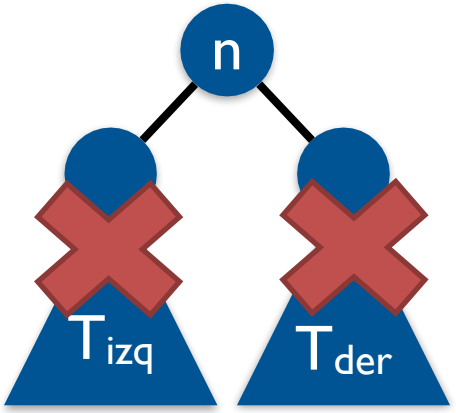
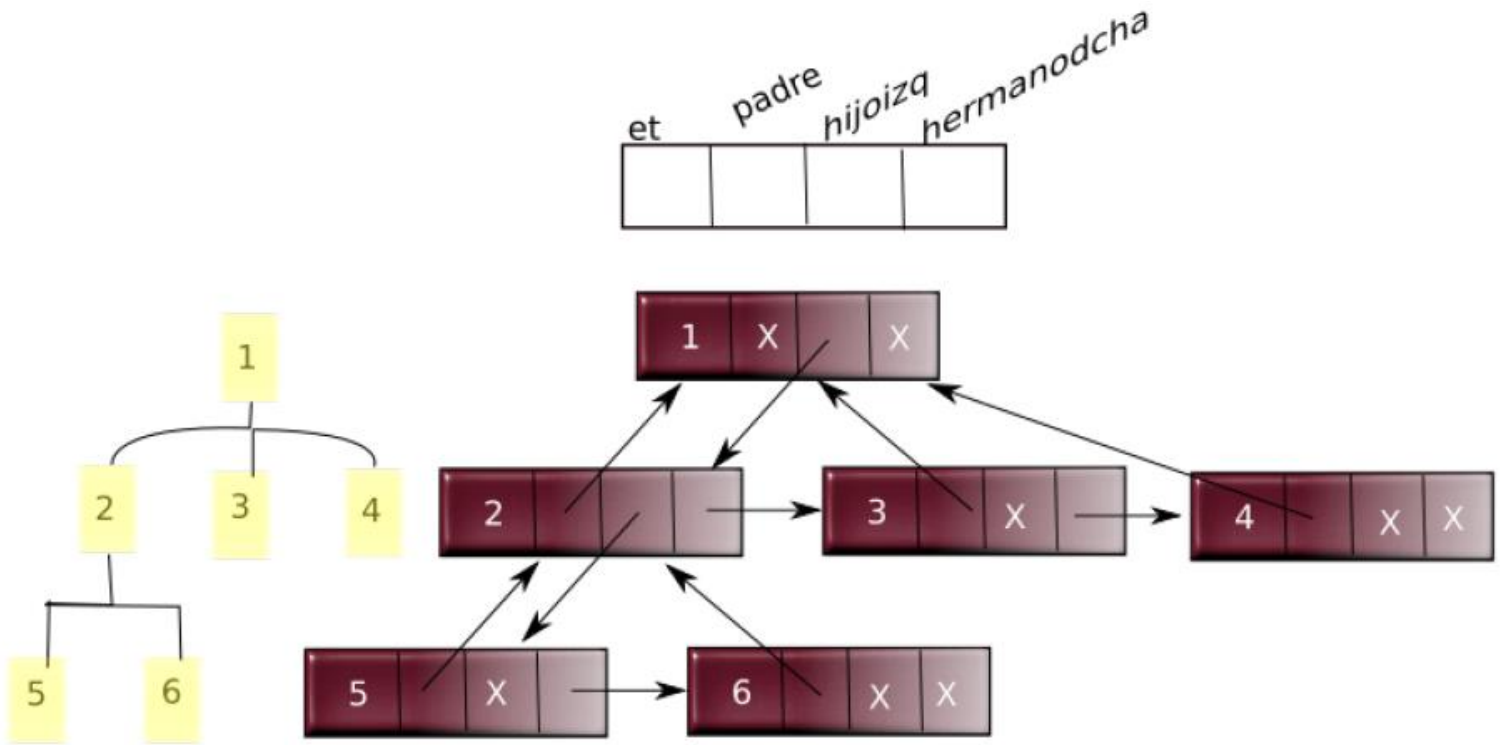
Árboles Generales

- Hasta ahora hemos estudiado los árboles binarios y su representación
- ¿Cómo se representaría un árbol con un número indeterminado de nodos (n-ario)?



