

INSTITUTO POLITÉCNICO NACIONAL
Centro de Investigación en Computación



*Taller de programación
en Common Lisp*

5.1

Árboles...



Dr. Salvador Godoy Calderón

Dos formas...

Para implementar árboles, existen por lo menos, dos mecanismos: representar los árboles con listas anidadas, o bien, representarlos con registros (o cualquier otra estructura como arreglos)

Evidentemente, la representación con listas no es óptima en cuanto a tiempo de ejecución, pero es didáctica y sirve como un buen primer ejercicio...

Dr. Salvador Godoy Calderón



Primer paso...

El primer paso es decidir la interpretación de las listas y de su anidamiento...

La primera forma que viene a la mente, por su simplicidad aparente es:

Representar cada *subárbol* como una lista propia en la que el primer elemento es la raíz y los restantes elementos son sus subárboles descendientes...

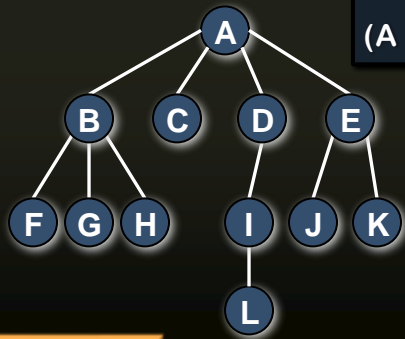
Representación "natural"...

La representación que primero viene a la mente es:

(<raíz> <subárbol#1> <subárbol#2> ...)

Por ejemplo, el árbol:

Se representaría:

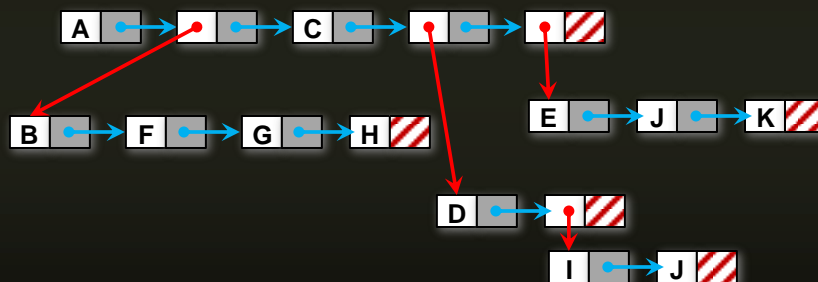
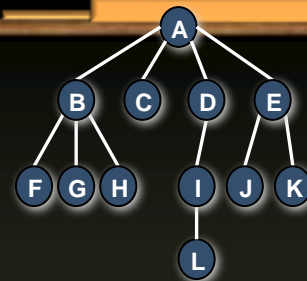


(A (B F G H) C (D (I L)) (E J K))

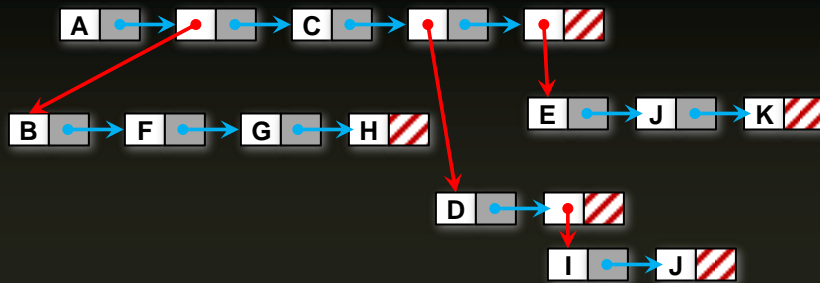
Dr. Salvador Godoy Calderón

Representación interna...

¿Cómo se representa internamente
(A (B F G H) C (D (I L)) (E J K)) ?



Dr. Salvador Godoy Calderón

Inconsistencia...

Esa representación es *inconsistente*, porque sólo en los elementos “raíz” tenemos la certeza del elemento representado, en los otros casos puede estar directamente ahí o apuntar a otra lista...

Dr. Salvador Godoy Calderón

Inconsistencia (2)...

