FUNDAMENTOS DE COMPUTACIÓN
GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
Universidad de Cantabria

Práctica 12 (laboratorio)
19 de diciembre del 2018

Recursión

Un algoritmo es **iterativo** si utiliza **ciclos** o **bucles** para indicar la repetición de una serie de instrucciones.

Un algoritmo es **recursivo** cuando expresa una parte de la solución de un problema en términos de una llamada a sí mismo.

Ya hemos visto en clase algunos ejemplos clásicos de algoritmos recursivos (el factorial de un entero, el máximo común divisor o las Torres de Hanoi).

Todo algoritmo recursivo admite uno iterativo, por ejemplo el caso del factorial de un entero:

```
1  def  factorial_recursivo(n):
2    if  n == 1:
3      fact = 1
4    elif n > 1:
5      fact = n * factorial_recursivo(n-1)
6    return fact
```

```
1  def factorial_iterativo(n):
2    fact = 1
3    for i in range(2,n+1):
4      fact = fact * i
5    return fact
```

---

1. SUMAR

   Diseña una función recursiva `suma_recursiva` y otra iterativa `suma_iterativa` que tengan como argumento una lista de números y devuelva la suma de todos ellos. Por ejemplo, si la lista es [1,2,5.1 la suma de todos los elementos es 8.1.

2. PALÍNDROMOS

   Un palíndromo es una palabra que se lee igual de izquierda a derecha, que de derecha a izquierda. Por ejemplo: *anna*, *salas*, *radar*, etc. Diseña una función recursiva `palindromo_recursiva` y otra iterativa `palindromo_iterativa` que tengan como argumento una cadena y devuelva el booleano TRUE si es un palíndromo o FALSE en caso contrario.

3. INVERTIR

   Diseña una función recursiva `invertir_recursiva` y otra iterativa `invertir_iterativa` que tengan como argumento una cadena y devuelva la cadena invertida. Por ejemplo, si la cadena es "santander"que devuelva rednatnas".

4. NÚMERO DE DÍGITOS

   Diseña una función recursiva `digitos_recursivo` y otra iterativa `digitos_iterativo` que teniendo como argumentos dos enteros positivos n y b, devuelva la cantidad de dígitos de n en base b. Por ejemplo,

   $$25 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2 + 1, \qquad 82459 = 12 \times 19^3 + 0 \times 19^2 + 7 \times 19 + 18$$

   Luego si $n = 25$ y $b = 2$, el número de dígitos necesarios son 5. Y si $n = 82459$ y $b = 19$ debería devolver 4.

5. LA SUCESIÓN DE FIBONACCI

   Supongamos que partimos de una pareja de conejos recién nacidos, y queremos calcular cuántas parejas de conejos forman la familia al cabo de n meses si:

   1. Los conejos nunca mueren.
   2. Un conejo puede reproducirse al comienzo del tercer mes de vida.
   3. Los conejos siempre nacen en parejas macho-hembra. Al comienzo de cada mes, cada pareja macho-hembra, madura, se reproduce en solamente un par de conejos macho-hembra.

Para un n pequeño, por ejemplo 6, la solución se puede obtener fácilmente a mano:

- Mes 1: 1 pareja, la inicial.
- Mes 2: 1 pareja, ya que todavía no es sexualmente madura.
- Mes 3: 2 parejas; la original y una pareja de hijos suyos.
- Mes 4: 3 parejas; la original, una pareja de hijos suyos nacidos ahora y la pareja de hijos suyos nacidos en el mes anterior.
- Mes 5: 5 parejas; la original, una pareja de hijos suyos nacidos ahora, las dos parejas nacidas en los meses 3 y 4, y una pareja de hijos de la pareja nacida en el mes 3.
- Mes 6: 8 parejas; las 5 del mes anterior, una pareja de hijos de la original, una pareja de hijos de la nacida en el mes 3 y una pareja de hijos nacida en el mes 4.

