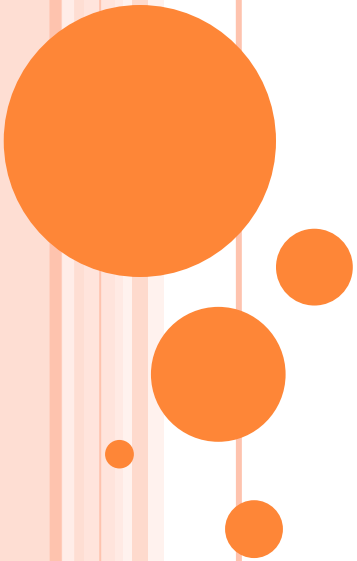


FUNDAMENTOS DE PROGRAMACIÓN

Datos recursivos II

Escuela de Ingeniería de Sistemas y Computación

Facultad de Ingeniería

Universidad del Valle

Primera parte:
Repaso de funciones con listas

Soluciones a algunos problemas de la tarea en clase

Problema 1: función que toma una lista como entrada y retorna un elemento.

- función elemento-n, que toma como entradas una lista y un natural n. La función retorna el n-esimo elemento en la lista.
- Ejemplo:
  - (elemento-n 2 (list 1 2 3))debe retornar 2
- Importante: la cuenta de los elementos arranca en 1

Análisis de datos:

- Si la lista está vacía:

  Puede ser por dos razones:

  - Se pasó una lista vacía como entrada
  - El índice es mayor que el tamaño de la lista

  En ambos casos hay un error en la entrada.
- Si n es igual a 1:

  Quiere decir que ya se encontró el elemento buscado y se retorna el primero de la lista
- En el caso contrario:

  Es necesario seguir buscando en el resto de la lista y con n-1 por medio de un llamado recursivo.

Problema 1: función que toma una lista como entrada y retorna una lista.

- función insertar-en-n que tiene como entradas: un elemento, una lista y un natural n. La función retorna una lista en la que se ha insertado el elemento en la n-esima posición.
- Ejemplo:

(insertar-en-n 5 (list 1 2 3 4) 2)

debe retornar (list 1 5 2 3 4)

- Importante:
  - la cuenta de los elementos arranca en 1

Análisis de datos:

- Si la lista está vacía:

  Puede ser por dos razones:
  - Se pasó una lista vacía como entrada
  - El índice es mayor que el tamaño de la lista

  En el primer caso no hay error, pero en el segundo si
- Si n es igual a 1:

  Quiere decir que ya se encontró la posición en la que se quiere ubicar el elemento, así que se hace cons del elemento y la lista
- En el caso contrario:

  Es necesario seguir contando posiciones, pero también conservando los elementos de la lista

Problema 1: función que toma una lista como entrada y retorna una lista.

- función insertar-ultimo que tiene como entradas: un elemento y una lista. La función retorna una lista en la que se ha insertado el elemento en la última posición.
- Ejemplo:
  - (insertar-ultimo 3 (list 2 1)) debe retornar (list 2 1 3)
- La clave en este ejercicio es encontrar el fin de la lista (empty) para poder insertar el elemento

Análisis de datos:

- Si la lista está vacía:

  Puede ser por dos razones:
  - Se pasó una lista vacía como entrada
  - Se recorrió la lista y llegamos a empty

  En ambos casos hacemos (cons elemento lista)

- En el caso contrario:

  Es necesario seguir buscando el fin de la lista haciendo un llamado recursivo con el resto.

Segunda parte: Datos recursivos

Estructuras en estructuras y Listas en listas, ejemplos: árboles

Listas con estructuras

- Definición de datos:

```
(define-struct toy (nom pre))
; La estructura toy repesenta
  un juguete
; nom es el nombre, de tipo
  símbolo
; pre es el precio, de tipo
  número
```

Un ejemplo de lista de juguetes :

```
(list
(make-toy ‘car 20)
(make-toy ‘doll 15)
(make-toy ‘xbox 300)
(make-toy ‘ipod 150)
)
```

(list

‘car	20
‘doll	15
‘xbox	300
‘ipod	150

)

Ejemplo de funciones con listas que contienen estructuras

- función quitar-caro que saca de una lista los juguetes que son considerados caros. La función tiene dos entradas, una lista de juguetes y un límite.