Además, se están almacenando átomos en la parte **CAR** de las celdas, pero ¿y si los elementos en los nodos fueran de tipo lista?

```

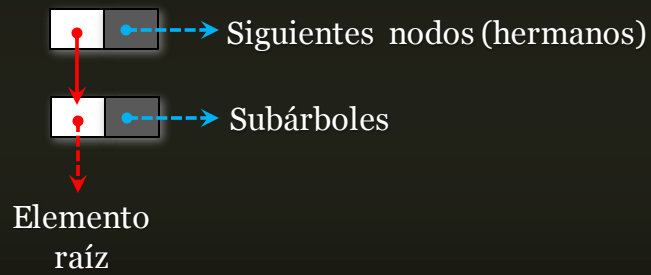
>> (cons '(a b c) 'd)
((A B C) . D)

>> (cons '(a b c) '(d e))
((A B C) D E)
  
```

Dr. Salvador Godoy Calderón

Alternativa...

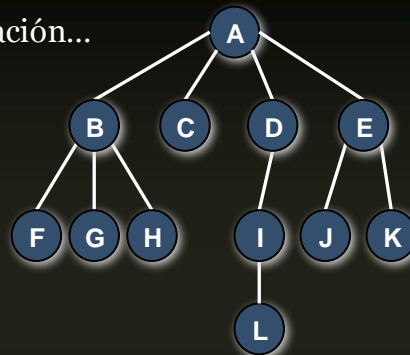
Para eliminar la inconsistencia, se puede usar una representación distinta de los nodos:



Dr. Salvador Godoy Calderón

Efecto...

Usando la nueva representación...

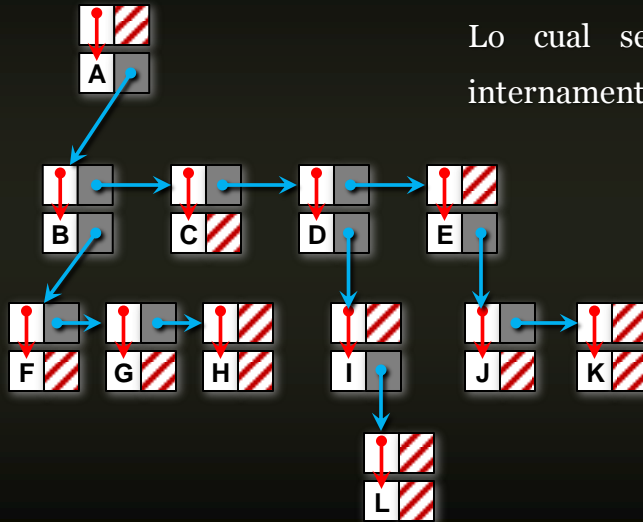


((A (B (F)(G)(H)) (C) (D (I (L))) (E (J) (K))))

Dr. Salvador Godoy Calderón

Efecto...

((A (B (F)(G)(H)) (C) (D (I (L))) (E (J) (K))))



Lo cual se representa internamente así:

Dr. Salvador Godoy Calderón

Código de acceso...

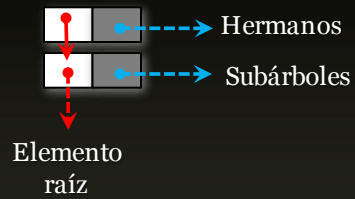
Ahora ya es posible escribir funciones de acceso...

Comenzando por definir un paquete nuevo donde estarán todas las funciones para manejo de árboles...

```
(defpackage :árboles
  (:use :common-lisp)
  (:export :dato-en
           :crea-nodo
           :primer-hijo
           :siguiente-hijo
           :agrega-hijo
           :lee-árbol))

(in-package :árboles)
```

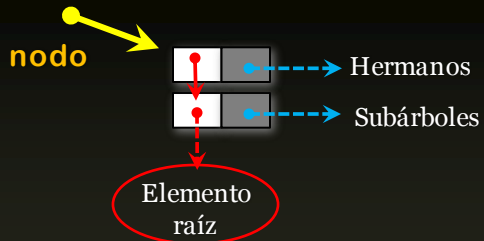
Dr. Salvador Godoy Calderón

El paquete...

Para crear un nodo se deben crear dos celdas de construcción...

```
(defun crea-nodo (dato)
  "Crea un nodo que contiene a dato"
  (cons (cons dato nil) nil))
```

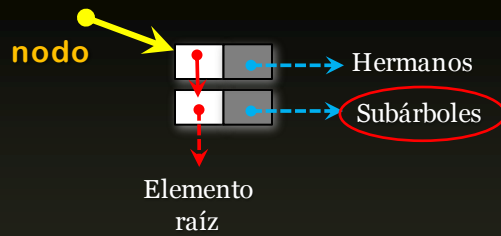
Dr. Salvador Godoy Calderón

El paquete...

```
(defun dato-en (nodo)
  "Regresa el elemento en un nodo"
  (first (first nodo)))
```

Dr. Salvador Godoy Calderón

El paquete...

