Algoritmos y Estructuras de Datos:

Introducción a la Recursión de Programas

Guillermo Román Díez

groman@fi.upm.es

Universidad Politécnica de Madrid

2021/2022

Métodos Recursivos

Pregunta

¿Qué es un método recursivo?

Métodos Recursivos

Pregunta

¿Qué es un método recursivo?

Método recursivo

"*Un método recursivo es un método que se invoca a sí mismo*"

Métodos Recursivos

Pregunta

¿Qué es un método recursivo?

Método recursivo

"*Un método recursivo es un método que se invoca a sí mismo*"

- El ejemplo típico es el *factorial*(n)
  - *factorial*(n) se calcula $n * (n-1) * (n-2) * (n-3) * \ldots * 1$
  - Observad que la expresión $(n-1) * (n-2) * (n-3) \ldots * 1$ es el *factorial*$(n-1)$

```
n * (n-1) * (n-2) * ... * 1
    \_____________________/
          factorial(n-1)
\_________________________/
        factorial(n)
```

```
factorial(n) = | 1                        si n=1
               | n * factorial(n-1)       si n>1
```

Recursión: Factorial

Ejercicio

Escribir una versión iterativa y otra recursiva de `factorial(n)`

Recursión: Factorial

Ejercicio

Escribir una versión iterativa y otra recursiva de `factorial(n)`

Versión Iterativa

```
int factorial(int n) {
  int r = 1;
  while (n > 1) {
    r = n * r;
    n--;
  }
  return r;
}
```

Versión Recursiva

```
int factorial(int n){
 if (n <= 1)
  return 1;
 else
  return n*factorial(n-1);
}
```

- Cualquier programa iterativo tiene uno equivalente recursivo y viceversa
- Llamarse recursivamente es una forma de iterar

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|    5 * factorial(4)
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|    5 * factorial(4)
|    |    4 * factorial(3)
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|    5 * factorial(4)
|    |    4 * factorial(3)
|    |    |    3 * factorial(2)
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
|   |   |   |   2
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
|   |   |   |   2
|   |   |   2
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
|   |   |   |   2
|   |   |   2
|   |   6
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
|   |   |   |   2
|   |   |   2
|   |   6
|   24
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
|   |   |   |   2
|   |   |   2
|   |   6
|   24
120
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
|   |   |   |   2
|   |   |   2
|   |   6
|   24
120
```

Ejemplo de Ejecución

Ejercicio

"Dibujar" la ejecución recursiva de `factorial(5)`

```
factorial(5)
|   5 * factorial(4)
|   |   4 * factorial(3)
|   |   |   3 * factorial(2)
|   |   |   |   2 * factorial(1)
|   |   |   |   |   1
|   |   |   |   2
|   |   |   2
|   |   6
|   24
120
```

- Vemos que las llamadas recursivas se van **apilando**
- Una de ellas llega al **caso base** y ya no se hacen nuevas llamadas
- Entonces se van resolviendo las llamadas pendientes hasta llegar a la primera llamada que se produjo

Recursión vs. Iteración

- La recursión se puede entender como una forma de "iteración"
- Todo programa iterativo se puede escribir de forma recursiva

Recursión

```
int factorial(int n){
 if (n <= 1)
  return 1;
 else
  return n*factorial(n-1);
}
```

Recursión vs. Iteración

- La recursión se puede entender como una forma de "iteración"
- Todo programa iterativo se puede escribir de forma recursiva

Recursión

```
int factorial(int n){
 if (n <= 1)
  return 1;
 else
  return n*factorial(n-1);
}
```

Iteración

```
int factorial(int n){
 int res = 1;
 while (n > 1) {
   res = res * n;
   n--;
 }
 return res;
}
```

Recursión de Cola

- La recursión puede implementar utilizando el *return* de las diferentes llamadas apiladas, pero...
- También se puede implementar a estilo **paso de continuaciones** usando **recursión de cola**
- Para esto pasamos un parámetro extra sobre el que vamos acumulando el resultado
- La llamada recursiva sería el último mandato del programa

Recursión "normal"