- Importante: esta función retorna una lista, por tanto tiene como operación principal cons porque la irá construyendo en cada paso

Análisis de datos:

- Si la lista está vacía:

  Puede ser por dos razones:

  - Se pasó una lista vacía como entrada
  - Se llegó al final de la lista

  Dado que esta función retorna una lista se debe retornar empty

- En el caso contrario:
  - Si el precio del juguete es menor o igual al límite: se incluye en juguete en la lista y se sigue comparando con los elementos del resto de la lista por medio de un llamado recursivo.
  - Si es mayor: no se incluye en la lista y se sigue comparando con los elementos del resto de la lista por medio de un llamado recursivo.

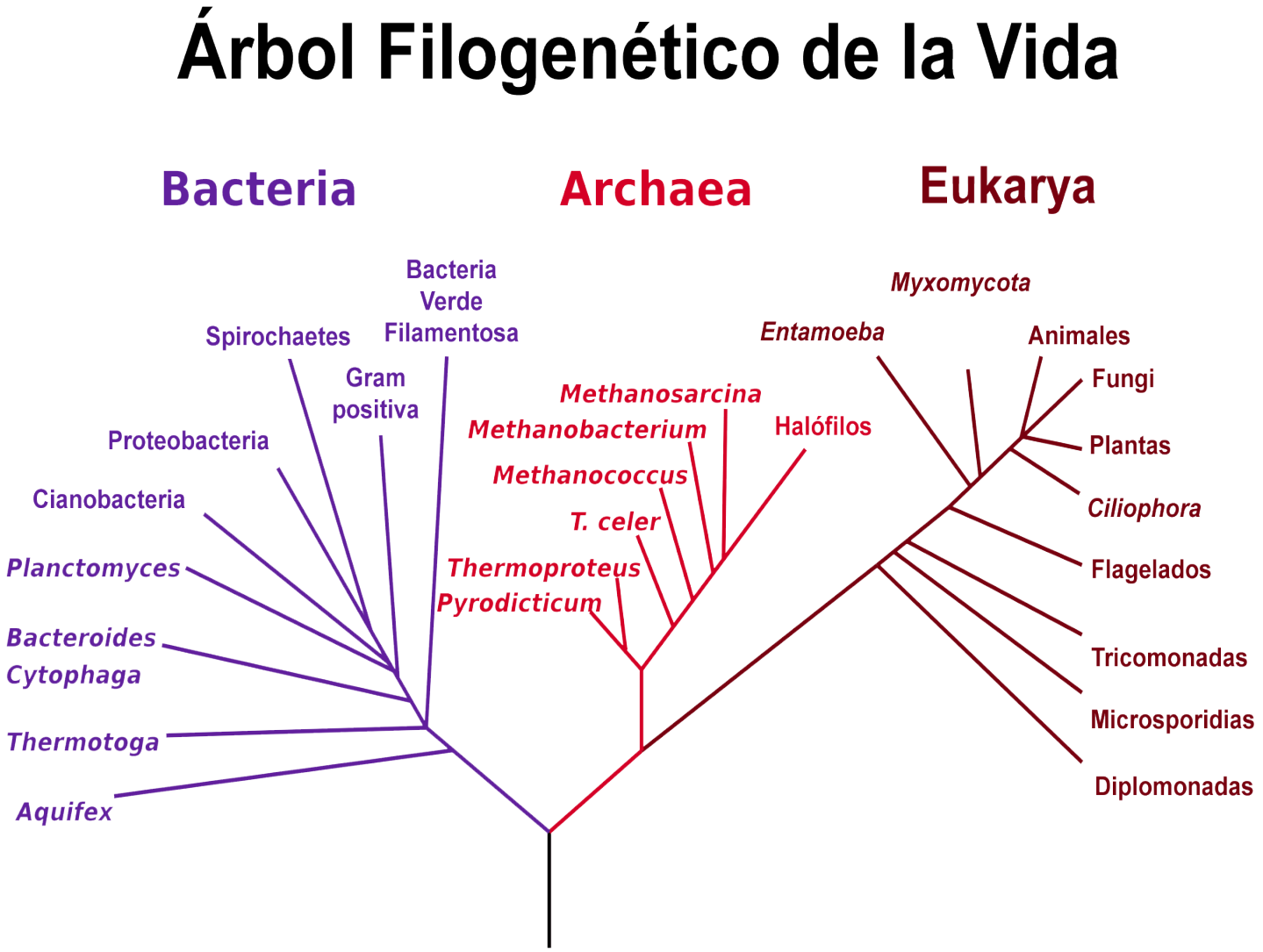
Recordemos

- Los datos recursivos son aquellos que están definidos en términos de sí mismos.
- Definición de una lista:
  - empty
  - (cons elemento lista)

