Matemática Discreta

Teoría de Números y R

Sesión de laboratorio

Resumen

Muchas de las funciones de la teoría de números están implematadas en el paquete "numbers"del lenguaje R. En esta sesión trabajaremos con algunas de las funciones del paquete e incluso implementaremos los algoritmos vistos en clase.

1. El paquete numbers

Para poder utilizar las funciones integradas en un paquete de R hay que instalar el paquete. Para ello hay que ejecutar el comando que lo instala:

```
> install.packages("numbers")
```

Si ya lo tenemos instalado, para poder hacer uso de las funciones del paquete, hay que cargarlo:

```
> library("numbers")
```

Si necesitáramos ayuda sobre las funciones implementadas, siempre podemos pedirla mediante:

```
> library(help="numbers")
```

lo que nos muestra la lista de funciones implementadas en el paquete. Se puede obtener más información en:

```
http://cran.r-project.org/web/packages/numbers/numbers.pdf
```

Adems, si escribimos el nombre de una función en el intérprete de comandos de R, nos responde con el código asociado al mismo, por ejemplo:

```
> isNatural
```

responde con:

```
function (n)
{
    stopifnot(is.numeric(n))
    floor(n) == ceiling(n) & n >= 1 & n <= 2^53 - 1
}
<environment: namespace:numbers>
```

Donde podemos ver que para que un número sea considerado natural debe ser menor que 2^{53}

2. ejercicios

1. Utilizar algunas funciones del paquete "numbers". Para ver el código asociado a cualquier función basta con escribir el nombre de la función, ya que el nombre es una variable al que se le ha asignado el cdigo de una función. No confundamos una llamada a una función (requiere parentesis tras el nombre de la funcin) con la variable que representa a la función. Por ejemplo:

   ```
   isNatural
   ```

   Cuidado! vemos que en el código los números mayores que $2^{53} - 1$ no se consideran números naturales.

   Veamos las funciones básicas:

   - Números naturales y primos. Para verificar que es natural: isNatural(n), caracteres: is.character(n), primo: isPrime(n), para generar una secuencia de números primos: Primes(n1,n2), para obtener el menor primo mayor que un número dado: nextPrime(n), el previo: previousPrime(n)

     ```
     isNatural(8)
     2^53
     2^52
     isNatural(2^52)
     isNatural(2^53)
     # 15 ceros -> 16 d\'igitos
     isNatural(1000000000000000)
     # 16 ceros-> 17 d\'igitos
     isNatural(10000000000000000)
     is.character(8)
     is.character("8")
     abs(-8)
     isPrime(7)
     isPrime(8)
     Primes(1,20)
     Primes(100,200)
     nextPrime(2000)
     nextPrime(99999)
     previousPrime(100019)
     ```

   - Para verificar que dos números son primos relativos: coprime(n,m)

     ```
     coprime(1448,1001)
     coprime(1448,1002)
     ```

   - Descomposición de números en primos: primeFactors(n)

     ```
     primeFactors(1448)
     primeFactors(1001)
     ```

   - División Euclídea. Para obtener el resultado de la división Euclídea entre dos números n y m dados: div(n,m) aunque también n %/ % m, el resto de la división: mod(n,m) ó n % %m

```
div(7,2)
7%/%2
mod(7,2)
7%%2
```

- El máximo comun divisor (greatest common divisor): GCD(n,m)

```
GCD(1448,1001)
GCD(1448,1002)
```

- La identidad de Bezout: sabemos que el mcd(a, b) se puede expresar como combinación lineal de a y b, y que podemos obtener los coeficientes de dicha combinación mediante el algoritmo de Euclides. Es decir, mcd$(a, b) = k_1a+k_2b$. La función extGCD(a, b) devuelve los valores mcd$(a, b), k_1$ y k_2:

```
extGCD(1448,1001)
[1]    1 -421  609
> 1==-421*1448+609*1001
[1] TRUE
extGCD(1448,1002)
[1]    2 -164  237
> 2==-164*1448+237*1002
[1] TRUE
```

2. Dados a y b, la siguiente funciń escrita en ADA calcula el Máximo Común Divisor de los dos números. La implementación de la función se basa en el algoritmo de Euclides.

```
function MCD(a, b : Positive) return Positive is
        --Aurre:
        --Post: devuelve MCD(a,b)

      r, c, d : Integer;
    begin
        c := a;
        d := b;
        while d /= 0 loop
            r := c mod d;
            c := d;
            d := r;
        end loop;
        return c;
    end zkh;
```

