

# Temario

## Tema 5. Estructuras de Datos no Lineales

5.1 Árboles Binarios

5.2 Árboles n-arios

- *Especificación*
- *Utilización*
- *Representación Enlazada*

5.3 Árboles Binarios de Búsqueda

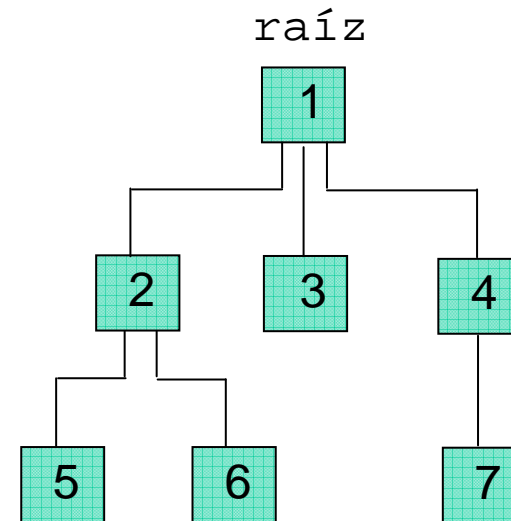
5.4 Árboles Parcialmente Ordenados



## 5.2 Árboles N-arios

### Árbol n-ario:

- O bien es el conjunto vacío  
 $\Rightarrow$  árbol **vacío**.
- O bien no es vacío:
  - elemento **Raíz**.
  - $A_1, \dots, A_n$  **Subárboles**.



## 5.2 Árboles N-arios

### Formas de construir Árboles n-arios:

- Añadir nodos a un Árbol:
  - como Hijo más a la Izquierda.
  - como Hermano Derecho.
- Utilizar directamente en la definición recursiva de Árbol n-ario:
  - Entradas:
    - $A_1, \dots, A_n$
    - Nodo Raiz
  - Salida:
    - El nuevo Árbol n-ario



## 5.2 Árboles N-arios

### Operaciones de Construcción:

```
Arbol(Elemento e);
```

### Operaciones de Acceso:

```
Elemento recupera(Posicion p) const;  
void asigna(Posicion p, Elemento e);
```

### Operaciones de Modificación:

```
void insertaIzquierdo(Posicion p, Elemento e);  
void insertaDerecho(Posicion p, Elemento e);  
void supprimeIzquierdo(Posicion p);  
void supprimeDerecho(Posicion p);
```

### Operaciones de Posicionamiento:

```
Posicion inicio() const;  
Posicion fin() const;  
Posicion izquierdo(Posicion p) const;  
Posicion derecho(Posicion p) const;  
Posicion padre(Posicion p) const;
```

## 5.2 Árboles N-arios

### TDA Árbol : Especificación Informal (Construcción y Acceso)

`Arbol(Elemento e)`

**efecto:** devuelve un árbol n-ario con el elemento `e` en la raíz.

`Elemento recupera(Posicion p) const`

**requerimientos:** `p` no es fin.

**efecto:** devuelve el elemento que ocupa la posición `p`.

`void asigna(Posicion p, Elemento e)`

**requerimientos:** `p` no es fin.

**efecto:** asigna el elemento `e` a la posición `p`.

## 5.2 Árboles N-arios

### TDA Árbol: Especificación Informal (Modificación)

`void insertaIzquierdo(Posicion p, Elemento e)`

**requerimientos:** `p` no es fin.

**efecto:** inserta `e` como hijo más a la izquierda de `p`.

`void insertaDerecho(Posicion p, Elemento e)`

**requerimientos:** `p` no es fin y `p` no es la raíz.

**efecto:** inserta `e` como hermano derecho de `p`.

`void suprimeIzquierdo(Posicion p)`

**requerimientos:** `p` no es fin y `p` tiene hijos.

**efecto:** suprime el hijo más a la izquierda de `p` y todos sus descendientes.

`void suprimeDerecho(Posicion p)`

**requerimientos:** `p` no es fin y `p` tiene hermanos a la derecha.

**efecto:** suprime el hermano derecho de `p` y todos sus descendientes.



## 5.2 Árboles N-arios

### TDA Árbol: Especificación Informal (Posicionamiento)

Posicion inicio() const

**efecto:** devuelve la posición raíz.

Posicion fin() const

**efecto:** devuelve la posición fin.

Posicion izquierdo(Posicion p) const

**requerimientos:** p no es fin.

**efecto:** devuelve el hijo más a la izquierda de p o la posición fin si p no tiene hijos.

Posicion derecho(Posicion p) const

**requerimientos:** p no es fin.

**efecto:** devuelve el hermano derecho de p o la posición fin si p no tiene hermanos a la derecha.

Posicion padre(Posicion p) const

**requerimientos:** p no es fin.

**efecto:** devuelve el padre de p o la posición fin si p es la posición raíz.



## 5.2 Árboles N-arios

### Clase Arbol: Operaciones de Clase Mínima

```
class Arbol
{
    private:
        ...
    public:
        Arbol(Elemento e);
        ~Arbol();
        Arbol(const Arbol & a);
        Arbol & operator=(const Arbol & a);
        ...
};
```



## 5.2 Árboles N-arios

### Clase Arbol: Operaciones de Posicionamiento

```
class Arbol
{
    private:
        ...
    public:
        ...
        Posicion inicio() const;
        Posicion fin() const;
        Posicion izquierdo(Posicion p) const;
        Posicion derecho(Posicion p) const;
        ...
};
```

## 5.2 Árboles N-arios

### Clase Arbol: Operaciones de Acceso y Modificación

```
class Arbol
{
    private:
        ...
    public:
        ...
        Elemento recupera(Posicion p) const;
        void asigna(Posicion p, Elemento e);

        void insertaIzquierdo(Posicion p, Elemento e);
        void insertaDerecho(Posicion p, Elemento e);
        void suprimeIzquierdo(Posicion p);
        void suprimeDerecho(Posicion p);
};
```

## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

Arbol a(1);

raíz




## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Arbol a(1);
```

```
Posicion p1 = a.inicio();
```

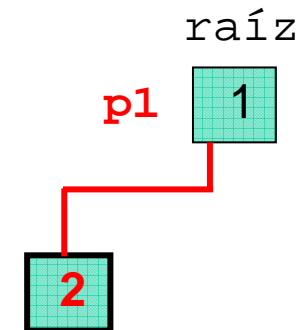
raíz  
p1 



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

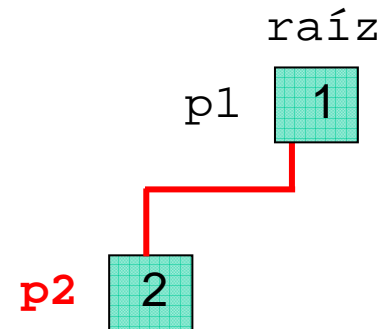
```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);
```



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

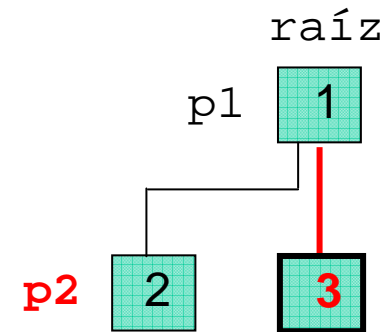
```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);
```



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

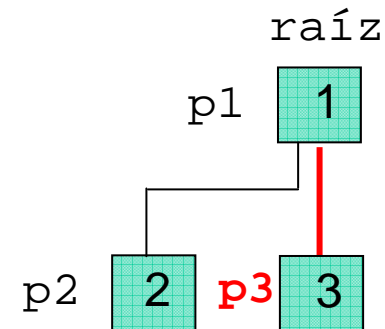
```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);  
a.insertaDerecho(p2,3);
```



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);  
a.insertaDerecho(p2,3);  
Posicion p3 = a.derecho(p2);
```

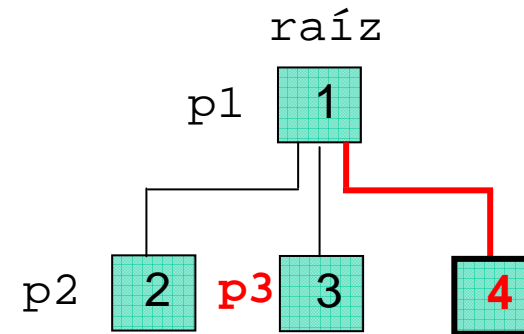




## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

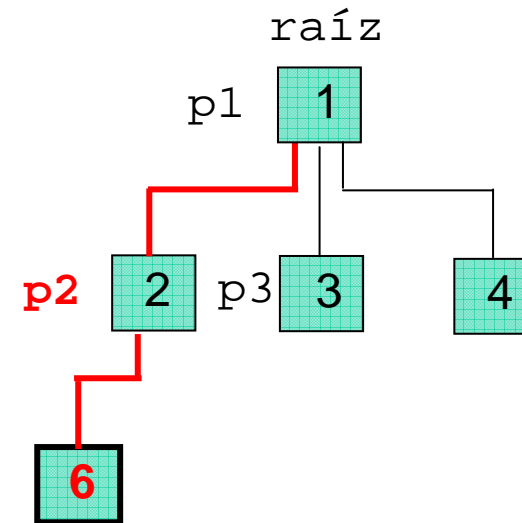
```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);  
a.insertaDerecho(p2,3);  
Posicion p3 = a.derecho(p2);  
a.insertaDerecho(p3,4);
```



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

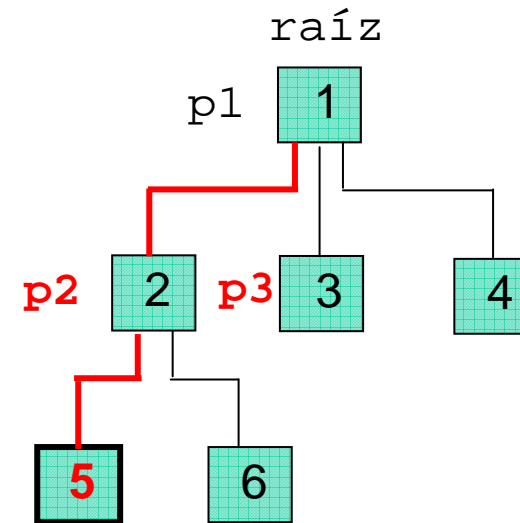
```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);  
a.insertaDerecho(p2,3);  
Posicion p3 = a.derecho(p2);  
a.insertaDerecho(p3,4);  
a.insertaIzquierdo(p2,6);
```



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

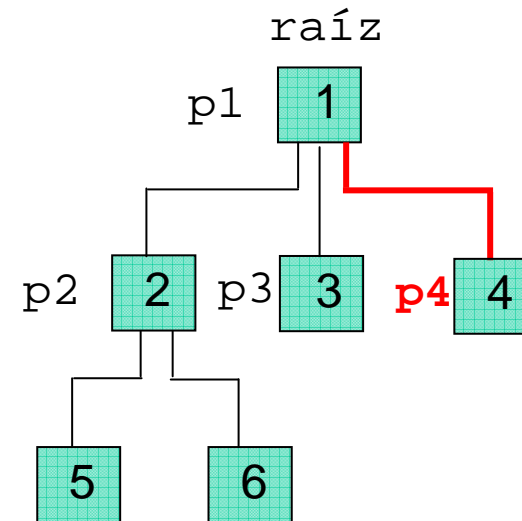
```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);  
a.insertaDerecho(p2,3);  
Posicion p3 = a.derecho(p2);  
a.insertaDerecho(p3,4);  
a.insertaIzquierdo(p2,6);  
a.insertaIzquierdo(p2,5);
```