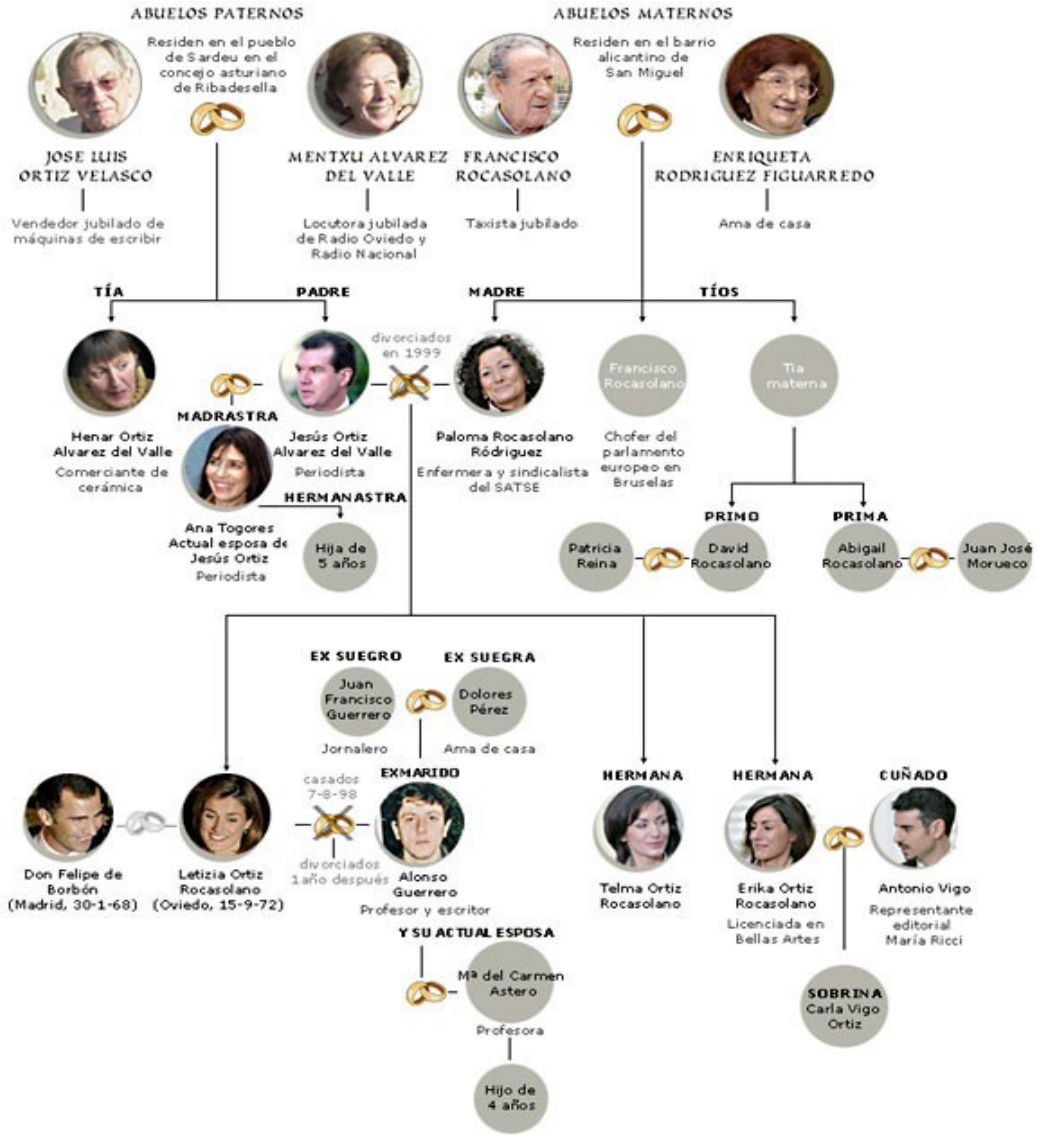
Árboles

- Los árboles son usados para representar información en la que existe una relación de jerarquía. Ejemplo: árboles genealógicos, filogenéticos, de sintaxis.
- Los árboles son considerados datos recursivos porque sus elementos son árboles