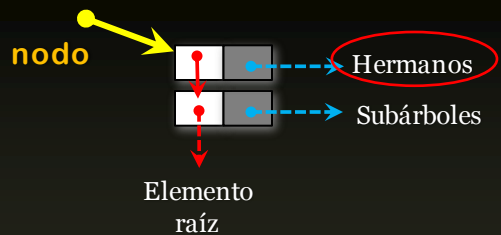


```
(defun dato-en (nodo)
  "Regresa el elemento en un nodo"
  (first (first nodo)))

(defun primer-hijo (nodo)
  "Regresa el primer hijo de un nodo"
  (rest (first nodo)))
```

Dr. Salvador Godoy Calderón

El paquete...



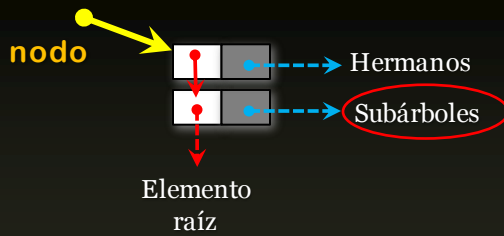
```
(defun dato-en (nodo)
  "Regresa el elemento en un nodo"
  (first (first nodo)))

(defun primer-hijo (nodo)
  "Regresa el primer hijo de un nodo"
  (rest (first nodo)))

(defun siguiente-hijo (nodo)
  "Regresa el siguiente hijo de un nodo"
  (rest nodo))
```

Dr. Salvador Godoy Calderón

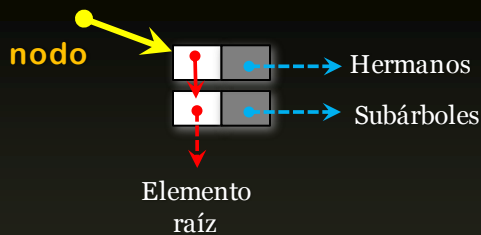
El paquete...



```
(defun agrega-hijo (nodo nodohijo)
  "Conecta nodohijo como hijo de nodo"
  (cond
    ((null nodo) nil)
    ((null nodohijo) nodo)
    (T (nconc (first nodo) nodohijo)
        nodo)))
```

Dr. Salvador Godoy Calderón

El paquete...



```
(defun lee-árbol (nodo &optional (márgen 0))
  "Recorre e imprime un árbol completo"
  (when (not (null nodo))
    (format t "~&~v@T Dato: ~A" márgen (dato-en nodo))
    (when (not (null (primer-hijo nodo)))
      (format t " Hijos: ~A"
        (maplist #'(lambda (x) (dato-en x))
                  (primer-hijo nodo)))
      (lee-árbol (primer-hijo nodo) (+ márgen 3))) )
```

Dr. Salvador Godoy Calderón

Uso...

Una vez creado el paquete `:árboles`, para usarlo, se requiere cargarlo y usar el paquete...

```
(load "árboles.lisp")
(use-package :árboles)
```

```
(defparameter *raíz* (crea-árbol 'A))
```

Gracias al `use-package` no es necesario escribir

```
(defparameter *raíz* (árboles:crea-árbol 'A))
```

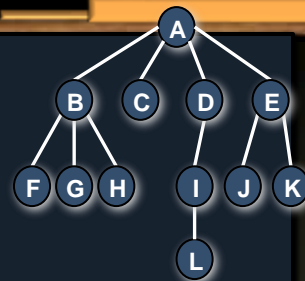
Dr. Salvador Godoy Calderón

```
(defun crea-árbol ()
  (let ((aux nil))
    (agrega-hijo *raíz* (crea-nodo 'B))
    (agrega-hijo *raíz* (crea-nodo 'C))
    (agrega-hijo *raíz* (crea-nodo 'D))
    (agrega-hijo *raíz* (crea-nodo 'E))

    (setq aux (first-child *raíz*))
    (agrega-hijo aux (crea-nodo 'F))
    (agrega-hijo aux (crea-nodo 'G))
    (agrega-hijo aux (crea-nodo 'H))

    (setq aux (siguiente-hijo (siguiente-hijo (primer-hijo *raíz*))))
    (agrega-hijo aux (crea-nodo 'I))
    (agrega-hijo (primer-hijo aux) (crea-nodo 'J))

    (setq aux (siguiente-hijo aux))
    (agrega-hijo aux (crea-nodo 'K))
    (agrega-hijo aux (crea-nodo 'L'))))
```