```
int factorial(int n){
 if (n <= 1)
  return 1;
 else
  return n*factorial(n-1);
}
```

Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)
  return r;
 else
  return factorial(n*r,n-1);
}
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
| | | factorial(1*5*4*3*2,1)
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
| | | factorial(1*5*4*3*2,1)
| | | | 120
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
| | | factorial(1*5*4*3*2,1)
| | | | 120
| | | 120
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
| | | factorial(1*5*4*3*2,1)
| | | | 120
| | | 120
| | 120
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
| | | factorial(1*5*4*3*2,1)
| | | | 120
| | | 120
| | 120
| 120
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
| | | factorial(1*5*4*3*2,1)
| | | | 120
| | | 120
| | 120
| 120
120
```

Ejemplo Recursión de Cola

```
int factorial(int r,int n){
 if (n <= 1)     return r;
 else            return factorial(n*r,n-1);
}

factorial(1,5)
factorial(1*5,4)
| factorial(1*5*4,3)
| | factorial(1*5*4*3,2)
| | | factorial(1*5*4*3*2,1)
| | | | 120
| | | 120
| | 120
| 120
120
120
```

Errores Comunes

- La función debe tener uno o más casos base (no recursivos)
  - Si no hay caso base `StackOverflow`!!

```
public int factorial(int n) {
   return n * factorial(n-1);    // no hay caso base
}
```

- Debe haber un orden bien fundado en los valores de los argumentos para acercarnos al caso base
- Las llamadas recursivas eventualmente llegan a alguno de los casos
  - Si no llega al caso base o se lo saltan: `StackOverflow`!!

```
public int factorial(int n) {
  if (n == 1) return 1;
  return n*factorial(n-2); // no siempre llega al caso base
}
```

Ejemplo: Máximo Común Divisor

Ejercicio

Método que calcula el Máximo Común Divisor (Euclides)

Versión Iterativa

```
int mcd(int n, int m) {
  while (m != 0) {
    int tmp = m;
    m       = n % m;
    n       = tmp;
  }
  return n;
}
```

Ejemplo: Máximo Común Divisor

Ejercicio

Método que calcula el Máximo Común Divisor (Euclides)

Versión Iterativa

```
int mcd(int n, int m) {
  while (m != 0) {
    int tmp = m;
    m       = n % m;
    n       = tmp;
  }
  return n;
}
```

Versión Recursiva

```
int mcd(int n, int m) {
  if (m == 0)
    return n;
  else
    return mcd(m, n % m);
}
```

Ejemplo: Fibonacci

Ejercicio

Función que calcule el n-ésimo elemento de la sucesión de Fibonacci

$$\begin{aligned} \text{fib}(0) &= 0 \\ \text{fib}(1) &= 1 \\ \text{fib}(\text{N}) &= fib(N-1) + fib(N-2) \end{aligned}$$

- Fibonacci tiene 2 casos base diferentes
- Fibonacci tiene 2 llamadas recursivas

Ejemplo: Fibonacci

Ejercicio

Función que calcule el n-ésimo elemento de la sucesión de Fibonacci

$$\begin{aligned} \text{fib}(0) &= 0 \\ \text{fib}(1) &= 1 \\ \text{fib}(N) &= fib(N-1) + fib(N-2) \end{aligned}$$

- Fibonacci tiene 2 casos base diferentes
- Fibonacci tiene 2 llamadas recursivas

```
public static int fib(int n) {
  if (n == 0)  return 0;
  if (n == 1) return 1;    /* dos casos base */
  return fib(n-1) + fib(n-2);
}
```

Método show sobre un array

```
public static void show (int [] arr) {
```

Pregunta

¿cómo podemos saber la posición del array por la que vamos?

Método show sobre un array

```
public static void show (int [] arr) {
```

Pregunta

¿cómo podemos saber la posición del array por la que vamos?

- Es necesario un método auxiliar recursivo que incluya un parámetro extra

Método show sobre un array

```
public static void show (int [] arr) {
```

Pregunta

¿cómo podemos saber la posición del array por la que vamos?

