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

Recorrido ascendente

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

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

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

Recorrido descendente

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

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

```
/** inicio == 0
  * -1 <= fin <= a.length-1            Caso Base de Longitud 0 */
[…] recorridoDescendente(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 recorridoDescendente(a, inicio, fin-1)
       // si procede, devolver el resultado con return
    }
}
```

Método lanzadera

- Se suele definir un método público homónimo, llamado guía o lanzadera, el cual hace la llamada inicial al método recursivo sobre todo el array `a`. Esta técnica permite ocultar la estructura recursiva asociada a los parámetros `inicio`/`fin` del perfil del método recursivo correspondiente, el cual ahora lógicamente se puede (y se debería) definir como `private`.

```
public […] recorridoAscendente(tipoBase[] a) {
        // llamar a recorridoAscendente(a, 0, a.length-1)
        // si procede, devolver el resultado con return
}

public […] recorridoDescendente(tipoBase[] a) {
        // llamar a recorridoDescendente(a, 0, a.length-1)
        // si procede, devolver el resultado con return
}
```

Recorrido ascendente simplificado

- Sustituir las referencias al parámetro `fin` por su valor inicial `a.length-1`:

```
[…] recorridoAscendenteSimplificado(tipoBase[] a, int inicio)
```

- Llamada inicial: `inicio` se instancia a `0`

  `recorridoAscendenteSimplificado(a, 0)`:

- Asumimos un Caso Base de Longitud 0:

```
/** 0 <= inicio <= a.length          Caso Base de Longitud 0 */
[…] recorridoAscendenteSimplificado(tipoBase[] a, int inicio) {
    if (inicio >= a.length) { // Condición CB de Longitud 0
       // tratar array sin elementos: C. Base de Longitud 0
       // si procede, devolver el resultado con return
    }
    else {
       // operar localmente sobre a[inicio]
       // llamar a recorridoAscendenteSimplificado(a, inicio+1)
       // si procede, devolver el resultado con return
    }
}
```

Recorrido descendente simplificado

- Sustituir las referencias al parámetro `inicio` por su valor inicial `0`:

```
[…] recorridoDescendenteSimplificado(tipoBase[] a, int fin)
```

- Llamada inicial: `fin` se instancia a `a.length-1`

```
recorridoDescendenteSimplificado(a, a.length-1)
```

- Asumimos un Caso Base de Longitud 0:

```
/** -1 <= fin <= a.length-1          Caso Base de Longitud 0 */
[…] recorridoDescendenteSimplificado(tipoBase[] a, int fin) {
    if (fin < 0) {      // Condición Caso Base de Longitud 0
       // tratar array sin elementos: C. Base de Longitud 0
       // si procede, devolver el resultado con return
    }
    else {
       // operar localmente sobre a[fin]
       // llamar a recorridoDescendenteSimplificado(a, fin-1)
       // si procede, devolver el resultado con return
    }
}
```

Ejemplo: Sumar el contenido de un array (ascendente)

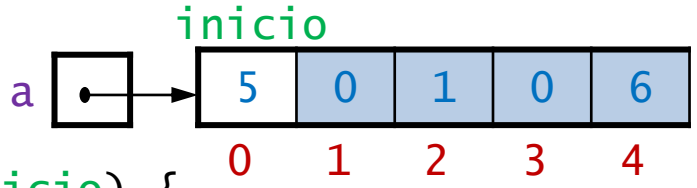
1. Sumar los elementos de un array de enteros `a[0..a.length-1]`

Recorrido ascendente simplificado

```
/** 0 <= inicio <= a.length */
public static int sumaA(int[] a, int inicio)
```

2. Caso Base de Longitud 0 (`inicio >= a.length`): El resultado es `0`
   Caso General. En otro caso, el resultado será:
   `a[inicio]` más la sumaA de `a[inicio+1..a.length-1]`

   Llamada inicial: `sumaA(a, 0)`




```
/** 0 <= inicio <= a.length */
public static int sumaA(int[] a, int inicio) {
        if (inicio >= a.length) return 0;
        else return a[inicio] + sumaA(a, inicio+1);
}
```

3. En el caso general, en cada llamada el parámetro `inicio` crece de 1 en 1; eventualmente llegará a valer `a.length`, alcanzando por tanto el caso base y finalizando así el algoritmo. Para todas las llamadas generadas, el valor de `inicio` siempre cumple la precondición `0 <= inicio <= a.length`.

