Árboles binarios

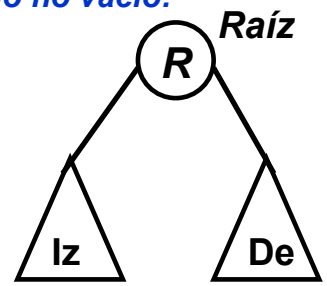
Lección 12

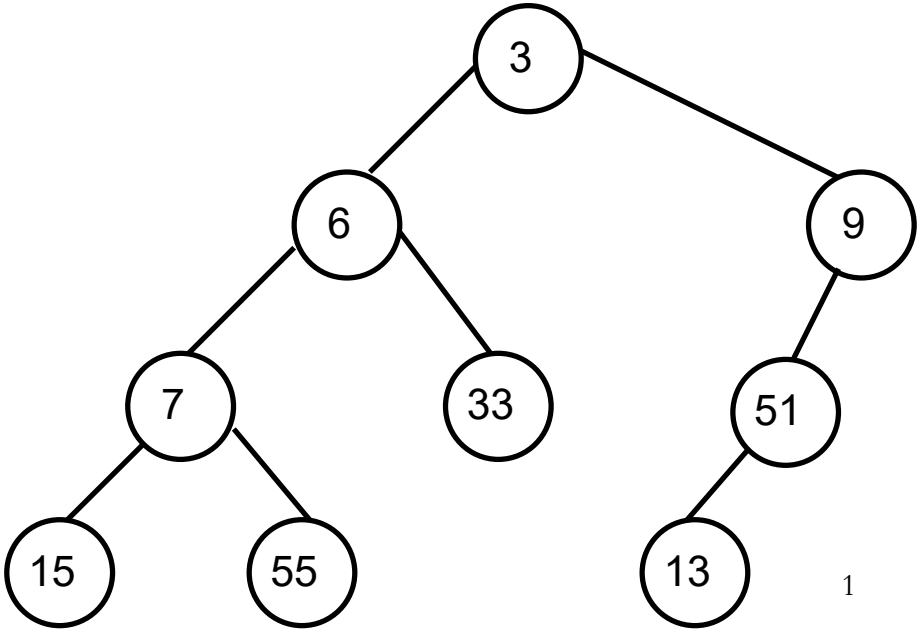
Árboles Binarios

- **Árbol binario**:
  - Conjunto de elementos o nodos del mismo tipo, tal que:
    - o bien es el conjunto vacío, y entonces se llama ***árbol vacío***
    - o bien es no vacío, en cuyo caso existe un elemento destacado llamado ***raíz***, y el resto de los elementos se distribuyen en **dos** subconjuntos disjuntos, llamados ***subárbol izquierdo*** y ***subárbol derecho***, cada uno de los cuales es un árbol binario

Árbol binario vacío:

Árbol binario no vacío:





Esquema

- Una especificación del TAD árboles binarios genéricos
- Implementación estática
- Implementación dinámica
- Implementación de recorridos

Especificación de árboles binarios

espec árbolesBinarios
usa booleanos,naturales
parámetro formal
género elemento
fpf
género arbin *{Los valores del genero arbin representan la definición dada (transp. 1) de árbol binario…}*
operaciones
vacío: → arbin
{Devuelve el árbol vacío}
plantar: elemento e , arbin ai , arbin ad → arbin
{Devuelve un árbol cuyo elemento raíz es e, su subárbol izquierdo es ai y el derecho es ad}
esVacío?: arbin a → booleano
{Devuelve verdad si y sólo si a es el árbol vacío}
parcial raíz: arbin a → elemento
{Devuelve el elemento raíz de a. ***Parcial: la operación no está definida si esVacío?(a)****}*
parcial sublzq: arbin a → arbin
{Devuelve el subárbol izquierdo de a. ***Parcial: la op. no está definida si esVacío?(a)****}*
parcial subDer: arbin a → arbin
{Devuelve el subárbol derecho de a. ***Parcial: la op. no está definida si esVacío?(a)****}*
parcial altura: arbin a → natural
{Devuelve la altura de a. ***Parcial: la operación no está definida si esVacío?(a)****}*
fespec

Implementación estática

- *Representación basada en cursores a los hijos:*
  - Representación de árboles en base a vectores
    - Cada componente del vector guarda un nodo con:
      - el *elemento* que contiene,
      - los índices *(cursores*) donde se encuentran sus hijos *izq* y *der,*
      - booleano indicando si la componente del vector está en uso o no
    - El árbol estará formado por el índice (cursor) donde se encuentra la raíz del árbol, y el vector que almacena los nodos
      - Si el vector está indexado de [1..max], el máximo árbol representable tendrá **max** nodos
      - El **Árbol vacío** podrá representarse con valor ***0*** como índice de la raíz
        - » En general, para el árbol vacío usaríamos un valor de índice para la raíz igual a 1 menos que el menor valor de índice para el vector…