Árboles: ejemplos

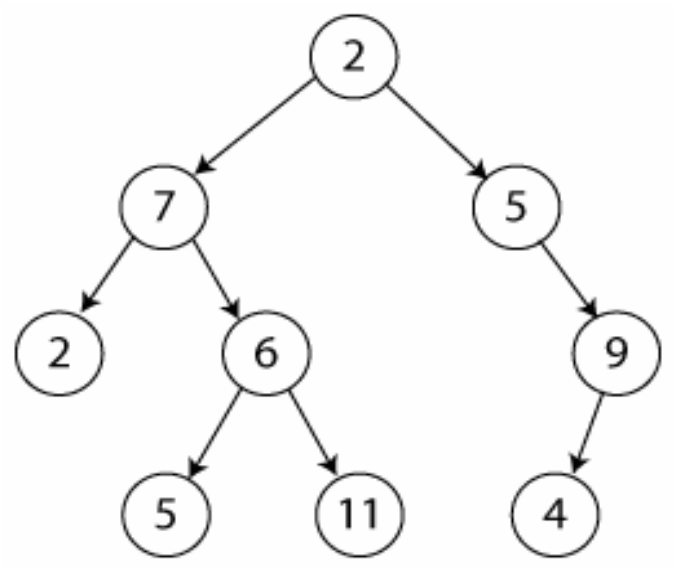


Árboles: ejemplo

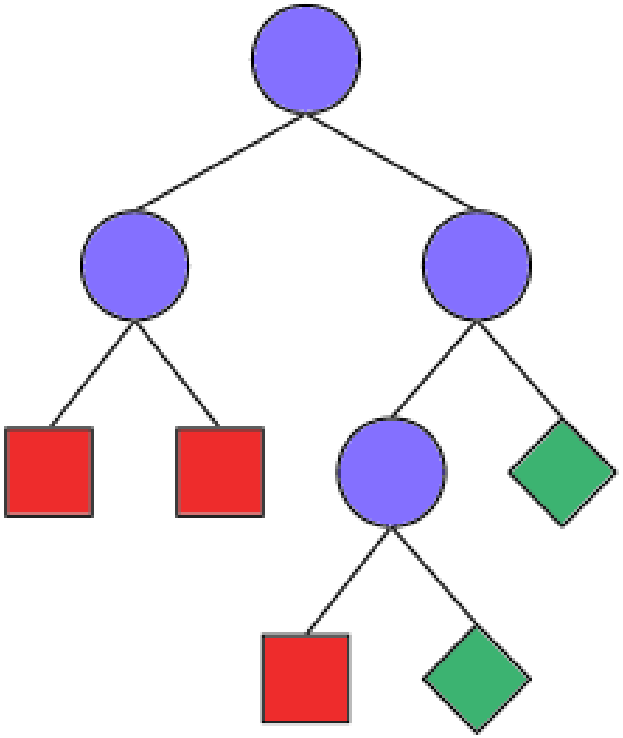


Árboles: ejemplos

Árbol binario



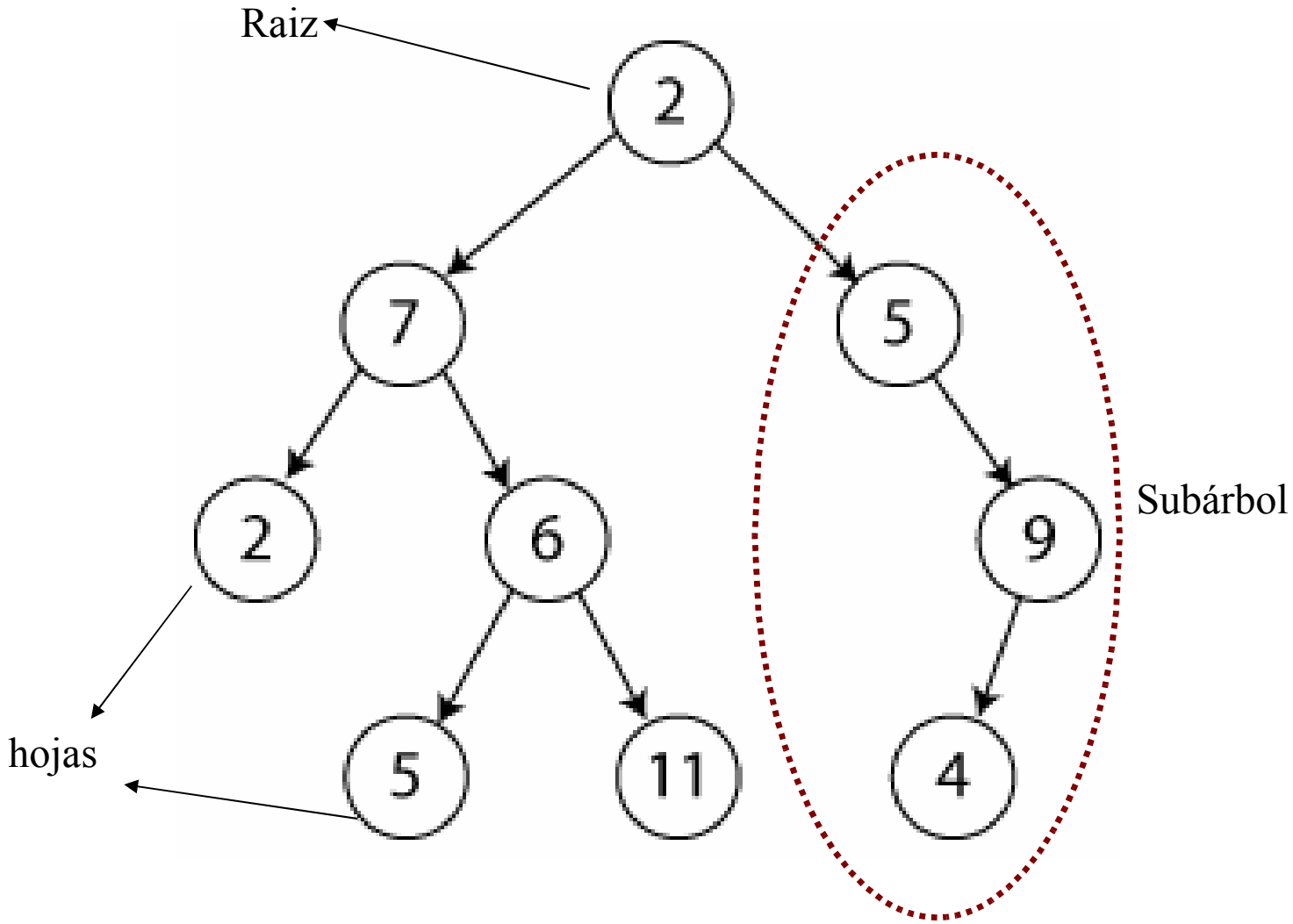
Árbol de búsqueda



Representación de árboles

- Para representar árboles debemos tener en cuenta las siguientes definiciones:
  - nodo
    - hoja: que no tiene nodos hijos (sucesores).
    - raíz: que no tiene nodos predecesores.
  - Subárbol: el conjunto de todos los predecesores de un nodo.