- Escribe la función en lenguaje R. llama a la función que has definido y compara los resultados con lo que devuelve la función `GCD(a,b)`, `coprime(a,b)` y `extGCD(a,b)`.

```
## llamadas a la funci\'on. a=231, b=1820  -------------------------
zkh(231,1820)
coprime(231,1820)
GCD(231,1820)
extGCD(231,1820)
## llamadas a la funci\'on. a=1369, b=2597  ---------------------------
zkh(1369,2597)
```

```
coprime(1369,2597)
GCD(1369,2597)
extGCD(1369,2597)
## llamadas a la funci\'on. a=2689, b=4001  ----------------------------
zkh(2689,4001)
coprime(2689,4001)
GCD(2689,4001)
extGCD(2689,4001)
```

- Mejora la implementación de la función. Si los parámetros recibidos a y b no son enteros haz que devuelva un mensaje de error. Para ello puedes utilizar la funciones `is.character(a)`, `isNatural(a)` y `abs(a)`, y para escribir mensajes en pantalla disponemos de la función `print()`.

```
zkh(3.2,4)
# "los par\'ametros deben ser enteros"

zkh("a","b")
# "los par\'ametros no pueden ser de tipo car\'acter"
```

3. Define la función que devuelva un número primo relativo al parametro que reciba. Necesitaremos la función en el cálculo de claves del algoritmo de encriptación RSA. Implementaremos dos versiones de la función.

   - función que calcula el número natural más pequeño que sea primo relativo al valor m.

```
primo_relativo_minimo <- function(m)
{
}
## llamadas a la funci\'on.
primo_relativo_minimo(13797)
primo_relativo_minimo(16974)
primo_relativo_minimo(56970)
primo_relativo_minimo(1000000000000000)
```

   - En el algoritmo RSA conviene que los números utilizados sean grandes, por lo que en este caso devolverá el número natural más pequeño primo relativo a m que sea mayor que el segundo parámetro t.

```
primo_relativo <- function(m,t)
{
}
## llamadas a la funci\'on.
primo_relativo(13797, 50)
primo_relativo(16974, 54687)
primo_relativo(56970,26378)
primo_relativo(9674,26378)
primo_relativo(1000000000000000,26378)
primo_relativo(1000000000000000,2637800000)
```

4. La codificación ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) es un estándar para la codificación de los caracteres de los alfabetos basados en el alfabeto latino. La mayoría de los sistemas digitales actuales utilizan el código ASCII para representar los caracteres y la información en los ficheros. Tienes más información en la dirección http://eu.wikipedia.org/wiki/ASCII. En la tabla de códigos, en la sección de *los caracteres imprimibles*, puedes ver los códigos que corresponden a cada uno de los caracteres del alfabeto, a los dígitos, a los signos de puntuación etc. En concreto a las 5 letras de la palabra "kaixo"les corresponde los códigos ASCII 107 97 105 120 111.

   En R puedes utilizar la función `strtoi(charToRaw("k"),16L)` para obtener el código ASCII de cualquier letra. Puedes pasarle como parámetro un string o tabla de caracteres...

```
strtoi(charToRaw("k"),16L)
strtoi(charToRaw("a"),16L)
strtoi(charToRaw("i"),16L)
strtoi(charToRaw("x"),16L)
strtoi(charToRaw("o"),16L)
strtoi(charToRaw("kaixo"),16L)
```

   Para la función inversa, es decir, para pasar de código ASCII a letras o dígitos o, en general, a caracteres, podemos utilizar `rawToChar(as.raw(`n`))`. La función no solo acepta un código, también acepta vectores de códigos y devuelve strings.

```
rawToChar(as.raw(107))
rawToChar(as.raw(97))
rawToChar(as.raw(105))
rawToChar(as.raw(120))
rawToChar(as.raw(111))
palabra<-c(107,97,105,120,111)
rawToChar(as.raw(palabra))
```

   Para ocultar un mensaje podemos transformar el código de cada letra, pero dicha transformación debe ser reversible para volver al mensaje original... Escribe una función que modifique el texto original y lo convierta en texto secreto mediante un cambio a tu elección.

   Por ejemplo, puedes convertir el texto en código ASCII, después, podrías sumar 5 a cada código, y puedes convertir los nuevos códigos a texto. Obtendrías lo siguiente:

```
texto_secreto("kaixo")
[1] "pfn}t"
```

   Ya que, "k"$\rightarrow$ 107 $\rightarrow$ 112 $\rightarrow$ "p".

```
texto secreto <- function(texto)
{


}
```

   Así empezó la criptografí en sus orígenes... Tienes un poco de historia en los siguientes artículos:

   http://zientzia.eus/artikuluak/kode-sekretuak-i-ii-iii/