Recursión – II

Clase #24

IIC1103 – Introducción a la Programación

Marcos Sepúlveda (marcos@ing.puc.cl)

Veremos hoy …

- Retomar ideas sobre recursión
- Ejemplos de recursión
- Mergesort
- Quicksort

Recursión – recordemos

- Las soluciones recursiva involucran 2 partes:

1. Caso Base:
   - Problema es simple de resolver directamente

2. Caso Recursivo:
   a) Dividir el problema en una o más partes simples del problema.
   b) Llamar a la función (recursivamente) en cada parte.
   c) Combinar la solución de las partes en una solución completa del problema.

Recordemos ejemplo de Fibonacci con recursión

- F(0)=0
- F(1)=1
- F(n)=F(n-1)+F(n-2)

Recursivo

```
def Fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return Fibonacci(n-1) + \
               Fibonacci(n-2)
```

```
>>> for i in range(0,11):
      print(Fibonacci(i), end=',')
    print("")

0,1,1,2,3,5,8,13,21,34,55,
```

Obtención del máximo de una lista de números

Algoritmo:

- Caso base:
  - Si la lista tiene un único elemento, ese elemento es el **máximo**.

- Caso recursivo:
  - Si el valor en la posición inicial de la lista es mayor que el **máximo** del resto de la lista, entonces el valor de la primera posición es el **máximo** global.
  - Si el valor en la posición inicial de la lista es menor que el **máximo** del resto de la lista, entonces el **máximo** global será el **máximo** del resto de la lista.

Obtención del máximo de una lista de números

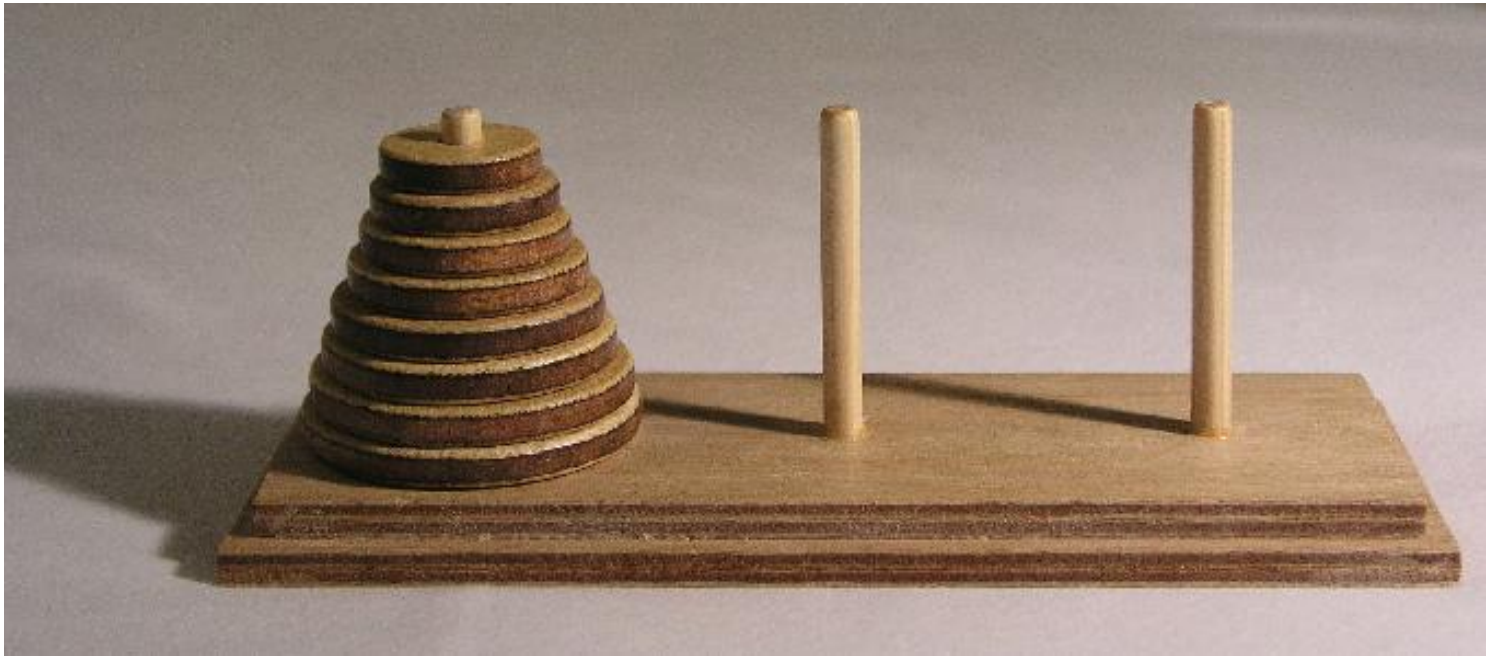
```
def Maximo(lista, inferior):
    if (inferior == len(lista)-1):
        mayor = lista[inferior]
    else:
        maxResto = Maximo(lista,inferior+1)
        if (lista[inferior] > maxResto):
            mayor = lista[inferior]
        else:
            mayor = maxResto
    return mayor

numeros = [20, 14, 28, 1, 47, 23, 8, 33, 19]

print("Lista:", numeros)
print("Máximo:", Maximo(numeros, 0))
```