- Es necesario un método auxiliar recursivo que incluya un parámetro extra

```
public static void show (int [] arr) {
  showRec(arr,0);
}

public static void showRec (int [] arr, int i) {
  if (i >= arr.length) { return; }
  System.out.println(arr[i]);
  showRec(arr, i+1);
}
```

Búsqueda Lineal

```
public static boolean member(int elem, int [] arr) {
  if (arr == null || arr.length == 0)
    return false;

  return memberRec(elem, arr, 0);
}

private static boolean memberRec(int elem, int [] arr,
                                 int pos) {
  if (pos >= arr.length)     return false;    /* caso base */
  if (elem == arr[pos])      return true;     /* caso base */

  return memberRec(elem,arr,pos+1);
}
```

Búsqueda binaria

Ejercicio

Implementar de forma recursiva el algoritmo de búsqueda binaria sobre un vector ordenado

```
public boolean memberBin(int elem, int [] arr) {
  if (arr == null   || arr.length == 0 ||
      elem < arr[0] || elem > arr[arr.length - 1])
    return false;
  int start = 0;
  int end   = arr.length - 1;
  int m = (start+end)/2;
  while (start<=end && arr[m] != elem) {
    if (elem < arr[m])
      end   = m-1;
    else
      start = m+1;
    m = (start+end)/2;
  }
  return start <= end;
}
```

Búsqueda Binaria: Recursiva

- Es necesario un método auxiliar que lleve la cuenta del rango en el que estamos buscando [start–end]

Búsqueda Binaria: Recursiva

- Es necesario un método auxiliar que lleve la cuenta del rango en el que estamos buscando [start–end]

```
public static boolean memberBin(int elem, int [] arr) {
  if (arr == null    || arr.length == 0 ||
      elem < arr[0] || elem > arr[arr.length-1])
    return false;
  else
    return memberRec(elem, arr, 0, arr.length-1);
}

private static boolean memberBinRec(int elem, int [] arr,
                                    int start, int end) {
 if (start > end)     return false;    /* caso base */
 int m = (start + end) / 2;
 if (elem == arr[m]) return true;     /* caso base */
 if (elem < arr[m]) return memberBinRec(elem,arr,start,m-1);
                    return memberBinRec(elem,arr,m+1,end);
}
```

Recursión sobre Listas de Posiciones

- Las listas de posiciones se pueden definir de forma recursiva
  - Como una secuencia de nodos que, o bien es vacía, o está formada por un nodo seguido por una secuencia de nodos
- Podemos definir métodos recursivos para explotar esta definición

```
void show(PositionList<E> list) {
  if (list != null) showRec(list, list.first());
}

void showRec(PositionList<E> list,
             Position<E> cursor) {
  if (cursor != null) {
    System.out.println(cursor.element());
    showRec(list, list.next(cursor));
  }
}
```

Imprimir una lista al revés

Pregunta

¿y si queremos imprimir el contenido de la lista al revés?

Imprimir una lista al revés

Pregunta

¿y si queremos imprimir el contenido de la lista al revés?

```
void showRev(PositionList<E> list) {
  if (list != null) showRec(list, list.last());
}

void showRevRec(PositionList<E> list,
                Position<E> cursor) {
  if (cursor != null) {
    showRevRec(list, list.prev(cursor));
    System.out.println(cursor.element());
  }
}
```

`SumaElementos`

Ejercicio

Método que sume los elementos de una `PositionList` cuyos elementos pueden ser `null`

Ejercicio

Método que sume los elementos de una `PositionList` cuyos elementos pueden ser `null`

```
int sumaElems(PositionList<Integer> list) {
  if (list == null || list.isEmpty()) return 0;
  else return sumaRec(list, list.first());
}
int sumaRec(PositionList<Integer> list,
            Position<Integer> cursor){
  if (cursor == null)
    return 0;
  else if (cursor.element() == null)
    return sumaRec(list,list.next(cursor));
  else
    return cursor.element() +
           sumaRec(list,list.next(cursor));
}
```