Ejemplo: Sumar el contenido de un array (descendente)

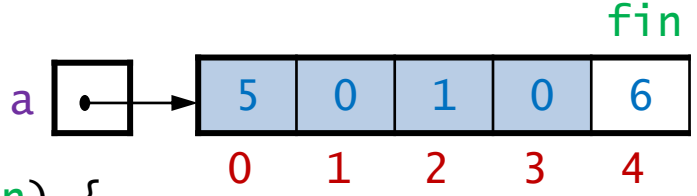
1. Sumar los elementos de un array de enteros `a[0..a.length-1]`

Recorrido descendente simplificado

```
/** -1 <= fin <= a.length-1 */
public static int sumaD(int[] a, int fin)
```

2. Caso Base de Longitud 0 (`fin < 0`): El resultado es `0`
   Caso General. En otro caso, el resultado será `a[fin]` más la `sumaD` de `a[0..fin-1]`

   Llamada inicial: `sumaD(a, a.length-1)`




```
/** -1 <= fin <= a.length-1 */
public static int sumaD(int[] a, int fin) {
        if (fin < 0) return 0;
        else return a[fin] + sumaD(a, fin-1);
}
```

3. En el caso general, en cada llamada el parámetro `fin` se decrementa en 1; eventualmente éste llegará a ser negativo, alcanzando por tanto el caso base y finalizando así el algoritmo. Para todas las llamadas generadas, el valor de `fin` siempre permanece dentro de rango `-1 <= fin <= a.length-1`.

Ejemplo: Contar ocurrencias en un array (ascendente)

1. Contar las apariciones de x (int) en un array de enteros a[0..a.length-1]

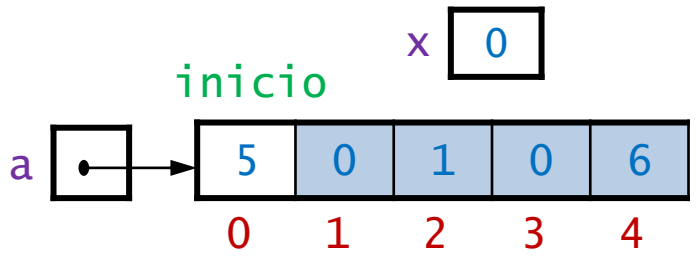
Recorrido ascendente simplificado

```
/** 0 <= inicio <= a.length */
public static int contarA(int[] a, int x, int inicio)
```

2. Caso Base de Longitud 0 (inicio >= a.length): El resultado es 0
Caso General: En otro caso, el resultado es ver el número de apariciones de x en el subarray derecho (contarA aplicado en a[inicio+1..a.length-1]), y sumarle 1 si resulta que en a[inicio] también se encuentra el valor de x.

Llamada inicial: contarA(a, x, 0)

```
/** 0 <= inicio <= a.length */
public static int contarA(int[] a, int x, int inicio) {
   if (inicio >= a.length) return 0;
   else {
      int cont = contarA(a, x, inicio+1);
      if (a[pos]!=x) return cont;
      else return cont + 1;
   }
}
```



3. Similar a la validación del recorrido ascendente anterior.

Ejemplo: Contar ocurrencias en un array (descendente)

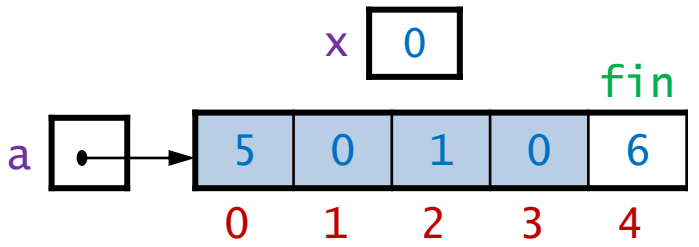
1. Contar las apariciones de `x` (int) en un array de enteros `a[0..a.length-1]`

Recorrido descendente simplificado

```
/** -1 <= fin <= a.length-1 */
public static int contarD(int[] a, int x, int fin)
```

2. Caso Base de Longitud 0 (`fin < 0`): El resultado es `0`
   Caso General: En otro caso, el resultado es ver el número de apariciones de `x` en el subarray izquierdo (llamar a `contarD` aplicándola sobre `a[0..fin-1]`), y sumarle `1` si resulta que en `a[inicio]` también se encuentra el valor de `x`.
   Llamada inicial: `contarD(a, x, a.length-1)`

```
/** -1 <= fin < a.length */
public static int contarD(int[] a, int x, int fin) {
   if (fin < 0) return 0;
   else { int cont = contarD(a, x, fin-1);
          if (a[fin]!=x) return cont;
          else return 1 + cont;
        }
}
```



3. Similar a la validación del recorrido descendente anterior.