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

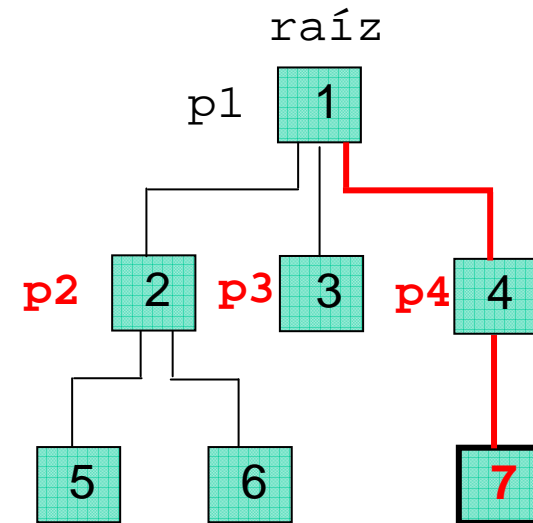
```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);  
a.insertaDerecho(p2,3);  
Posicion p3 = a.derecho(p2);  
a.insertaDerecho(p3,4);  
a.insertaIzquierdo(p2,6);  
a.insertaIzquierdo(p2,5);  
Posicion p4 = a.derecho(p3);
```



## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Arbol a(1);  
Posicion p1 = a.inicio();  
a.insertaIzquierdo(p1,2);  
Posicion p2 = a.izquierdo(p1);  
a.insertaDerecho(p2,3);  
Posicion p3 = a.derecho(p2);  
a.insertaDerecho(p3,4);  
a.insertaIzquierdo(p2,6);  
a.insertaIzquierdo(p2,5);  
Posicion p4 = a.derecho(p3);  
a.insertaIzquierdo(p4,7);
```



## 5.2 Árboles N-arios

Esquemas de recorrido en Árboles n-arios:

- **Preorden(A):**
  - Si A es vacío  $\Rightarrow$  lista vacía.
  - Si A no es vacío  $\Rightarrow$  Raíz + Preorden( $A_1$ ) + ... + Preorden( $A_n$ ).
- **Inorden(A):**
  - Si A es vacío  $\Rightarrow$  lista vacía.
  - Si A no es vacío  $\Rightarrow$  Inorden( $A_1$ ) + Raíz + Inorden( $A_2$ ) + ... + Inorden( $A_n$ ).
- **Postorden(A):**
  - Si A es vacío  $\Rightarrow$  lista vacía.
  - Si A no es vacío  $\Rightarrow$  Posorden( $A_1$ ) + ... + Posorden( $A_n$ ) + Raíz.

## 5.2 Árboles N-arios

### Utilización de Árboles n-arios:

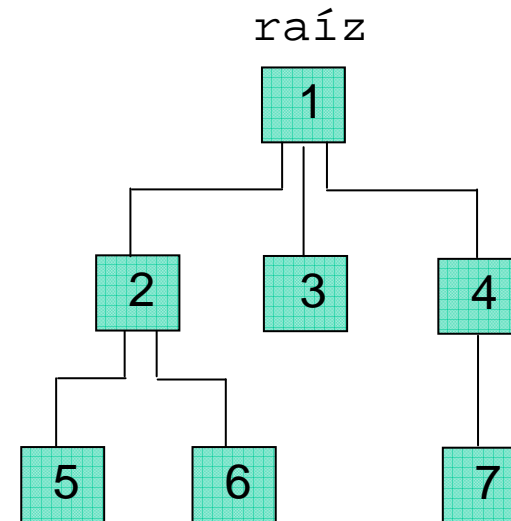
- Métodos Recursivos
- Bucle (for o while) para recorrer los n hijos
- Paso de parámetros
  - Por valor: no es eficiente, no funciona
  - Por referencia: eficiente, adecuado si modificamos el árbol
  - Por referencia constante: eficiente, adecuado si no modificamos el árbol
- Por simplicidad, utilizaremos siempre el paso por referencia

## 5.2 Árboles N-arios

### Esquemas de Recorrido en Árboles n-arios: Preorden

1, 2, 5, 6, 3, 4, 7

```
void preorden(Arbol & a, Posicion p)
{
    if (p!=a.fin())
    {
        cout << a.recupera(p) << " ";
        for(Posicion q=a.izquierdo(p); q!=a.fin(); q=a.derecho(q))
            preorden(a,q);
    }
}
```



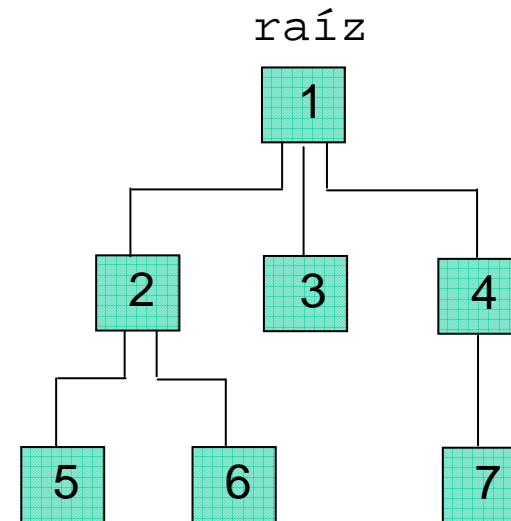


## 5.2 Árboles N-arios

### Esquemas de Recorrido en Árboles n-arios: Inorden

5, 2, 6, 1, 3, 7, 4

