

Fundamentos de programación

Recursión generativa

carlos.andres.delgado@correounivalle.edu.co

Carlos Andrés Delgado S.

Facultad de Ingeniería. Universidad del Valle

Noviembre de 2016

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Recursión
estructural

Recursión
generativa

1 Recursión estructural

2 Recursión generativa

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Recursión
estructural

Recursión
generativa

1 Recursión estructural

2 Recursión generativa

Definición

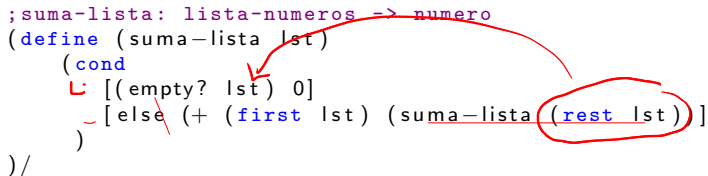
Recursión es definir algo en términos de sí mismo.

- Una función recursiva tiene un llamado a sí misma
- Un dato es recursivo, si es compuesto por si mismo.
Ejemplo: Árboles

Definición

Funciones recursivas:

```
; suma-lista: lista-numeros -> numero
(define (suma-lista lst)
  (cond
    [(empty? lst) 0]
    [else (+ (first lst) (suma-lista (rest lst)))]
  )
)/
```



La recursión tiene dos partes: condición de parada y llamado recursivo.

Definición

- 1 Las recursiones que hemos trabajado tienen condición de parada (caso base) y caso recursivo
- 2 El caso base, se diseña la respuesta cuando el dato es lo más simple posible, ejemplo: la suma una lista vacía es 0
- 3 El caso recursivo se llama un a la misma función con una modificación de los datos (resto de la lista)

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Recursión
estructural

Recursión
generativa

1 Recursión estructural

2 Recursión generativa

Definición

- 1 En algunos casos, es más práctico partir un problema en partes que permitan una solución **divide y vencerás dicen**
- 2 Cuando la entrada, se parte sin tener en cuenta la estructura del tipo de dato y los llamados recursivos se aplican a estas partes, entonces se llama generativa.

Definición

El enfoque de la recursión generativa, es dividir un problema en otros más pequeños. Se obtiene la solución general combinando las soluciones de problemas más pequeños:

- 1 Problemas triviales: que no requieren partirse, ya que su solución es sencilla
- 2 Problemas no triviales: deben partirse en problemas más pequeños para ser solucionados

Quicksort

Un buen ejemplo de recursión generativa, es el ordenamiento con el algoritmo QuickSort

- 1 Este utiliza la estrategia divide y vencerás, el problema se divide en dos problemas más pequeños y así sucesivamente hasta llegar a soluciones triviales
- 2 Es un método de ordenamiento muy eficiente (su estudio será ya en FADA :)).

[4 5 6 1 2 3 9 8]

Algoritmo QuickSort

¿Como funciona?

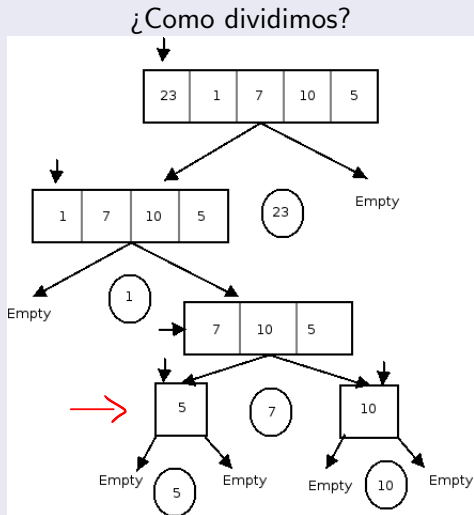
- 1 Escogemos un pivote (es un elemento cualquiera en la lista)
- 2 Se divide en dos listas:
 - 1 La primera lista, contiene los elementos menores o iguales que el pivote
 - 2 La segunda lista, contiene los elementos mayores que el pivote
- 3 Se sigue ordenando, hasta que se tienen vacías (Solución trivial) dividiendo
- 4 Finalmente, se combinan las soluciones

Algoritmo QuickSort

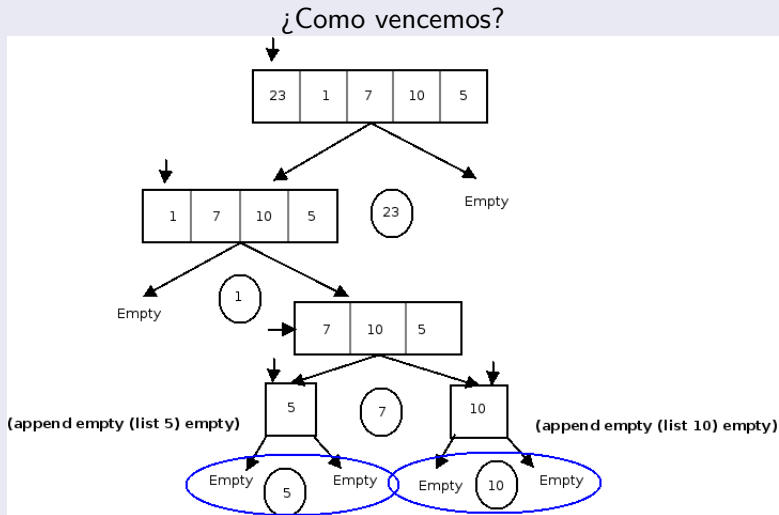
Un ejemplo, ordenemos la lista (list 23 1 7 10)