Si deseamos saber el número de parejas F_n al cabo de n meses, para un n cualquiera, podemos construir un algoritmo recursivo:

$$F_n = \begin{cases} 1, & \text{si} \quad 1 \leq n \leq 2; \\ F_{n-1} + F_{n-2}, & \text{si} \quad n > 2. \end{cases}$$

En esta fórmula recursiva F_{n-1} son las parejas vivas en el mes $n-1$, y F_{n-2} son las parejas que nacen en el mes n a partir de las que había en el mes $n-2$. La sucesión infinita F_n es conocida como la de **Fibonacci**:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...$$

la cual modela muchos fenómenos naturales:

- Aparecen por doquier en la vida real, en informática. La revista científica: *The Fibonacci Quarterly.*
- El árbol genealógico de las abejas es : 1 zángano tiene 1 madre, 2 abuelos, 3 bisabuelos, 5 tatarabuelos, 8 tatara-tatarabuelos, 13 tatara-tatara-tatarabuelos. . .
- También, la siguen muchos pétalos de flores: la flor del iris tiene 3, la de la rosa silvestre 5, la del dephinium 8, la de la cineraria 13, la de la achicoria 21. Y así sucesivamente (las hay con 34, 55 y 89 pétalos).

Se pide: realizar un programa iterativo `fibonacci_iterativo` y otro recursivo `fibonacci_recursivo` que dado un número n, encuentre el número F_n.

- Ejecuta el siguiente programa y compara con los del apartado anterior.

```
def fibonacci_formula(n):
   z = 1/5**0.5*( ((1+5**0.5)/2)**n - ((1-5**0.5)/2)**n)
   return int(z)
```

6. LA SUCESIÓN LA CUEVA DE LA LOCA

La siguiente sucesión infinita L_n es conocida como La cueva de la loca (descubierta el día 20/12/2017 a las 9:25):

$$-1, 0, 1, -2, 0, 11, -13, 229, 4560, ...$$

La sucesión inicia con $L_1 = -1, L_2 = 0, L_3 = 1$, y cada elemento siguiente $L_n, n > 3$ se obtiene con la fórmula:

$$L_n = L_{n-1} + 2L_{n-2}^2 + 3L_{n-3}^3$$

Por ejemplo,

$$L_4 = L_3 + 2L_2^2 + 3L_1^3 = 1 + 2 \times 0^2 + 3 \times (-1)^3 = 1 + 0 - 3 = -2.$$

Realizar un programa iterativo y otro recursivo que dado un número n, calcule el número de La cueva de la loca L_n.

7. BUSQUEDA BINARIA

Muchos problemas de localizar elementos (números, nombres, caracteres, etc,) en una estructura de datos (listas, conjuntos, arrays, etc.) se pueden plantear como sigue:

Entrada: Un número a y una sucesión de n números $L = [a_1, a_2, \ldots, a_n]$.
Salida: El primer índice i tal que $a_i = a$, ó -1 si a no es un elemento de L.

- Dada una entrada como $(22, [2, -5, 22, 6, 0, -10, 2, 22, 3])$, un algoritmo de búsqueda debería retornar 3 .
- Dada una entrada como $(7, [2, -5, 22, 6, 0, -10, 2, 22, 3])$, un algoritmo de búsqueda debería retornar -1 .

Un posible algoritmo consiste en buscar el elemento comparándolo secuencialmente con cada elemento de la sucesión hasta encontrarlo, o hasta que se llegue al final. La existencia se puede asegurar cuando el elemento es localizado, pero no podemos asegurar la no existencia hasta no haber comparado todos los elementos.

Supongamos, ahora que la sucesión(lista) está ordenada e ilustramos la estrategia con el siguiente ejemplo:

Supongamos que tenemos que buscar el número de teléfono de una persona con la ayuda de una guía telefónica. Abrimos la guía por la mitad y mirando el primer nombre de la página, inmediatamente sabemos en que mitad de la guía está el número. Podemos hacer el mismo procedimiento con esa parte y, así sucesivamente. Con solamente comprobar dos nombres, hemos eliminado $\frac{3}{4}$ de los números de la guía.

Esta técnica se denomina búsqueda binaria, porque en cada etapa divide la lista (=la guía de teléfonos) en dos partes iguales: los valores que están antes y los que están después del valor que hemos comprobado.

El siguiente programa proporciona un método iterativo:

```
def  busqueda_binaria_iterativo(a, L):
  menor = 0
  grande = len(L)-1
  while menor <= grande:
    mitad = (menor + grande)//2
    if L[mitad] > a:
      grande = mitad -1
    elif L[mitad] < a:
      menor = mitad +1
    else:
      return mitad +1
  return -1
```

Se pide: realizar un programa recursivo: `busqueda_binaria_recursivo(a, L)`.