Árbol



Representación de árboles en scheme

- En scheme podemos representar árboles usando estructuras ó usando listas.
- Cuando los nodos tienen una cantidad restringida de hijos (no más de 2, 3, etc) se pueden usar estructuras.
- Cuando la cantidad de hijos de un nodo no es restringida, es más apropiado usar listas.

Árboles usando estructuras

- Árboles binarios:
  - nodo: estructura con tres campos: valor, árbol izquierdo y árbol derecho.

```
(define-struct nodo-b (num izq der))
```

- Árbol genealógico:
  - nodo: estructura con 4 campos: identificador, nombre, padre y madre.

```
(define-struct nodo-a (id nom padre madre))
```

Árboles usando estructuras

- Definición de datos:
  - Árboles binarios

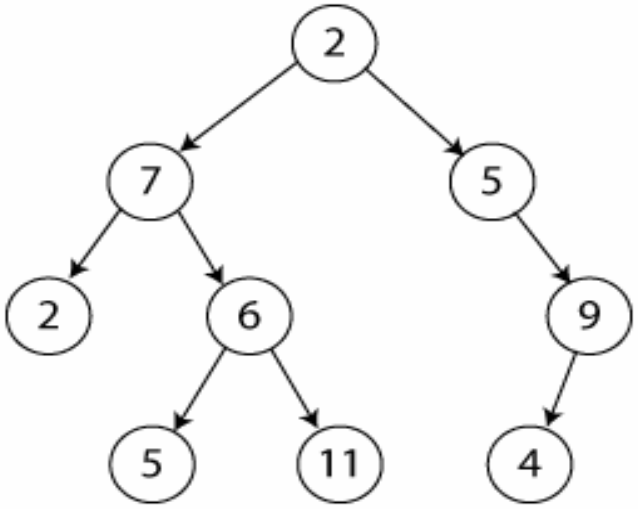
    Un nodo es:
    - empty
    - (make-nodo-b <numero> <izqu> <der>)
  - Árbol genealógico

    Un nodo es:
    - empty
    - (make-nodo-a <id> <nombre> <padre> <madre>)

Árboles usando listas

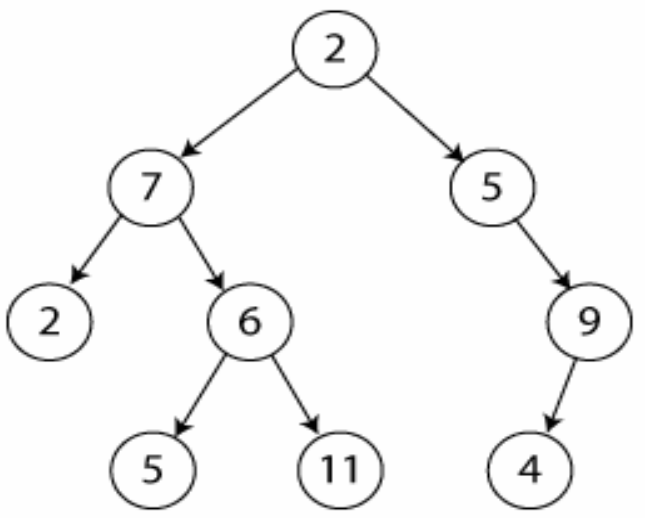
- Árbol binario.
  - nodo: lista de tres elementos: número, árbol izquierdo y árbol derecho.
  - Un nodo es:
    - empty
    - (list <numero> <subárbol izquierdo> <subárbol derecho>)
- Árbol genealógico:
  - nodo: Nombre, madre y padre.
  - Un nodo es:
    - empty
    - (list <nombre> <madre> <padre>)

Ejemplos: Árbol binario con estructuras




```
(make-nodo 2
  (make-nodo 7
    (make-nodo 2 empty empty)
    (make-nodo 6
     (make-nodo 5 empty empty)
     (make-nodo 11 empty empty)))
  (make-nodo 5
    empty
    (make-nodo 9
      (make-nodo 4 empty empty))
    empty))
```

Ejemplos: árbol binario con listas




```
(list 2
  (list 7
    (list 2 empty empty)
    (list 6
      (list 5 empty empty)
      (list 11 empty empty)))
  (list 5
    empty
    (list 9
      (list 4 empty empty)
      empty)))
```

Operaciones principales

- Arboles binarios:
  - **es hoja?:** retorna un booleano, **true** si el nodo es hoja, **false** si tiene subárboles.
  - **retornar árbol izquierdo:** retorna un árbol.
  - **retornar árbol derecho:** retorna un árbol.
- ¿Por qué son importantes?