```
void inorden(Arbol & a, Posicion p)
{
    if (p!=a.fin())
    {
        Posicion q = a.izquierdo(p);
        inorden(a,q);
        cout << a.recupera(p) << " ";
        while(q!=a.fin()) { q = a.derecho(q); inorden(a,q); }
    }
}
```

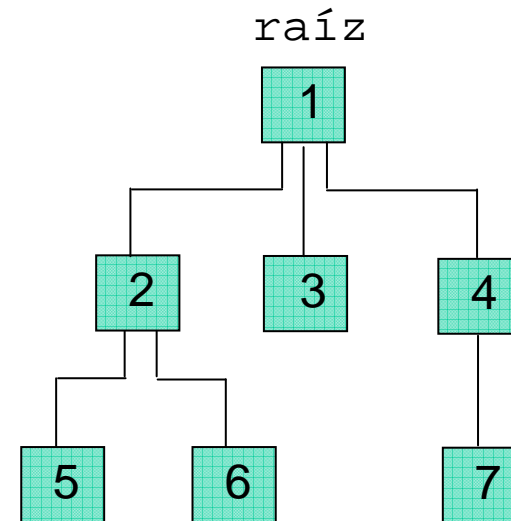


## 5.2 Árboles N-arios

### Esquemas de Recorrido en Árboles n-arios: Posorden

5, 6, 2, 3, 7, 4, 1

```
void posorden(Arbol & a, Posicion p)
{
    if (p!=a.fin())
    {
        for(Posicion q=a.izquierdo(p); q!=a.fin(); q=a.derecho(q))
            posorden(a,q);
        cout << a.recupera(p) << " ";
    }
}
```



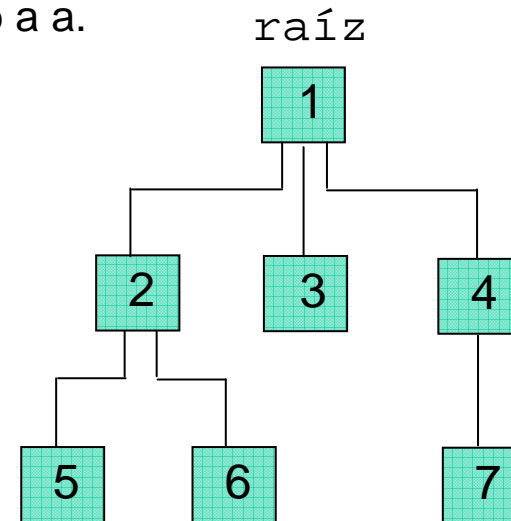
## 5.2 Árboles N-arios

### Descendientes:

Un nodo a es descendiente de b, si existe un camino de b a a.

Todo nodo es descendiente de sí mismo.

Nodo	Descendientes	Descendientes Propios
1	1,2,3,4,5,6,7	2,3,4,5,6,7
2	2,5,6	5,6
3	3	-
4	4,7	7
5	5	-
6	6	-



```
// Imprime los descendientes de p en preorden
void descendientes(Arbol & a, Posicion p)
{
    if (p!=a.fin())
    {
        cout << a.recupera(p) << " ";
        for(Posicion q=a.izquierdo(p); q!=a.fin(); q=a.derecho(q))
            descendientes(a,q);
    }
}
```



## 5.2 Árboles N-arios

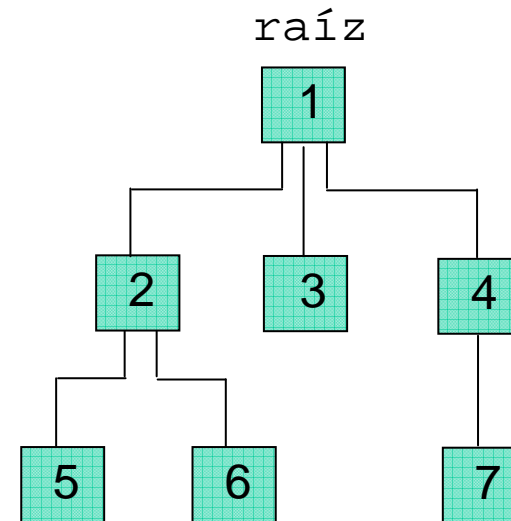
### Hoja:

Nodo sin descendientes propios.

Nodo	Hojas Descendientes del Nodo
1	5,6,7
2	5,6
3	3
4	7
5	5
6	6

// Imprime las hojas descendientes de p

```
void hojas(Arbol & a, Posicion p)
{
    if (p!=a.fin())
        if (a.izquierdo(p)==a.fin())
            cout << a.recupera(p) << " ";
        else
            for(Posicion q=a.izquierdo(p); q!=a.fin(); q=a.derecho(q))
                hojas(a,q);
}
```



## 5.2 Árboles N-arios

### Altura de un nodo:

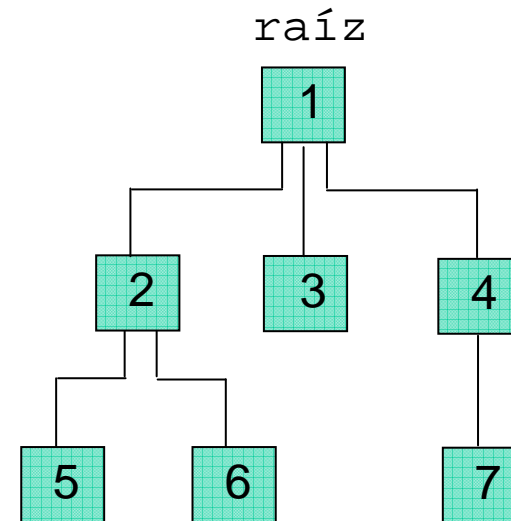
Longitud del camino más largo de ese nodo a una hoja.

#### Nodos      Altura

1	2
2,4	1
3,5,6,7	0