Implementación estática

constante max = ... *{máximo número de elementos almacenables, o nodos del árbol}*

tipo

arbin = 0..max; *{componente del vector en la que se encuentra la raíz del árbol. Si es 0 significa árbol vacío}*

nodo = **registro**

dato:elemento;

izq,der:arbin;

ocupado:booleano

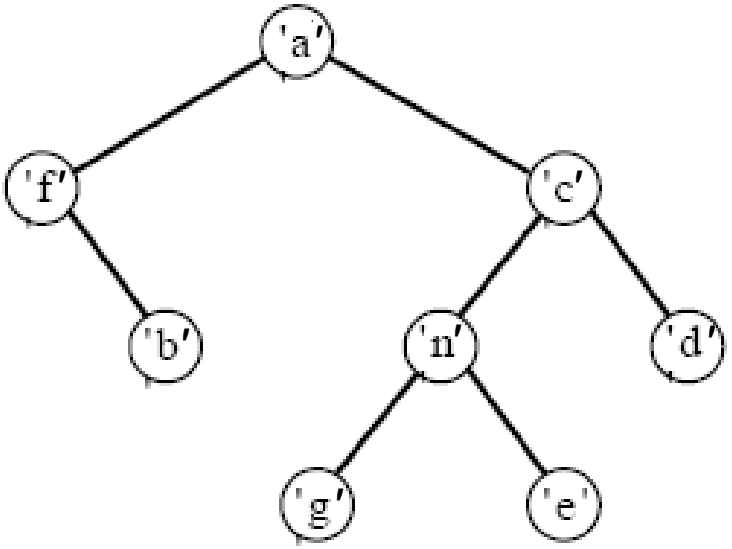
freg;

tpVectorDeNodos = **vector**[1..max] **de** nodo;

variable a,b: arbin; vector_nodos:tpVectorDeNodos

{con esta representación un mismo vector puede incluso almacenar varios árboles no conectados}

Implementación estática



a 1

{con esta representación un mismo vector puede incluso almacenar varios árboles no conectados}

vector_nodos

[1]	'a'	2	4	v
[2]	'f'	0	3	v
[3]	'b'	0	0	v
[4]	'c'	5	8	v
[5]	'n'	6	7	v
[6]	'g'	0	0	v
[7]	'e'	0	0	v
[8]	'd'	0	0	v
[9]	?	?	?	f
[10]	?	?	?	f

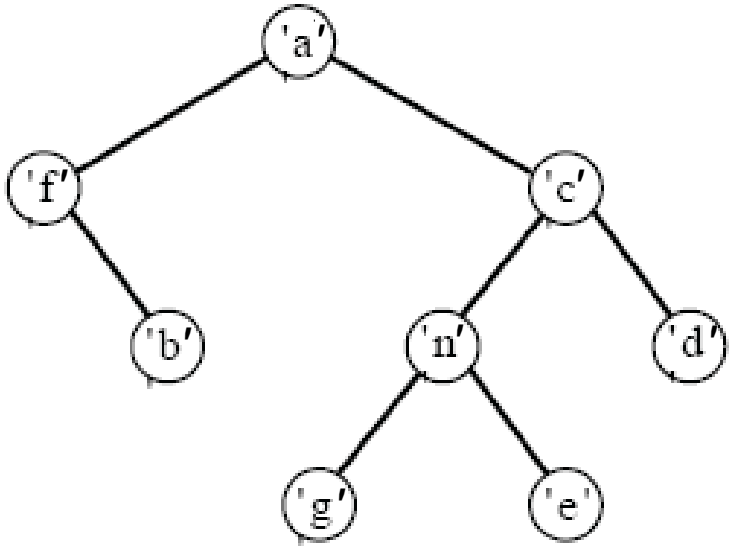
{Problemas: ¿Qué tamaño de vector usar? ¿Cómo localizar eficientemente una componente libre del vector? Y si se puede borrar: ¿Qué hay que hacer para borrar un nodo del árbol? por ejemplo, borrar 'e' o 'b'…}

Implementación dinámica

- Encadenamiento mediante punteros (a los hijos) de los elementos del árbol

```
tipos arbin = ↑nodo;
      nodo = registro
                dato: elemento;
                izq,der: arbin
             freg
```

¿Cómo quedará representado el siguiente árbol?



