Tablas hash

Complemento

Función fhash

- Lo más simple (en Python): hash()

  `hash(x)` ⟶ `y: entero` ⟶ `y%n`

- Lo siguiente más simple, si x es un entero:

```
def fhash(x,n):
    return x%n
```

- Si x es cualquier cosa y no usamos hash():
  - x ⟶ secuencia ⟶ Pearson hash ⟶ y ⟶ y%n
  - x ⟶ secuencia ⟶ Jenkins hash ⟶ y ⟶ y%n
  - otros métodos de hashing...

Operaciones bit a bit en Python

- AND & 5 & 12 ⟶ 4
- OR | 5 | 12 ⟶ 13
- XOR ^ 5 ^ 12 ⟶ 9
- NOT ~ ~5 ⟶ -6
- RSHIFT >> 37 >> 3 ⟶ 4
- LSHIFT << 37 << 3 ⟶ 296

Cómo generar una permutación aleatoria

```
import random
def init_T(n):
    T = list(range(n))
    random.shuffle(T)
    return T
```

Secuencia de enteros calculada a partir de un objeto cualquiera

```
def get_seq_int(x): # x es un objeto cualquiera
    s=repr(x)       # s es la cadena que representa x
    seq=[]          # seq es una lista de enteros
    for c in s:
        seq.append(ord(c))
    return seq

x="Cadena ejemplo, con caracteres raros: $&%$!ƒ¥™¶§"
s=get_seq_int(x)
print(s)
```

Pearson hash (I)

```
# s: cadena de caracteres
# t: tabla de permutaciones
#    debe usarse SIEMPRE LA MISMA TABLA
def pearson_hash(s,t):
    h=0
    TSIZE=len(t)
    for c in s:
        index=(h^ord(c))%TSIZE
        h=t[index]
    return h
```

Pearson hash (II)

```
# s: secuencia de enteros
# t: tabla de permutaciones
#    debe usarse SIEMPRE LA MISMA TABLA
def pearson_hash(s,t):
    h=0
    TSIZE=len(t)
    for j in s:
        index=(h^j)%TSIZE
        h=t[index]
    return h
```

Jenkins hash (I)

```
# s: cadena de caracteres
# base: típicamente, base=2**k
def jenkins_hash(s,base=32):
    h=0
    for c in s:
        h=(h+ord(c))%base
        h=(h+(h<<10)%base)%base
        h^=(h>>6)
h=(h+(h<<3)%base)%base
h^=(h>>11)
h=(h+(h<<15)%base)%base
return h
```

Jenkins hash (II)

```
# s: secuencia de enteros
# base: típicamente, base=2**k
def jenkins_hash(s,base=32):
    h=0
    for j in s:
        h=(h+j)%base
        h=(h+(h<<10)%base)%base
        h^=(h>>6)
h=(h+(h<<3)%base)%base
h^=(h>>11)
h=(h+(h<<15)%base)%base
return h
```

Java hash

```
# típicamente, BASE = 2**k
def init_java(size):
    global BASE
    BASE=size

# s: cadena de caracteres
def java_hash(s):
    h=0
    for c in s:
        h=(31*h+ord(c))%BASE
    return h

init_java(256)
while True:
    print(java_hash(input("Cadena: ")))
```