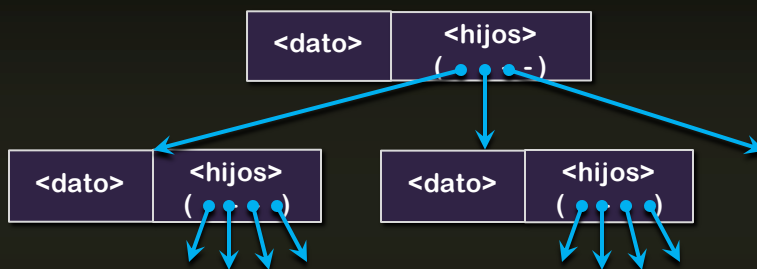


Dr. Salvador Godoy Calderón

Procesamiento con registros...

Registros...

Una forma “eficiente” de representar árboles es usando registros y recorridos recursivos:



```
>> (defstruct nodo dato hijos)
NODO
```


make-nodo, nodo-p, nodo-dato, nodo-hijos

Globalmente...

Declaramos global el inicio del árbol...

```
(defparameter *raíz* (make-nodo :dato 'A :hijos NIL))
```

```
(defun agrega-hijo (nodo elem)
  (if (null nodo)
      nil
      (let ((liga (find elem (nodo-hijos nodo) :key #'nodo-dato)))
        (cond
         (liga liga)
         (t (push (make-nodo :dato elem :hijos nil)
                   (nodo-ligas nodo)))))))
```



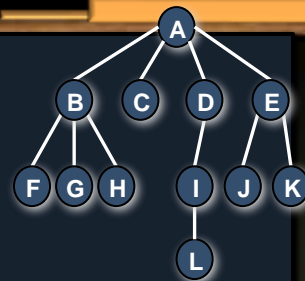
Dr. Salvador Godoy Calderón

```
(defun crea-árbol-con-registros ()
  (let ((aux nil))
    (agrega-hijo *raíz* 'B)
    (agrega-hijo *raíz* 'C)
    (agrega-hijo *raíz* 'D)
    (agrega-hijo *raíz* 'E)

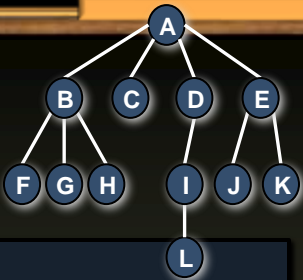
    (setq aux (first (nodo-hijos *raíz*)))
    (agrega-hijo aux 'F)
    (agrega-hijo aux 'G)
    (agrega-hijo aux 'H)

    (setq aux (third (nodo-hijos *raíz*)))
    (agrega-hijo aux 'I)
    (agrega-hijo (first (nodo-hijos aux)) 'J)

    (setq aux (fourth (nodo-hijos *raíz*)))
    (agrega-hijo aux 'K)
    (agrega-hijo aux 'L)))
```



Dr. Salvador Godoy Calderón



```
>> (construye-árbol)

#S(NODO
:DATO A
:HIJOS #S(NODO :DATO E :HIJOS (#S(NODO :DATO K :HIJOS NIL))
      #S(NODO :DATO L :HIJOS NIL)))
      #S(NODO :DATO D :HIJOS (#S(NODO :DATO I :HIJOS
      #S(NODO :DATO J :HIJOS NIL))
      #S(NODO :DATO C :HIJOS NIL)
      #S(NODO :DATO B :HIJOS (#S(NODO :DATO H :HIJOS NIL)
      #S(NODO :DATO G :HIJOS NIL)
      #S(NODO :DATO F :HIJOS NIL))))))
```

Dr. Salvador Godoy Calderón



**Algunas
aclaraciones
Importantes...**

Taxonomía...

Los recorridos en árboles BINARIOS (*tree traversals*) se dividen en dos grandes grupos:

Recorridos por nivel:

- ✦ Nivelorden (*level order*)

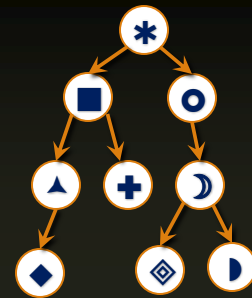
Recorridos a profundidad:

- ✦ Preorden (*preorder*)
- ✦ Inorden (*inorder*)
- ✦ Postorden (*postorder*)

Dr. Salvador Godoy Calderón

Especificación...