- 1 Escogemos el primer elemento de la lista, es decir 23. (Se puede escoger cualquier otro)
- 2 Se generan dos listas, (list 1 7 10) y empty
- 3 Estas dos listas son nuestro nuevo problema de ordenamiento.
- 4 El llamado recursivo incluye, la composición de el resultado del llamado con (list 1 70 10), el pivote que es 23 y el llamado con empty.

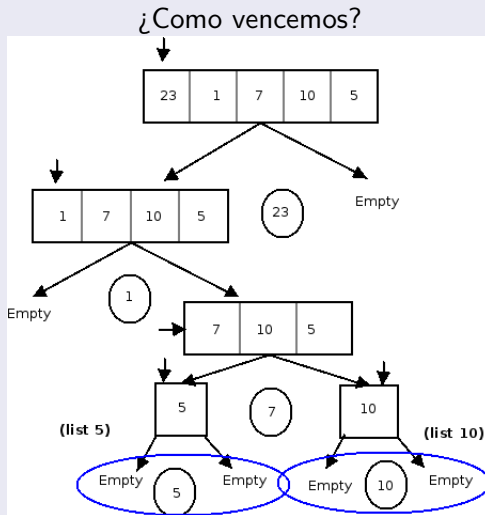
Algoritmo QuickSort



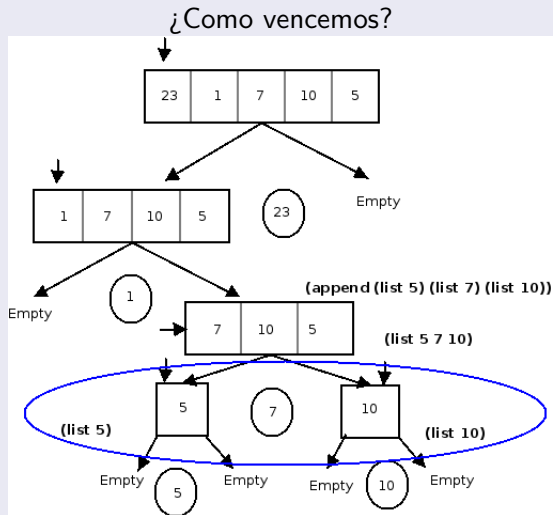
Algoritmo QuickSort



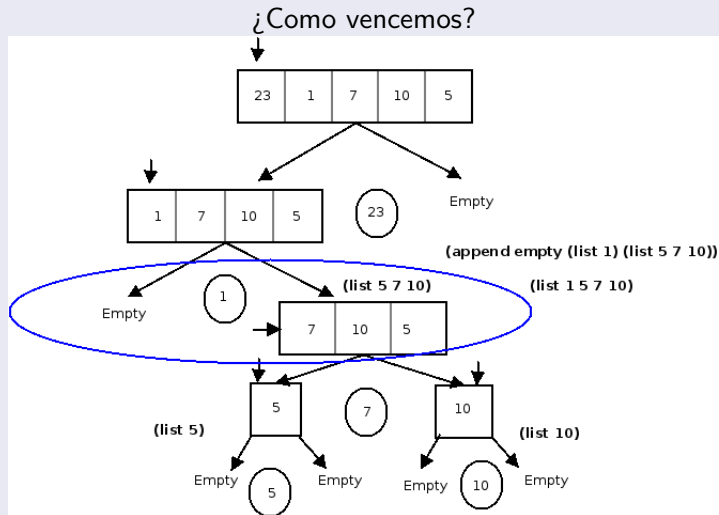
Algoritmo QuickSort



Algoritmo QuickSort

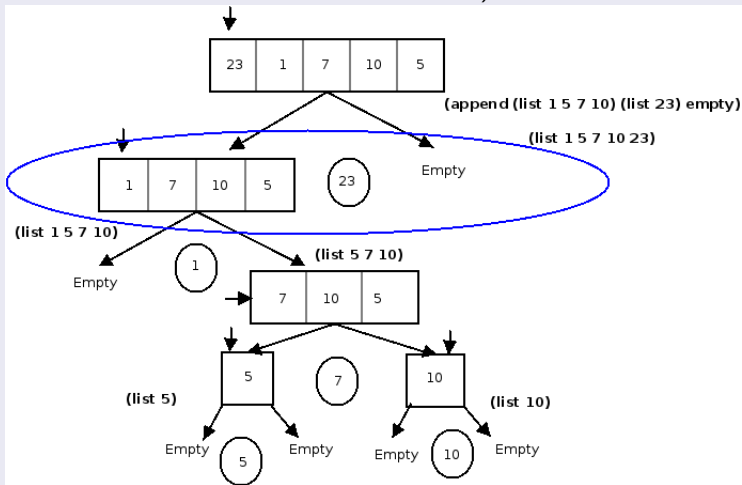


Algoritmo QuickSort



Algoritmo QuickSort

Y hemos vencido :)



Algoritmo QuickSort

Práctiquemos un poco, ordenemos bajo el enfoque las siguientes listas:

1 (list 9 0 12 3 4 -3) ←

2 (list 19 23 5 6 7)

3 (list -4 10 -3 5 8)

Escoja como pivote el primer número.