  Porque nos permiten hacer operaciones con los árboles, como recorrerlos, contar los nodos, averiguar cuales son los pares, etc.

Operaciones principales en scheme

- Con estructuras:
  - **es hoja?**

```
(define (es hoja un-nodo)
  (cond
   [(and
     (empty? (nodo-izq un-nodo))
     (empty? (nodo-der un-nodo)) true]
   [else false]))
```

  - **árbol-izquierdo**

```
(nodo-izq un-nodo)
```

  - **árbol-derecho**

```
(nodo-izq un-nodo)
```

Operaciones principales en scheme

- Con listas:
  - **es-hoja?**

```
(define (es-hoja? nodo)
 (cond
   [(and
      (empty? (izquierdo nodo))
      (empty? (derecho nodo)) true]
   [else false]
 ))
```

  - **árbol-izquierdo**

```
(define (izquierdo nodo)
 (first (rest nodo)))
```

  - **árbol-derecho**

```
(define (derecho nodo)
  (first (rest (rest nodo)))))
```

Ejemplo

- Función que permite contar cuantos nodos tiene un árbol binario:
  - Idea: contar cuantos nodos tiene el árbol izquierdo, contar cuantos tiene el árbol derecho y sumar teniendo en cuenta la raíz. Si el árbol es vacío, entonces el total de nodos es cero.

Ejemplo

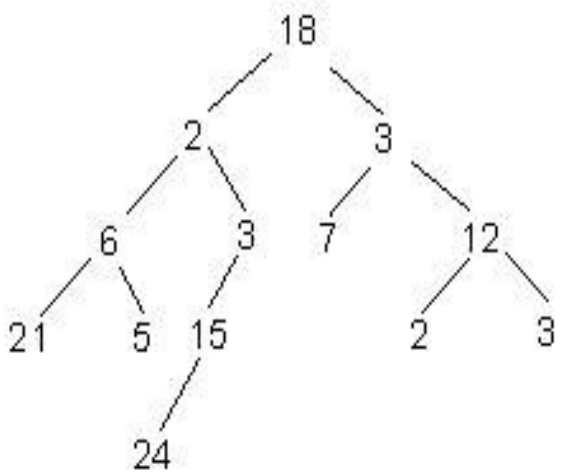
- Contrato

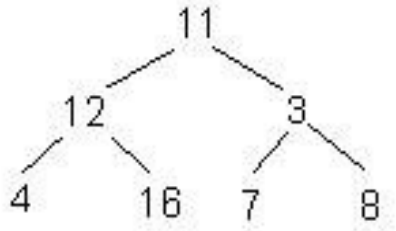
  ;cuantos: arbol_binario -> numero

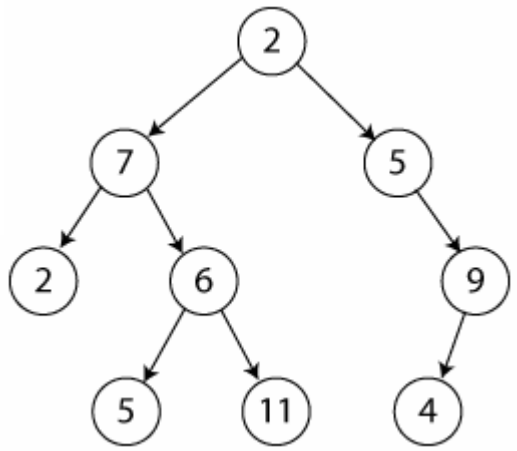
- Propósito

  ;Función que retorna la cantidad de nodos que contiene un árbol binario

- Ejemplos:







