UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Tema 1. Recursividad

Programación (PRG)

Jorge González Mollá

Departamento de Sistemas Informáticos y Computación

Índice

Definición

- Dada la declaración de un array `a` de `num` elementos de tipo `tipoBase`:

  ```
  tipoBase[] a = new tipoBase[num];
  ```

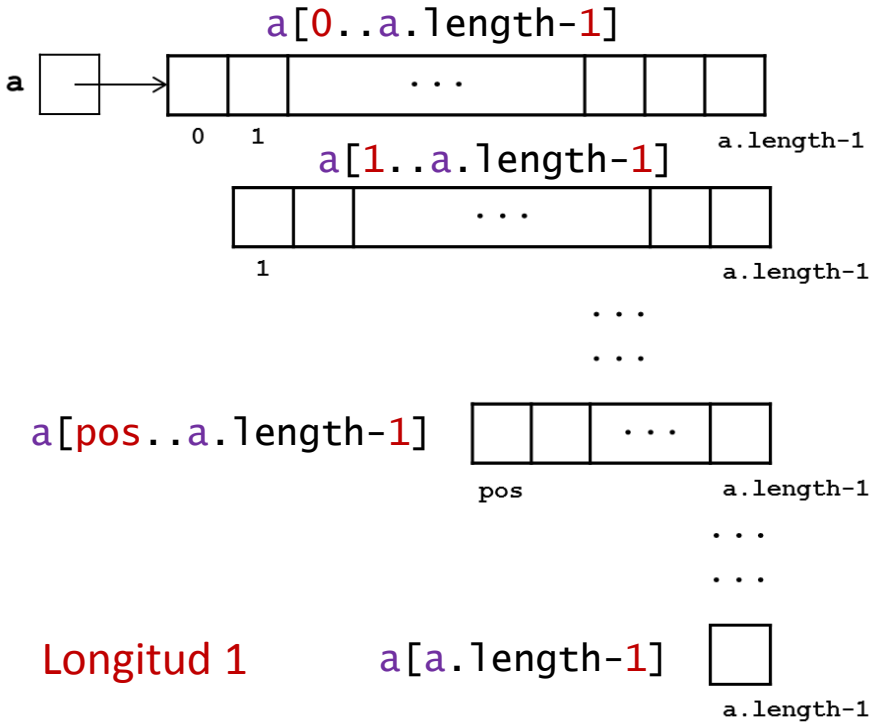
  se puede considerar que dicho array forma la siguiente secuencia:

  `a[0], a[1], a[2], …, a[a.length-2], a[a.length-1]`

  denotada como `a[0..a.length-1]`.
- Dicha secuencia se puede definir recursivamente como la secuencia formada por `a[0]` y el subarray derecho con el resto de elementos de `a`, es decir, `a[1..a.length-1]`.
- A su vez, el subarray `a[1..a.length-1]` puede definirse recursivamente del mismo modo, y así sucesivamente, el tamaño del subarray en cuestión va menguando hasta tener longitud 1, o incluso estar vacío, sin elementos. Es lo que se denomina descomposición recursiva ascendente.
- Análogamente, el array `a` también se puede definir de forma recursiva considerando por un lado el subarray de la izquierda `a[0..a.length-2]` y por otro, el último elemento del array, es decir, el dato `a[a.length-1]`. A esta otra técnica se le denomina descomposición recursiva descendente.