```
// Devuelve la altura de p  
// Si p es fin devuelve -1
```

```
int altura(Arbol & a, Posicion p)  
{  
    if (p==a.fin()) return -1;  
    int max = 0;  
    for(Posicion q = a.izquierdo(p); q!=a.fin(); q=a.derecho(q))  
    {  
        int n = altura(a,q)+1;  
        if (n>max) max = n;  
    }  
    return max;  
}
```



## 5.2 Árboles N-arios

// Elimina en a a partir de p todos los hijos de los nodos iguales a e

```
bool poda(Arbol & a, Posicion p, Elemento e)
{
    if (p!=a.fin())
        if (a.recupera(p)==e)
            while(a.izquierdo(p)!=a.fin())
                a.suprimeIzquierdo(p);
        else
            for(Posicion q = a.izquierdo(p);q!=a.fin();q=a.derecho(q))
                poda(a,q,e);
}
```

## 5.2 Árboles N-arios

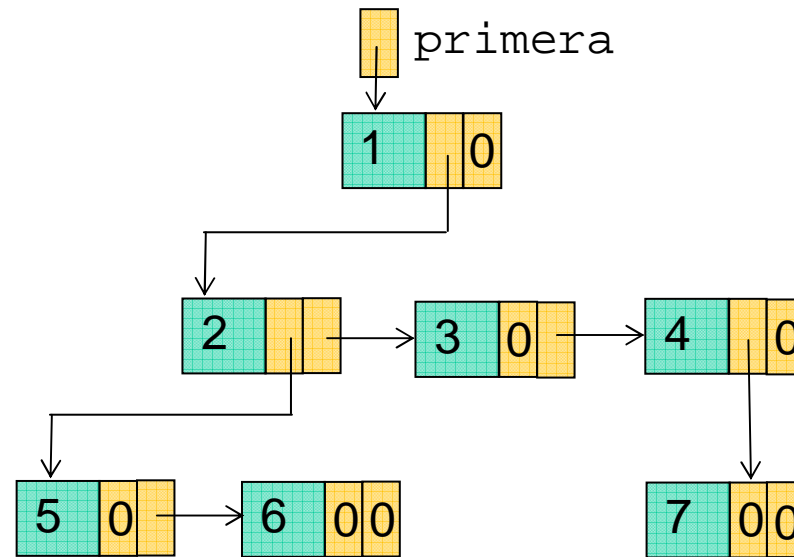
### Representación Enlazada Hijo Izquierdo – Hermano Derecho

```
typedef int Elemento;
```

```
struct Celda  
{  
    Elemento elem;  
    Celda * izq, * der;  
};
```

```
typedef Celda * Posicion;
```

```
class Arbol  
{  
private:  
    Celda * primera;  
public:  
    ...  
};
```



## 5.2 Árboles N-arios

### Celda: Operaciones de Clase Mínima

```
struct Celda
{
    Elemento elem;
    Celda * izq, * der;
    Celda(Elemento e, Celda * i, Celda * d) { elem = e; izq = i; der = d; }
    ~Celda();
    Celda(const Celda & c);
};

Celda::~~Celda()
{
    if (izq!=0) delete izq;
    if (der!=0) delete der;
}

Celda::Celda(const Celda & c)
{
    elem = c.elem;
    if (c.izq==0) izq = 0; else izq = new Celda(*c.izq);
    if (c.der==0) der = 0; else der = new Celda(*c.der);
}
```





## 5.2 Árboles N-arios

### Arbol: Operaciones de Clase Mínima

```
Arbol(Elemento e)                { primera = new Celda(e,0,0); }
~Arbol()                        { delete primera; }
Arbol(const Arbol & a)           { primera = new Celda(*a.primera); }

Arbol & operator=(const Arbol & a);

Arbol & Arbol::operator=(const Arbol & a)
{
    delete primera;
    primera = new Celda(*a.primera);
    return * this;
}
Arbol & Arbol::operator=(const Arbol & a)
{
    Arbol aux(a);
    swap(primeras,aux.primera);
    return * this;
}
```



## 5.2 Árboles N-arios

### Representación Enlazada Hijo Izquierdo – Hermano Derecho

```
Posicion inicio()                const { return primera; }
Posicion fin()                   const { return 0; }
Posicion izquierdo(Posicion p)   const { return p->izq; }
Posicion derecho(Posicion p)     const { return p->der; }

Elemento recupera(Posicion p)    const { return p->elem; }
void asigna(Posicion p, Elemento e) { p->elem = e; }

void insertaIzquierdo(Posicion p, Elemento e) { p->izq = new Celda(e,0,p->izq); }

void insertaDerecho(Posicion p, Elemento e) { p->der = new Celda(e,0,p->der); }

void supprimeIzquierdo(Posicion p) { Celda * c = p->izq;
                                     p->izq = p->izq->der;
                                     delete c;
                                     }

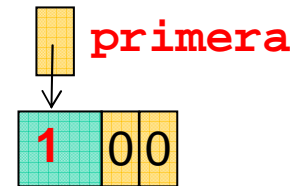
void supprimeDerecho(Posicion p) { Celda * c = p->der;
                                    p->der = p->der->der;
                                    delete c;
                                    }
```

## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Arbol(Elemento e)
{
    primera = new Celda(e,0,0);
}
```

```
Arbol a(1);
```

