The Man Who Loved Only Numbers" by Paul Hoffmann (it's the biography of Paul Erdős).

```
def roundness(x):
    f = factor(x)
    r = 0
    for key in iter(f):
        r += f[key]
    return r
```

Friendly Numbers and Perfect Numbers

I have strayed away from algorithms a little in this article. However, since we have been looking at a bit of number theory, I thought we could consider “friendly” and “perfect" numbers.

Pythagoras and his followers were interested in “Perfect” numbers. The sum of a perfect number’s proper divisors equals the number itself:

$$6 = 3 + 2 + 1$$
$$28 = 14 + 7 + 4 + 2 + 1$$

Euler noticed that the Perfect Numbers are closely linked to the Mersenne Primes. For each Mersenne Prime 2^p-1, there is a perfect number, $2^{p-1} \times (2^p - 1)$:

$$2^1 \times (2^2 - 1) = 6$$
$$2^2 \times (2^3 - 1) = 28$$
$$2^4 \times (2^5 - 1) = 496$$

Friendly numbers are pairs like 220 and 284. The sum of the proper divisors of 284 is 220 and the sum of the proper divisors of 220 is 284!

It’s quite easy to write a program to search for Friendly Numbers. You’ll need a proper divisors function that might be something like this:

```
def proper(n):
    """Returns a list of the proper divisors of n."""
    my_list = [1]
    for i in range(2, n // 2 + 1):
        if n % i == 0:
            my_list.append(i)
    return my_list
```

I hope you’ve enjoyed looking at a little bit of number theory this time!

Happy Hacking!