Ejemplos

- Programa (árboles con estructuras)

```
(define (cuantos arbol)
  (cond
    [(empty? arbol) 0]
    [(es_hoja? arbol) 1]
    [else (+  1 (cuantos (nodo-izq arbol)) (cuantos (nodo-der arbol)))]))
```

- Programa (árboles con listas)

```
(define (cuantos arbol)
  (cond
    [(empty? arbol) 0]
    [(es-hoja? arbol) 1]
    [else (+  1 (cuantos (izquierdo arbol)) (cuantos (derecho arbol)))]))
```

Ejericicio

- Diseñe un programa que cuente los nodos pares.
- Diseñe un programa que retorne los números de los nodos impares.
- ¿Cuales son las operaciones principales de un árbol genealógico?
- Escriba un ejemplo de árbol genealógico.

Soluciones (algunas)

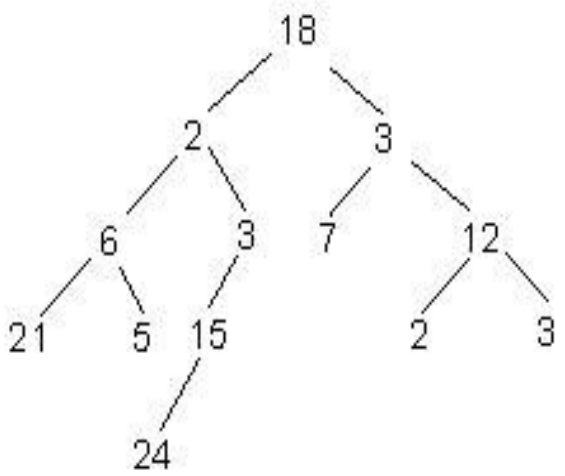
- Contrato

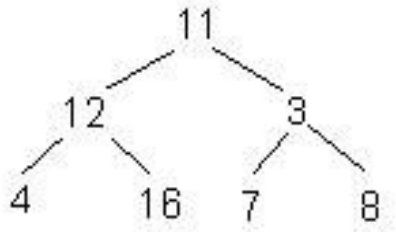
  ;cuantos_pares: arbol_binario -> numero

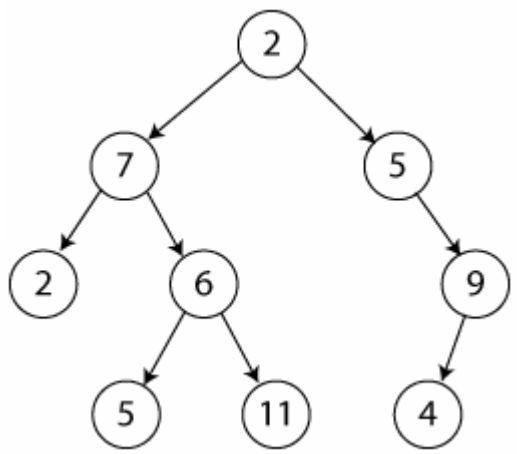
- Propósito

  ;Función que retorna la cantidad de nodos que contiene un árbol binario

- Ejemplos:







Ejercicio-1

- Programa

```
(define (cuantos_pares arbol)
  (cond
    [(empty? arbol) 0]
    [(and (es_hoja? arbol)
          (even? (nodo-dat arbol))) 1]
    [else
     (cond
       [(even? (nodo-dat arbol))
          (+  1 (cuantos_pares (nodo-izq arbol))
              (cuantos_pares (nodo-der arbol)))]
       [else
          (+ (cuantos_pares (nodo-izq arbol))
             (cuantos_pares (nodo-der arbol)))])]))
```

Ejercicio-2

- Programa

```
(define (list_pares arbol)
  (cond
    [(empty? arbol) 0]
    [(and (es_hoja? arbol)
        (even? (nodo-dat arbol))) (list (nodo-dat arbol))]
    [else
     (cond
        [(even? (nodo-dat arbol))
           (append  (list (nodo-dat arbol))
                    (list_pares (nodo-izq arbol))
                    (list_pares (nodo-der arbol)))]
        [else
           (append
                  (list_pares (nodo-izq arbol))
                  (list_pares (nodo-der arbol)))])]))
```