En árboles cuyos nodos tienen sólo dos descendientes: izquierdo y derecho, los órdenes de recorrido se definen recursivamente a partir de los siguientes patrones de visita:

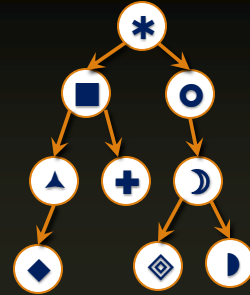


- ✦ nivelorden: *por niveles del árbol* * ■ ⊙ ▲ + ◐ ◑ ▼

Dr. Salvador Godoy Calderón

Especificación...

En árboles cuyos nodos tienen sólo dos descendientes: izquierdo y derecho, los órdenes de recorrido se definen recursivamente a partir de los siguientes patrones de visita:

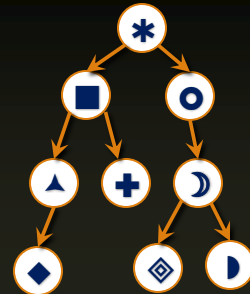


- ✦ nivelorden: *por niveles del árbol* * ■ ○ ▲ + ◐ ◑ ◒
- ✦ preorden: *raíz, izq, der* * ■ ▲ ◑ ◒ + ○ ◐ ◑

Dr. Salvador Godoy Calderón

Especificación...

En árboles cuyos nodos tienen sólo dos descendientes: izquierdo y derecho, los órdenes de recorrido se definen recursivamente a partir de los siguientes patrones de visita:

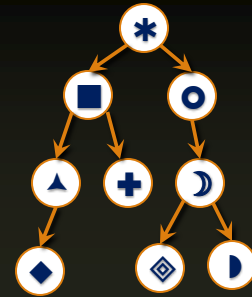


- ✦ nivelorden: *por niveles del árbol* * ■ ○ ▲ + ◐ ◑ ◒
- ✦ preorden: *raíz, izq, der* * ■ ▲ ◑ ◒ + ○ ◐ ◑
- ✦ inorden: *izq, raíz, der* ◑ ▲ ■ + * ◑ ◒ ◐ ◑

Dr. Salvador Godoy Calderón

Especificación...

En árboles cuyos nodos tienen sólo dos descendientes: izquierdo y derecho, los órdenes de recorrido se definen recursivamente a partir de los siguientes patrones de visita:



- ✦ nivelorden: *por niveles del árbol* * ■ ● ▲ + ☾ ◆ ◈ ◐
- ✦ preorden: *raíz, izq, der* * ■ ▲ ◆ + ● ☾ ◈ ◐
- ✦ inorden: *izq, raíz, der* ◆ ▲ ■ + * ◈ ☾ ◐ ●
- ✦ postorden: *izq, der, raíz* ◆ ▲ + ■ ◈ ◐ ☾ ● *

Dr. Salvador Godoy Calderón

Forma...

Todos estos órdenes se definen para árboles binarios y, con excepción del *inorden*, los demás pueden generalizarse para árboles no binarios...

La razón de ello es que, al tener más de dos descendientes, en un árbol no binario, no queda claro en qué momento visitar la raíz durante un recorrido en *inorden*...

En *preorden* y *postorden* se visitan todos los descendientes después (*preorden*) o antes (*postorden*) de visitar la raíz...

Dr. Salvador Godoy Calderón

En código...

Suponiendo el árbol representado con registros y con campos: *raíz*, *izq* y *der*...

```
(defun preorden (árbol)
  (cond
    ((null árbol) NIL)
    (T (print (árbol-raíz árbol))
        (print (preorden (árbol-izq árbol)))
        (print (preorden (árbol-der árbol))))))
PREORDEN
```

Dr. Salvador Godoy Calderón

Todos iguales...

```
(defun inorden (árbol)
  (cond
    ((null árbol) NIL)
    (T (print (inorden (árbol-izq árbol)))
        (print (árbol-raíz árbol))
        (print (inorden (árbol-der árbol))))))
INORDEN
```

```
(defun postorden (árbol)
  (cond
    ((null árbol) NIL)
    (T (print (postorden (árbol-izq árbol)))
        (print (postorden (árbol-der árbol)))
        (print (árbol-raíz árbol)))) )
POSTORDEN
```

Dr. Salvador Godoy Calderón