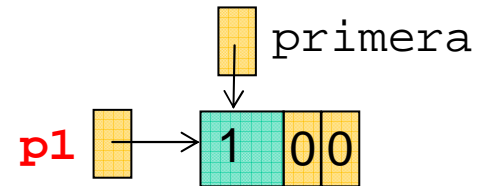


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Posicion inicio() const
{
    return primera;
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
```

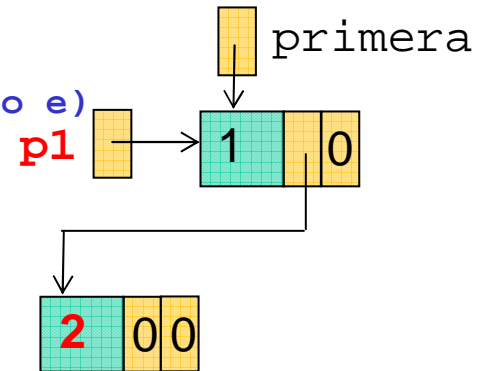


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
void insertaIzquierdo(Posicion p, Elemento e)
{
    p->izq = new Celda(e,0,p->izq);
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
```

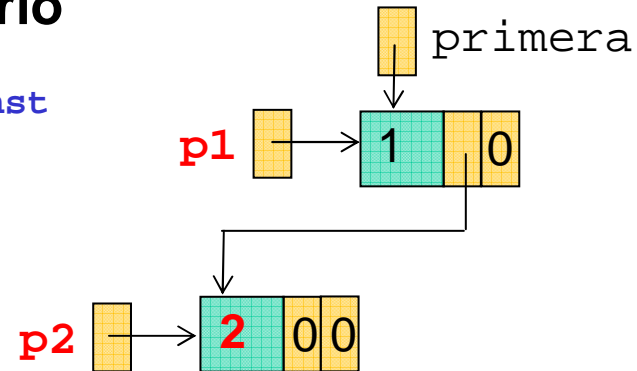


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Posicion izquierdo(Posicion p) const
{
    return p->izq;
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
```

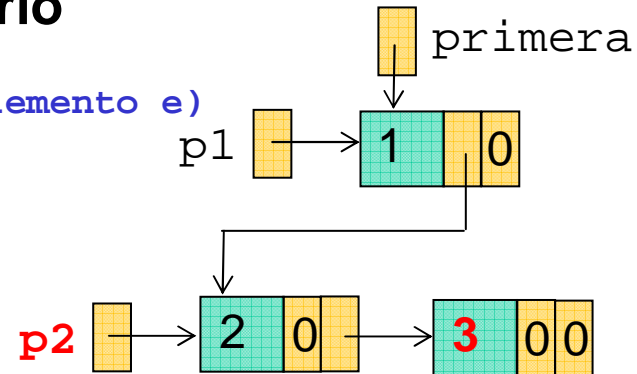


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
void insertaDerecho(Posicion p, Elemento e)
{
    p->der = new Celda(e,0,p->der);
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
a.insertaDerecho(p2,3);
```

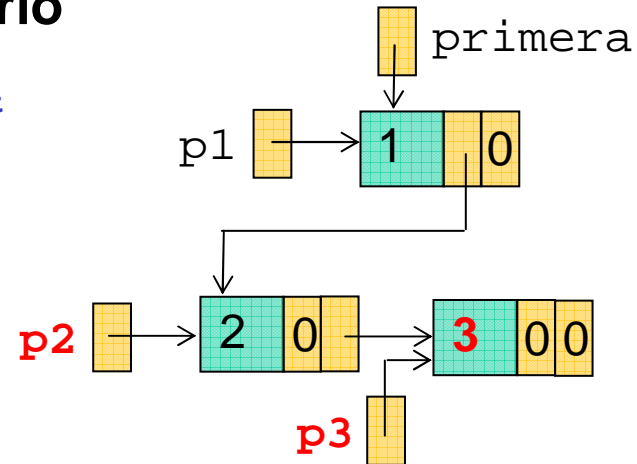


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Posicion derecho(Posicion p) const
{
    return p->der;
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
a.insertaDerecho(p2,3);
Posicion p3 = p2.derecho();
```



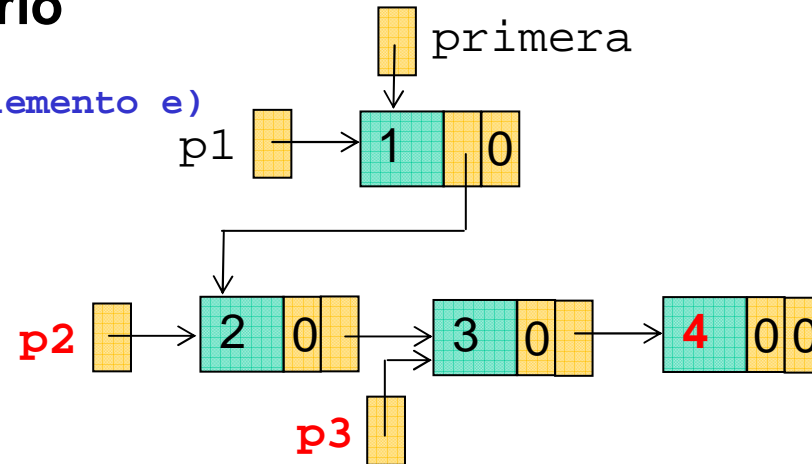


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
void insertaDerecho(Posicion p, Elemento e)
{
    p->der = new Celda(e,0,p->der);
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
a.insertaDerecho(p2,3);
Posicion p3 = p2.derecho();
a.insertaDerecho(p3,4);
```