Caso Base

Caso Recursivo

```
>>>
Lista: [20, 14, 28, 1, 47, 23, 8, 33, 19]
Máximo: 47
```

El problema de la Torre de Hanoi



http://www.dynamicdrive.com/dynamicindex12/towerhanoi.htm

El problema de la Torre de Hanoi

(originalmente "la torre de Brahma")

- Inventado por el matemático francés E. Lucas en los 1880s
  - "En el gran templo en Benares descansa una placa de bronce con tres agujas de diamantes; en una de ellas, Dios puso durante la creación 64 discos de oro, el más grande descansando sobre la placa y los otros, cada uno más pequeño, uno encima del otro. Esta es la torre de Brahma. Continuamente los sacerdotes transfieren los discos de una aguja a otra de acuerdo con las reglas inmutables de Brahma: no puede moverse más que un disco a la vez y ese disco debe ponerse en una aguja de modo que no quede encima de un disco más pequeño.
    Cuando los 64 discos hayan sido transferidos a una de las otras agujas de la torre, el templo y los brahmanes se desmoronarán, y con un estruendo, el mundo desaparecerá."

  - El problema es determinar el orden en que los sacerdotes deben transferir los discos para hacer desaparecer el mundo.

El problema de la Torre de Hanoi – pasos para su solución

- Análisis:
  - Llamemos ‘A’, ‘B’ y ‘C’ a las agujas; queremos mover los 64 discos que están en ‘A’ a ‘B’, ayudándonos con ‘C’.
  - No hay input; o tal vez el input es sólo el número 64, o tal vez los números 64 y 3, pero ambos números no cambian.
  - El output es la secuencia de movimientos de discos, de a uno, de una aguja a otra, hasta que todos haya sido transferidos de la aguja “A” a la “B” (¿cuántos movimientos en total?)
- Diseño:
  - Para mover los 64 discos de ‘A” a ‘B’, es absolutamente necesario que en algún instante los 63 discos más pequeños estén en ‘C’; sólo en ese momento podemos mover el disco más grande de ‘A’ a ‘B’; y después hay que mover los 63 discos de ‘C’ a ‘B’.
  - Similarmente, para mover los 63 discos de ‘A’ a ‘C’, es absolutamente necesario mover primero los 62 discos más pequeños de ‘A’ a ‘B’; luego movemos el disco #63 de ‘A’ a ‘C’; y por último movemos los 62 discos de ‘B’ a ‘C’.

