University of Zurich - HS 2013

Computational Science I
Exercise notes: Integers

Tobias Grubenmann

September 24, 2013

Exercise 1

`Primenumber(n)` generates the prime numbers between 1 and n and stores them in the list `primes`. For every prime p we add all the multiples grater or equal than p^2 of this prime to the list of non-primes. If a number hasn't appeared yet on the list of non-primes, it must be a prime, since we update both lists iteratively.

```
class Primenumber:
    def __init__(self, n):
    # generates the prime numbers between 1 and n
        self.primes = []
        self.nonPrimes = []

        for i in range(2, n+1):
            # check wether i is in the nonPrimes list
            prime = True
            for j in range(len(self.nonPrimes)):
                if self.nonPrimes[j] == i:
                    prime = False

            # if i is prime update lists
            if prime:
                self.primes.append(i)

                # add multiples of i to the nonPrimes list
                multiple = i * i
                factor = i
```

```
                    while multiple <= n:
                        self.nonPrimes.append(multiple)
                        factor = factor + 1
                        multiple = factor * i
```

We can now plot the distribution of primes for all primes between 1 and 100′000:

```
from Primenumber import Primenumber
import pylab

# get all primes between 1 and 100000
p = Primenumber(100000)

xAxis = p.primes

# calculate k*ln(p_k)
yAxis = reduce(lambda s, n: s + [n*log(p.primes[n-1])],
   range(1,len(p.primes)+1), [])

# plot p_k against k*ln(p_k)
pylab.xlabel("p_k")
pylab.ylabel("k*ln(p_k)")
pylab.plot(xAxis, yAxis)
```

This generates the following output:

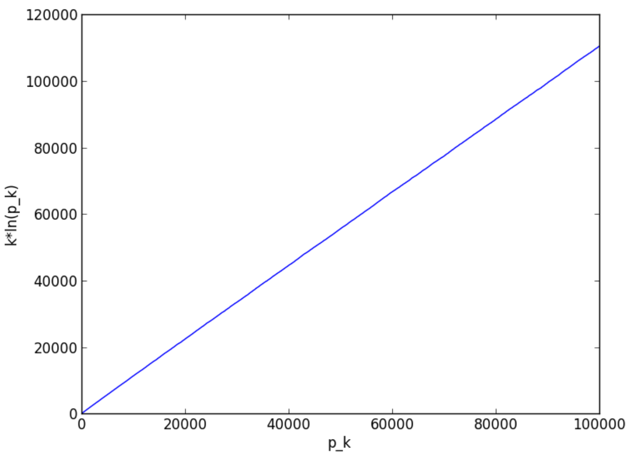

Figure 1: Distribution of primes between 1 and 100′000

Exercise 2

`getZetaAsSum(z)` and `getZetaAsProduct(z)` return the value of the Riemann Zeta-function for a complex number z calculated as a sum and as a product, respectively. Both function uses the first $1'000$ terms for an approximation.

```
from Primenumber import Primenumber

def getZetaAsSum(z):
# gets the value of the riemann zeta function for z

    # returns the first 1000 summands
    return reduce(lambda s, n: s + pow(n, -z), range
        (1,1001), 0)

def getZetaAsProduct(z):
# gets the value of the riemann zeta function for z

    # the first 1000 primes are smaller than 8000
    p = Primenumber(8000)

    # returns the first 1000 factors
    return reduce(lambda s, n: s * 1/(1 - pow(p.primes[n],
        -z)), range(1000), 1)
```

For $z = 2$ we have $\zeta(2) = \frac{\pi^2}{6}$ and the following output:

```
>>> pi*pi/6
1.6449340668482264
>>> getZetaAsSum(2)
1.6439345666815615
>>> getZetaAsProduct(2)
1.6449131747063364
```

Exercise 3

The function `getCarmichaelNumbers(N)` returns all the Carmichael numbers between 1 and N. For each integer i smaller or equal than N we first test whether for each integer j that is coprime to i (i.e. $gcd(i,j) = 1$) i fulfill the equation $j^{i-1} \bmod i \equiv 1$. If so, we test whether the number i is not in the list of primes between 1 and N.

```
from Primenumber import Primenumber
```

```python
def getCarmichaelNumbers(N) :
# calculates all carmichael numbers between 1 and N

    carmichaelNumbers = []

    p = Primenumber(N)

    for i in range(1, N+1) :

        canBeCarmichael = True

        # check all numbers j coprime to i wether they
          fullfil
        # the equation pow(j, i - 1, i) == 1
        for j in range(i) :
            if gcd(i, j) == 1 :
                if pow(j, i - 1, i) != 1 :
                    # can't be a carmichael number
                    canBeCarmichael = False
                    break

        # if i is a candidate for a carmichael number,
        # check if it's prime
        if canBeCarmichael :
            if i not in p.primes :
                carmichaelNumbers = carmichaelNumbers + [i]

    return carmichaelNumbers

def printCarmichaelNumbers(N) :
# prints all carmichael numbers between 1 and N
    c = getCarmichaelNumbers(N)

    for i in range(len(c)) :
        print(c[i])

def gcd(a, b) :
# returns the greates common divisor
    if b == 0 :
        return a
    r = a%b
    return gcd(b, r)
```

The function `printCarmichaelNumbers(10000)` prints the Carmichael numbers between 1 and 10′000:

```
>>> printCarmichaelNumbers(10000)
561
1105
1729
2465
2821
6601
8911
```

Exercise 4

To get the numbers r and d to break the RSA encryption the function `decrypt` goes through all numbers between 1 and N to check for the desired properties:

```
def decrypt(b, c, N) :
    i = 1
    while i <= N :
        if pow(b, i, N) == 1 :
            r = i
            break
        i += 1

    i = 1
    while i <= N :
        if (c*i)%r == 1 :
            d = i
            break
        i += 1

    return word(pow(b, d, N))

def word(n) :
    str = ""
    while n > 0 :
        c = chr(n % 32 + 96)
        if c < "a" or c > "z" :
            c = "␣"
        str += c
        n /= 32
```

```
    return str
```

With this method we can decrypt the encrypted word and get the result:

```
>>> decrypt(100156265, 910510237, 1024384027)
'eureka'
```