- 1) Una función que construya la salida (append)
- 2) Una función que construya la lista de menores
- 3) Una función que construya la lista de mayores

Algoritmo QuickSort

¿Como diseñemos la solución?

- 1 ¿Cual es la solución trivial? ¿Que hacemos en ese caso?
- 2 Y el caso recursivo. ¿Como partimos el problema? ¿Como combinamos las soluciones?

Escoja como pivote el primer número.

Algoritmo QuickSort

¿Como diseñemos la solución?

- 1 La solución trivial, es el caso de la lista vacía, ya que esta está ordenada por defecto.
- 2 El caso recursivo, se construye identificando el pivote y construyendo las listas de números menores e igual y la lista de mayores.
- 3 La combinación de soluciones, conserva el orden de pegar la lista del llamado con los elementos menores o iguales, el pivote y el llamado con los elementos mayores.

Escoja como pivote el primer número.

Algoritmo QuickSort

¿Como diseñemos la solución?

```
;contrato: ordenar-quicksort: lista-numeros ->  
          lista-numeros  
;(define (ordenar-quicksort lst)
```

Algoritmo QuickSort

¿Cual es el caso trivial? ¿Que hacemos con el?

Algoritmo QuickSort

¿Como diseñemos la solución?

```
;contrato: ordenar-quicksort: lista-numeros ->  
  lista-numeros  
(define (ordenar-quicksort lst)  
  (cond  
    [(empty? lst) empty]  
    [else ...])
```

Algoritmo QuickSort

¿Y el caso recursivo?

Algoritmo QuickSort

Debemos identificar lo siguiente:

- 1 Se divide en dos listas (mayores-e-iguales) y (menores).
Pista: Filtros.
- 2 Debemos pegar las salidas con algo (función append)
- 3 Hacer dos llamados recursivos, uno para la lista (mayores-e-iguales) y otro para lista (menores), **estos son nuestros subproblemas**

Algoritmo QuickSort

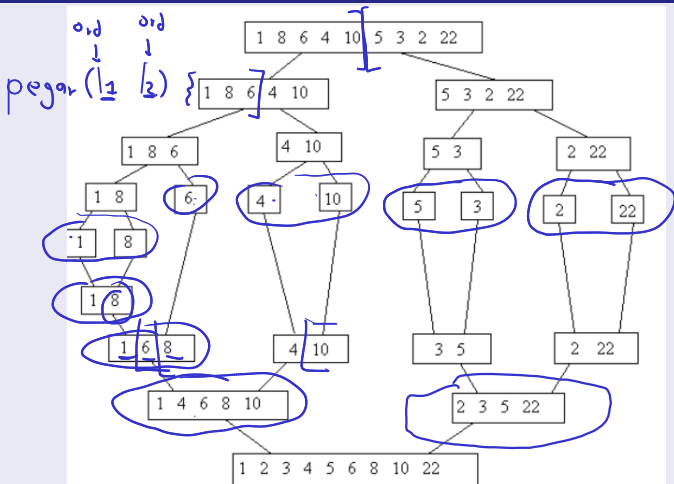
Observemos el ejemplo del campus.

MergeSort

Otro ejemplo, es el algoritmo MergeSort

- 1 Al igual que QuickSort, utiliza divide y vencerás
- 2 Se calcula la mitad, en caso de listas de tamaño impar aplique función techo.
- 3 Se parte la lista en dos, una que va desde el inicio hasta la mitad y otra que va hasta el final
- 4 Se parten las listas hasta que se tienen listas de tamaño uno
- 5 A diferencia de QuickSort, se realiza ordenamiento cada vez que se unen las listas
- 6 El caso trivial, es cuando se tiene un elemento.

MergeSort





list(1 2 3 4 5 7 8)

list(~~1~~ 5 10 12) list((2) 3 15 21)

pegar (l1 l2)

(< (first l1) (first l2))

~~T~~ ~~F~~

(cons (first l1)

(pegar rest l1) l2))

(cons (first l2)

(pegar l1 (rest l2))

MergeSort

Práctiquemos un poco, ordenemos bajo el enfoque las siguientes listas:

- 1 (list 9 0 12 3 4 -3)
- 2 (list 19 23 5 6 7)
- 3 (list -4 10 -3 5 8)



Universidad
del Valle

¿Preguntas?

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Recursión
estructural

Recursión
generativa

VAMO A