El problema de la Torre de Hanoi – solución recursiva

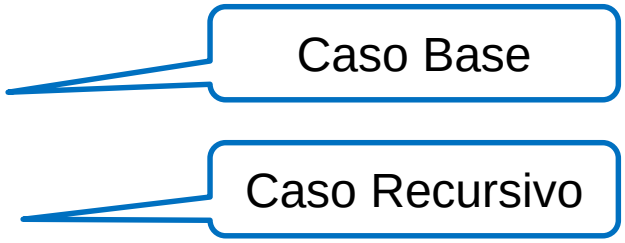
- El diseño anterior representa una estrategia recursiva:
  - Para solucionar el problema más grande (mover **n** discos), lo descomponemos en problemas más pequeños del mismo tipo.
  - Primero movemos **n–1** discos, luego movemos **1** disco, y finalmente movemos **n–1** discos nuevamente.
  - Por supuesto, para mover **n–1** discos hacemos los mismo: primero movemos **n–2**, luego movemos **1**, y luego movemos **n–2** de nuevo.

- El algoritmo se puede expresar recursivamente:
  - `Mover(n, A, B, C) =`
    - `Si n = 1, "mover de A a B"`
    - `De lo contrario:`
      - `Mover(n-1, A, C, B)`
      - `"mover de A a B"`
      - `Mover(n-1, C, B, A)`

El problema de la Torre de Hanoi – solución recursiva

- En Python

```
def Mover(n, ini, fin, tmp):
    if n == 1:
        print(ini,"->",fin)
    else:
        Mover(n-1, ini, tmp, fin)
        Mover(1, ini, fin, tmp)
        Mover(n-1, tmp, fin, ini)

Mover(3, 'A', 'B', 'C')
```




```
>>>
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
```

Otros ejercicios sobre recursión

- Verificar que un string es palíndromo

- Encontrar el máximo común divisor
  - mcd(n1,n2) =
    - Si n1<n2
      - mcd(n2,n1)
    - Si n1/n2 no tiene residuo
      - n2
    - Si no
      - mcd(n2,n1 mod n2)

- **Mergesort, Quicksort**: algoritmos de ordenamiento

Ordenamiento – links de interés

- Animaciones de distintos algoritmos de ordenamiento
  - http://www.sorting-algorithms.com
  - https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

- Algoritmos de ordenamiento en Python
  - http://danishmujeeb.com/blog/2014/01/basic-sorting-algorithms-implemented-in-python

Mergesort

- El algoritmo Mergesort fue desarrollado en 1945.
- Permite ordenar en O(n logn)
- Divide la lista en 2, cada una de las cuales se ordena por separado; luego se mezclan, manteniendo el orden

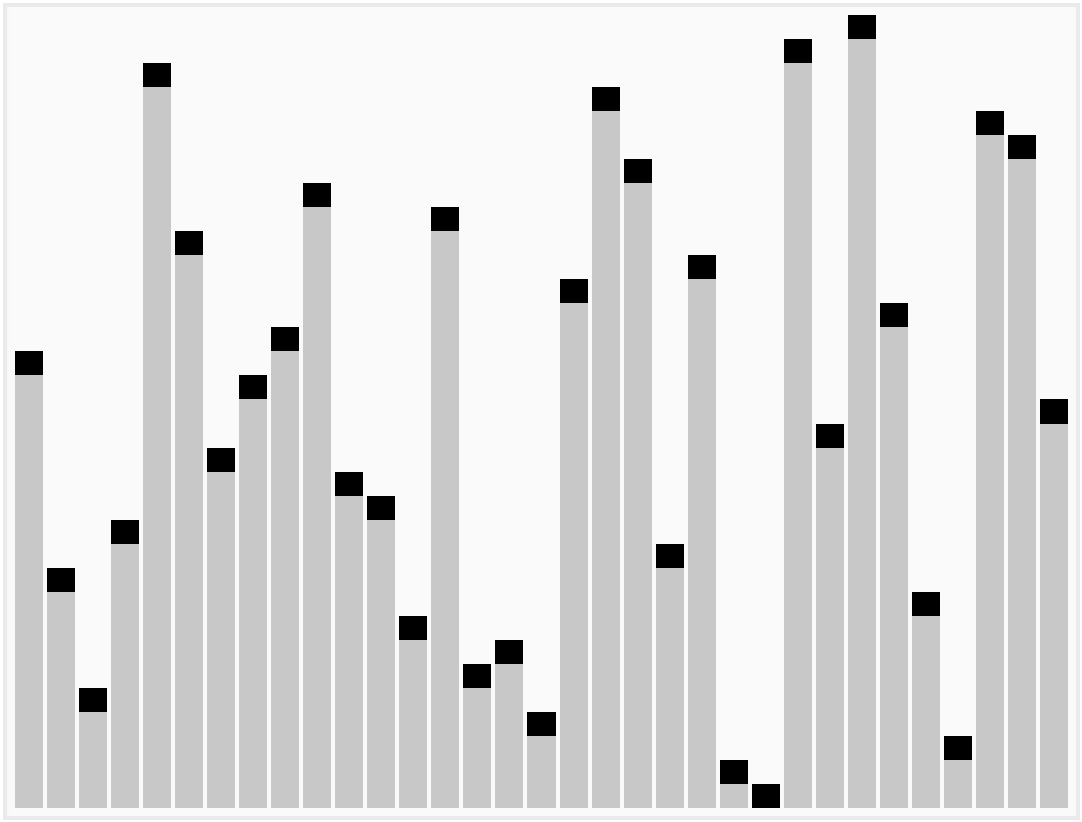
Fuente: http://en.wikipedia.org/wiki/File:Merge-sort-example-300px.gif