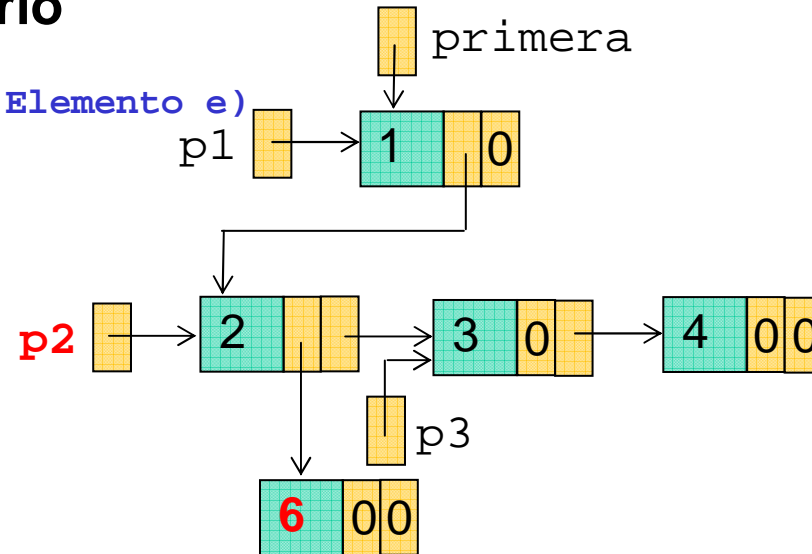


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
void insertaIzquierdo(Posicion p, Elemento e)
{
    p->izq = new Celda(e,0,p->izq);
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
a.insertaDerecho(p2,3);
Posicion p3 = p2.derecho();
a.insertaDerecho(p3,4);
a.insertaIzquierdo(p2,6);
```

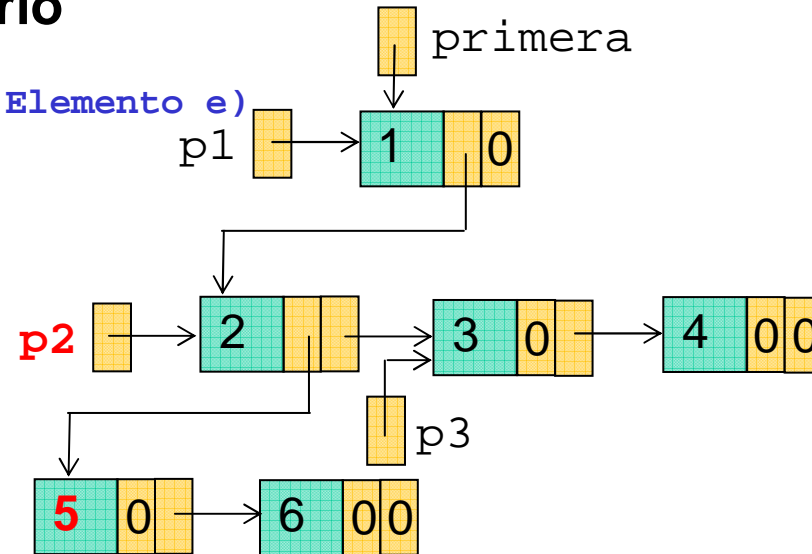


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
void insertaIzquierdo(Posicion p, Elemento e)
{
    p->izq = new Celda(e,0,p->izq);
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
a.insertaDerecho(p2,3);
Posicion p3 = p2.derecho();
a.insertaDerecho(p3,4);
a.insertaIzquierdo(p2,6);
a.insertaIzquierdo(p2,5);
```

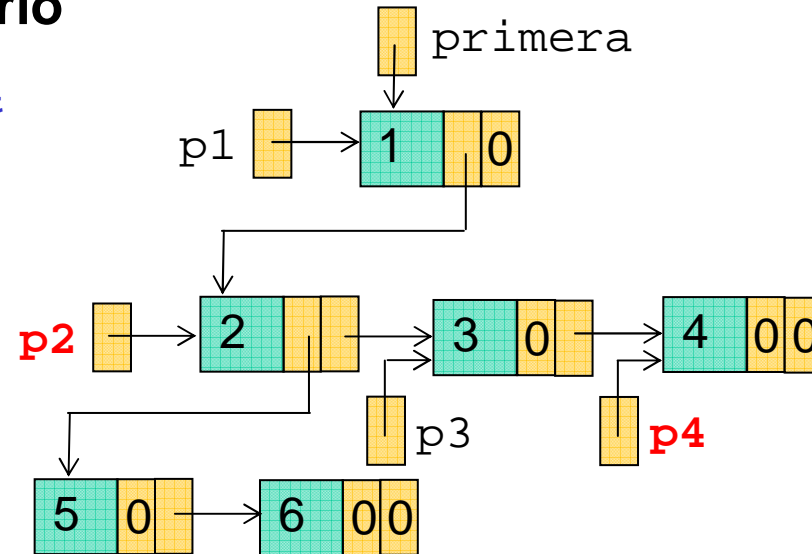


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
Posicion derecho(Posicion p) const
{
    return p->der;
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
a.insertaDerecho(p2,3);
Posicion p3 = p2.derecho();
a.insertaDerecho(p3,4);
a.insertaIzquierdo(p2,6);
a.insertaIzquierdo(p2,5);
Posicion p4 = p3.derecho();
```

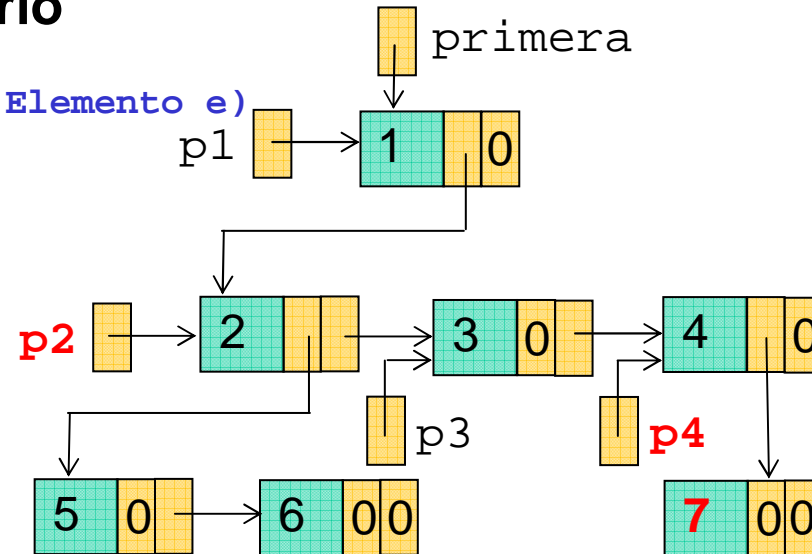


## 5.2 Árboles N-arios

### Construcción de un Árbol n-ario