Descomposición recursiva

descomposición recursiva ascendente

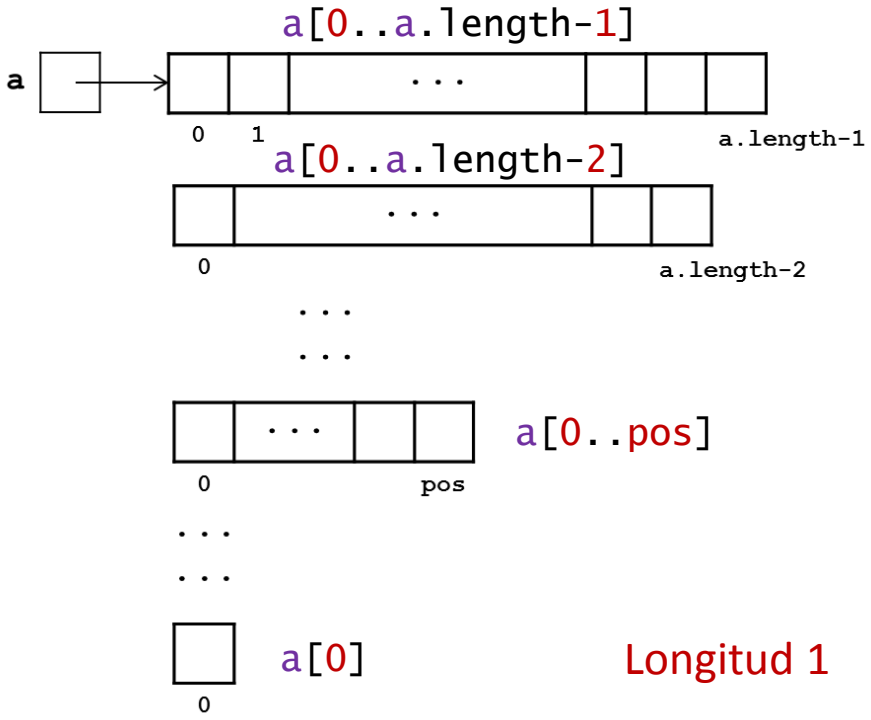


Longitud 1

Longitud 0

{}

descomposición recursiva descendente



Longitud 1

Longitud 0

{}

Implementación

- Los parámetros formales de un método recursivo que trabaje sobre un (sub)array `a` serán el propio (sub)array `a` y también las posiciones que delimitan el `inicio` y el `fin` del subarray (dentro del array `a`) a cuyo ámbito se refiere la aplicación de cada llamada.
- Además, habrá que determinar el tipo de descomposición recursiva:
  - Ascendente: el índice `inicio` se incrementa de 1 en 1 en cada llamada hasta que finalmente iguala o se cruza con el índice `fin` en el caso base:

    En la primera llamada, el índice `fin` se fija a `a.length-1`
    - Caso Base de Longitud 1
      `inicio == fin` siendo $0 \leq$ `inicio` $\leq$ `a.length-1`
    - Caso Base de Longitud 0
      `inicio > fin` siendo $0 \leq$ `inicio` $\leq$ `a.length`
  - Descendente: el índice `fin` se decrementa de 1 en 1 en cada llamada hasta que finalmente iguala o se cruza con el índice `inicio` en el caso base:

    En la primera llamada, el índice `inicio` se fija a 0
    - Caso Base de Longitud 1
      `inicio == fin` siendo $0 \leq$ `fin` $\leq$ `a.length-1`
    - Caso Base de Longitud 0
      `inicio > fin` siendo $-1 \leq$ `fin` $\leq$ `a.length-1`

Descomposición recursiva ascendente

[…] metodoAscendente(tipoBase[] a, int inicio, int fin)

- Llamada inicial: metodoAscendente(a, 0, a.length-1), o sea, inicio se instancia a 0 y fin se instancia a a.length-1.
- Asumimos un Caso Base de Longitud 0:

```
/** 0 <= inicio <= a.length                Caso Base de Longitud 0
  * fin == a.length-1 */
[…] metodoAscendente(tipoBase[] a, int inicio, int fin) {
    if (inicio>fin) { // Condición de Caso Base de Longitud 0
       // resolver el caso base
       // si procede, devolver el resultado con return
    }
    else {
       // operar localmente sobre a[inicio]
       // llamar a metodoAscendente(a, inicio+1, fin)
       // si procede, devolver el resultado con return
    }
}
```

Descomposición recursiva descendente

```
[…] metodoDescendente(tipoBase[] a, int inicio, int fin)
```

- Llamada inicial: `metodoDescendente(a, 0, a.length-1)`, o sea, `inicio` se instancia a `0` y `fin` se instancia a `a.length-1`.
- Asumimos un Caso Base de Longitud 0:

```
/** inicio == 0
  * -1 <= fin <= a.length-1            Caso Base de Longitud 0 */
[…] metodoDescendente(tipoBase[] a, int inicio, int fin) {
    if (inicio>fin) { // Condición de Caso Base de Longitud 0
       // resolver el caso base
       // si procede, devolver el resultado con return
    }
    else {
       // operar localmente sobre a[fin]
       // llamar a metodoDescendente(a, inicio, fin-1)
       // si procede, devolver el resultado con return
    }
}
```