Mergesort – implementación en Python

```
def Mezclar(listaI, listaD):
    lista = []
    i = 0; d = 0

    while(i < len(listaI) and d < len(listaD)):
        if(listaI[i] < listaD[d]):
            lista.append(listaI[i]); i += 1
        else:
            lista.append(listaD[d]); d += 1

    #copiamos lo que aún quede en listaI o listaD
    lista.extend(listaI[i:])
    lista.extend(listaD[d:])

    return lista

def Mergesort(lista):
    if len(lista) > 1:
        mid = len(lista) // 2
        listaI = Mergesort(lista[:mid])
        listaD = Mergesort(lista[mid:])
        lista = Mezclar(listaI, listaD)

    return lista
```

Quicksort

- El algoritmo Quicksort fue desarrollado en 1960.
- Permite ordenar en O(n logn)



Fuente: http://en.wikipedia.org/wiki/File:Sorting_quicksort_anim.gif

Quicksort – algoritmo

1. Elegir un elemento **x** cualquiera del vector, entre **a** y **b**. El elemento **x** se llama **pivote**. Usualmente se elige el elemento que está en el medio.
2. Particionar el vector en dos subvectores: **vec[a ... p]** y **vec[p+1 ... b]**, intercambiando elementos de modo que todos los elementos que son menores que **x** estén a la izquierda y todos los mayores que **x** estén a la derecha.
3. Aplicar **Quicksort** al subvector **vec[a ... p]**
4. Aplicar **Quicksort** al subvector **vec[p+1 ... b]**

Quicksort – implementación en Python

```
def ParticionarVector(lista, inicio, fin):
    pivote = lista[(inicio + fin) // 2]
    izq = inicio
    der = fin
    while (izq < der):
        while (lista[izq] < pivote):
            izq += 1
        while (lista[der] > pivote):
            der -= 1
        if (izq < der):
            aux = lista[izq]
            lista[izq] = lista[der]
            lista[der] = aux
            izq += 1
            der -= 1
    return der


def Quicksort(lista, inicio, fin):
    if (inicio < fin):
        pos = ParticionarVector(lista, inicio, fin)
        Quicksort(lista, inicio, pos)
        Quicksort(lista, pos + 1, fin)
```

Algoritmos de ordenamiento a través de la danza

- Selection sort:
  - https://www.youtube.com/watch?v=Ns4TPTC8whw
- Insertion sort:
  - https://www.youtube.com/watch?v=ROalU379l3U
- Bubble sort:
  - https://www.youtube.com/watch?v=lyZQPjUT5B4
- Merge sort:
  - https://www.youtube.com/watch?v=XaqR3G_NVoo
- Quick sort:
  - https://www.youtube.com/watch?v=ywWBy6J5gz8