```
void insertaIzquierdo(Posicion p, Elemento e)
{
    p->izq = new Celda(e,0,p->izq);
}
```

```
Arbol a(1);
Posicion p1 = a.inicio();
a.insertaIzquierdo(p1,2);
Posicion p2 = p1.izquierdo();
a.insertaDerecho(p2,3);
Posicion p3 = p2.derecho();
a.insertaDerecho(p3,4);
a.insertaIzquierdo(p2,6);
a.insertaIzquierdo(p2,5);
Posicion p4 = p3.derecho();
a.insertaIzquierdo(p4,7);
```



## 5.2 Árboles N-arios

### Representación Enlazada Hijo Izquierdo – Hermano Derecho – Padre

```
struct Celda
{
    Elemento elem;
    Celda * izq, * der, * padre;
    Celda(Elemento e, Celda *i, Celda *d, Celda *p) { elem=e; izq=i; der=d; padre=p; }
    ~Celda();
    Celda(const Celda & c);
};

Celda::~~Celda()
{
    if (izq!=0) delete izq;
    if (der!=0) delete der;
}

Celda::Celda(const Celda & c)
{
    elem = c.elem;
    if (c.izq==0) izq = 0; else { izq = new Celda(*c.izq); izq->padre = this; }
    if (c.der==0) der = 0; else { der = new Celda(*c.der); der->padre = this; }
    padre = 0;
}
```



## 5.2 Árboles N-arios

### Representación Enlazada Hijo Izquierdo – Hermano Derecho – Padre

```
Arbol(Elemento e)           { primera = new Celda(e,0,0,0); }
~Arbol()                   { delete primera; }
Arbol(const Arbol & a)      { primera = new Celda(*a.primera); }

Arbol & operator=(const Arbol & a);

Arbol & Arbol::operator=(const Arbol & a)
{
    delete primera;
    primera = new Celda(*a.primera);
    return * this;
}
Arbol & Arbol::operator=(const Arbol & a)
{
    Arbol aux(a);
    swap(primeras,aux.primera);
    return * this;
}
```

## 5.2 Árboles N-arios

### Representación Enlazada Hijo Izquierdo – Hermano Derecho – Padre

```
Posicion inicio()                const { return primera; }
Posicion fin()                   const { return 0; }
Posicion izquierdo(Posicion p)   const { return p->izq; }
Posicion derecho(Posicion p)     const { return p->der; }
Posicion padre(Posicion p)     const { return p->padre; }

Elemento recupera(Posicion p)    const { return p->elem; }
void asigna(Posicion p, Elemento e) { p->elem = e; }

void insertaIzquierdo(Posicion p, Elemento e) {
    p->izq = new Celda(e,0,p->izq,p);
}
void insertaDerecho(Posicion p, Elemento e) {
    p->der = new Celda(e,0,p->der,p->padre);
}
void supprimeIzquierdo(Posicion p) { Celda * c = p->izq;
                                     p->izq = p->izq->der;
                                     delete c; }
void supprimeDerecho(Posicion p) { Celda * c = p->der;
                                     p->der = p->der->der;
                                     delete c; }
```





## 5.2 Árboles N-arios

### Árbol n-ario con Excepciones

```
struct posicion_erronea
{
    char * error() { return "Posicion incorrecta"; }
};
```

## 5.2 Árboles N-arios

### Árbol n-ario con Excepciones

```
Posicion izquierdo(Posicion p) const
{
    if (p==0) throw posicion_erronea();
    return p->izq;
}
```

```
Posicion derecho(Posicion p) const
{
    if (p==0) throw posicion_erronea();
    return p->der;
}
```

```
Posicion padre(Posicion p) const
{
    if (p==0) throw posicion_erronea();
    return p->padre;
}
```

## 5.2 Árboles N-arios

### Árbol binario con Excepciones

```
Elemento recupera(Posicion p) const
{
    if (p==0) throw posicion_erronea();
    return p->elem;
}

void asigna(Posicion p, Elemento e)
{
    if (p==0) throw posicion_erronea();
    p->elem = e;
}
```

## 5.2 Árboles N-arios

### Árbol n-ario con Excepciones

```
void insertaIzquierdo(Posicion p, Elemento e)
{
    if (p==0) throw posicion_erronea();
    p->izq = new Celda(e,0,p->izq,p);
}
```

```
void insertaHijoDerecho(Posicion p, Elemento e)
{
    if ((p==0)|| (p==primera)) throw posicion_erronea();
    p->der = new Celda(e,0,p->der,p->padre);
}
```

## 5.2 Árboles N-arios

### Árbol n-ario con Excepciones

```
void supprimeIzquierdo(Posicion p)
{
    if ((p==0) || (p->izq==0)) throw posicion_erronea();
    Celda * c = p->izq;
    p->izq = p->izq->der;
    delete c;
}
```

```
void supprimeDerecho(Posicion p)
{
    if ((p==0) || (p->der==0)) throw posicion_erronea();
    Celda * c = p->der;
    p->der = p->der->der;
    delete c;
}
```