Ejercicio

Ahora la suma "mejorada"

```
int sumaElems(PositionList<Integer> list) {
  if (list == null || list.isEmpty()) return 0;
  else return sumaRec(list, list.first());
}
int sumaRec(PositionList<Integer> list,
            Position<Integer> cursor){
  if (cursor == null)
    return 0;
  else
    return (cursor.element()==null ?
                          0 : cursor.element())) +
           sumaRec(list,list.next(cursor));
}
```

Ejercicio

Ahora la suma con recursión de cola

```
int sumaElems(PositionList<Integer> list) {
  if (list == null || list.isEmpty()) return 0;
  else return sumaRec(0, list, list.first());
}
int sumaRec( int r, PositionList<Integer> list,
             Position<Integer> cursor) {
 if (cursor == null)
  return r;
 else if (cursor.element() == null)
  return sumaRec(r, list, list.next(cursor));
 else
  return sumaRec(r+cursor.element(),
                 list,
                 list.next(cursor));
}
```

Ejercicio

Ahora la suma con recursión de cola "mejorado"

```
int sumaElems(PositionList<Integer> list) {
  if (list == null || list.isEmpty()) return 0;
  else return sumaRec(0, list, list.first());
}

int sumaRec( int r,
             PositionList<Integer> list,
             Position<Integer> cursor) {
  if (cursor != null) {
    r = sumaRec( r + (cursor.element() == null ? 0
                                  : cursor.element()),
                 list,
                 list.next(cursor))
  }
  return r;
}
```

Ejercicio

Método que borra el primer elemento distinto de null

```
void borraElemento(PositionList<E> list, E elem) {
  if (list != null)
    borraRec(list, elem, list.first());
}

void borraRec( PositionList<E> list, E elem,
               Position<E> cursor) {
  if (cursor != null) {
    if (eqNull(cursor.element(), elem))
      list.remove(cursor);
    else
      borraRec(list, elem, list.next(cursor));
  }
}
```

Ejercicio

Método que borra TODOS los elementos iguales a elem

```
void borraTodos(PositionList<E> list, E elem) {
  if (list != null)
    borraTodosRec(list, elem, list.first());
}

void borraTodosRec( PositionList<E> list, E elem,
                    Position<E> cursor) {
  if (cursor != null) {
    Position<E> aux = cursor;
    cursor = list.next(cursor);
    if (eqNull(aux.element(), elem)) {
      list.remove(aux);
    }
    borraRec(list, elem, cursor);
  }
}
```

Creación de listas de forma recursiva

- Hay 2 opciones para crear una lista nueva de forma recursiva
  - Opción 1: Recibir una lista ya creada por parámetro e ir añadiendo los elementos en cada llamada recursiva
  - Opción 2: Crear la lista en el caso base e ir pasándola "hacia arriba" en el *return* el método
    - ★ Ojo, que esto implica que la lista se construye empezando en la última llamada recursiva

Creación de listas de forma recursiva (opción 1)

Ejercicio

Método que copia una lista usando un parámetro

```
PositionList<E> copiarParam(PositionList<E> list) {
  if (list == null) {return null;}
  PositionList<E> res = new NodePositionList<E>();
  copiarParamRec(list, list.first(), res);
  return res;
}
void copiarParamRec(PositionList<E> list,
                    Position<E> cursor,
                    PositionList<E> res) {
  if (cursor != null) {
    res.addLast(cursor.element());
    copiarParamRec(list, list.next(cursor), res);
  }
}
```

Creación de listas de forma recursiva (opción 2)

Ejercicio

Método que copia una lista mediante el *return* del método

```
PositionList<E> copiar(PositionList<E> list) {
  if (list == null) {
    return null;
  }
  return cpRec(list, list.first());
}
PositionList<E> cpRec(PositionList<E> list,
                      Position<E> cursor) {
  if (cursor == null) {
    return new NodePositionList<E>();
  }
  PositionList<E> res = cpRec(list,list.next(cursor));
  res.addFirst(cursor.element());
  return res;
}
```