Implementación dinámica

módulo genérico árbolesBinarios

parámetros

tipo elemento *{si se añade la operación de comparar por igualdad dos árboles, entonces es necesario que los datos de tipo elemento se puedan comparar por igualdad, y aquí habría que poner: con*

función "="(e1,e2:elemento) devuelve booleano{verdad si e1=e2…} *}*

exporta

tipo arbin *{Árboles binarios de elementos (cumplen la definición dada (transp. 1) de árbol binario…)}*

procedimiento vacío(**sal** a:arbin)

{devuelve un árbol vacío en a}

procedimiento plantar(**sal** a:arbin; **ent** e:elemento;
ent ai,ad:arbin)

{devuelve un árbol a cuyo elemento raíz es e, su subárbol izquierdo es ai y su subárbol derecho es ad (cuidado: no se crean copias profundas de dichos subárboles)}

función esVacío?(a:arbin) **devuelve** booleano

{devuelve verdad si y sólo si a es un árbol vacío}

...

Implementación dinámica

```
...
  procedimiento raíz(ent a:arbin; sal error:booleano;
                                  sal e:elemento)
```

{Si no esVacío?(a), devuelve en e el elemento raíz de a y devuelve error= falso. Si a es vacío, devuelve error=verdad}

```
  procedimiento subIzq(ent a:arbin;
                       sal error:booleano; sal ai:arbin)
```

{Si no esVacío?(a), devuelve error=falso y en ai devuelve el subárbol izquierdo de a (cuidado: no se devuelve una copia profunda de dicho subárbol). En caso contrario devuelve error=verdad.}

```
  procedimiento subDer(ent a:arbin;
                       sal error:booleano; sal ad:arbin)
```

{Si no esVacío?(a), devuelve error=falso y en ad devuelve el subárbol derecho de a (cuidado: no se devuelve una copia profunda de dicho subárbol). En caso contrario devuelve error=verdad.}

```
  procedimiento altura(ent a:arbin;
                       sal error:booleano; sal h:natural)
```

{Si no esVacío?(a), devuelve error=falso y en h la altura de a. En caso contrario devuelve error=verdad.}

```
...
```

Implementación dinámica

```
...

  procedimiento duplicar(sal nuevo:arbin; ent viejo:arbin)
  {Duplica (copia profunda) la representación del árbol viejo guardándolo en nuevo.}

  función iguales (a1,a2:arbin) devuelve booleano
  {Devuelve verdad, si y sólo si a1 y a2 tienen los mismos elementos y exactamente en
   las mismas posiciones. Se devolverá falso en cualquier otro caso.}

  procedimiento liberar(e/s a:arbin)
  {Libera la memoria dinámica accesible desde a, quedando a como árbol vacío.}

Implementación   {Fin parte pública de la implementación. Inicio parte privada:}

    tipos arbin = ↑nodo;

          nodo = registro
                    dato:elemento;
                    izq,der:arbin
                 freg

...
```

Podrían especificarse funciones "iguales" con otros significados

Implementación dinámica

```
...

  procedimiento vacío(sal a:arbin)
  principio
    a:=nil
  fin

  procedimiento plantar(sal a:arbin;
                        ent e:elemento; ent ai,ad:arbin)
  {Devuelve un árbol binario con el elemento e como raíz, y como
subárboles hijos enlaza los árboles dados: ai y ad}

  principio
    nuevoDato(a);
    a↑.dato:=e;
    a↑.izq:=ai;
    a↑.der:=ad {¡OJO! No crea copias de los subárboles,
                se están enlazando}
  fin
...
```

Coste $\Theta(1)$

Coste $\Theta(1)$

Implementación dinámica

```
...

función esVacío(a:arbin) devuelve booleano
principio
  devuelve (a=nil)                                    Coste Θ(1)
fin

procedimiento raíz(ent a:arbin;
                   sal error:booleano; sal e:elemento)
principio
  si esVacío(a) entonces
    error:=verdad
  sino
    error:=falso;                                     Coste Θ(1)
    e:=a↑.dato
  fsi
fin
...
```

Implementación dinámica

```
...
procedimiento subIzq(ent a:arbin;
                     sal error:booleano; sal ai:arbin)
principio
  si esVacío(a) entonces
    error:=verdad
  sino
    error:=falso;
    ai:=a↑.izq   {¡OJO! No crea copia del subárbol,
                  devuelve el subárbol}
  fin
Fin

procedimiento subDer(ent a:arbin;
                     sal error:booleano; sal ad:arbin)
{… implementación análoga a la del anterior…}
...
```

Coste $\Theta(1)$

Coste $\Theta(1)$