Árboles Generales

Para representar un árbol general, cada nodo contendrá su etiqueta y punteros al padre, al hijo a la izquierda y al hermano a la derecha



Árboles Generales

Para representar un árbol general, cada nodo contendrá su etiqueta y punteros al padre, al hijo a la izquierda y al hermano a la derecha

```
1  template <class T>
2  struct info_nodo {
3        T et;
4        info_nodo<T> * padre, * hijoizq, * hermanodcha;
5        info_nodo() {
6              padre = hijoizq = hermanodcha = 0;
7        }
8        infonodo(const T & e) {
9              et = e;
10             padre = hijoizq = hermanodcha = 0;
11       }
12 }
```

Árboles Generales

La representación de *ArbolGeneral* sería de la siguiente forma

```
1  typename <class T>
2  class ArbolGeneral{
3   private:
4      info_nodo<T> *raiz;
5
6      //aqui todas las funciones privadas
7
8    public:
9      //aqui la interfaz de ArbolGeneral
10
11
12     ...
13 };
```

Árboles Generales

Copiar

```
1  template <class T>
2  void ArbolGeneral<T>::Copiar(info_nodo<T>* s, info_nodo<T>* &d) {
3        if (s==0)
4              d = 0;
5
6        else {
7              d = new info_nodo<T>(s->et);
8              Copiar(s->hijoizq,d->hijoizq);
9              Copiar(s->hermanodcha,d->hermanodcha);
10             // le asignamos a los nodos que hemos copiado su padre
11             if (d->hijoizq != 0){
12               d->hijoizq->padre=d;
13               for (info_nodo<T> aux = d->hijoizq->hermanodcha;
14                    aux!=0;aux= aux->hermanodcha)
15                    aux->padre= d;
16             }
17       }
18 }
```

Árboles Generales

- De esta forma el conjunto de nodos origen se indica por el puntero que tiene la variable *s*
- *E*l conjunto de nodos destino se identifican desde la variable *d*
- En primer lugar, si *s* no apunta a nada (es decir 0) entonces *d* también lo hará
- En otro caso se solicita nueva memoria para un *info_nodo* con igual etiqueta que la etiqueta que contiene el nodo al que apunta *s*
- Recursivamente se aplica el procedimiento de copiar para el hijo más a la izquierda y el hermano a la derecha
- Un vez realizada la copia hace falta ajustar los padres de los hijos del nodo al que apunta *d*
- Para ello se ejecutan las líneas 11-15

Árboles Generales

Destruir

Este método libera la memoria en primer lugar de todos lo hijos y, a continuación, la memoria de los hermanos a la derecha

```
1  template<class T>
2  void ArbolGeneral<T>::Destruir (info_nodo<T>* t) {
3    // Debemos empezar con el hermano a la derecha del ultimo nodo hoja
4    // del arbol, si no lo hacemos en este orden, perdemos los enlaces.
5    // Es decir, para destruir el arbol tenemos que hacerlo en recorrido
6    // postorden
7    if (t != 0) {
8      Destruir(t->hijoizq);          // cada hijo resuelve su destruccion
9      Destruir(t->hermanodcha);      // antes de hacer el delete
10     delete t;
11   }
12   // cuando t es cero no entra al if, vuelve a la llamada recursiva
13   // y hace el siguiente paso.
14 }
```

Árboles Generales

Constructor Copia, Destructor, Operador Asignación

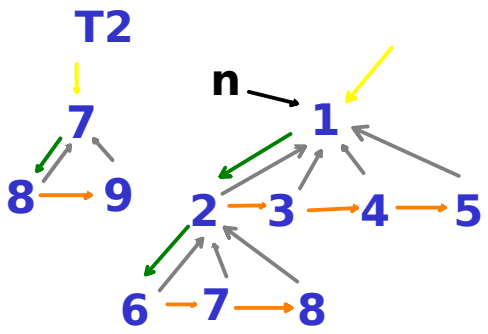
```
1  template<class T>
2  ArbolGeneral<T>::ArbolGeneral(const ArbolGeneral<T> & ag){
3    Copiar(ag.raiz,raiz);
4  }
5
6  template<class T>
7  ArbolGeneral<T>::~ArbolGeneral(){
8    Destruir(raiz);
9  }
10
11
12 template<class T>
13 ArbolGeneral<T> & ArbolGeneral<T>::operator=(const ArbolGeneral<T> & ag){
14   if (this!=&ag){
15     Destruir(raiz);
16     Copiar(ag.raiz,raiz);
17  }
18  return *this;
19 }
```

Árboles Generales

InsertarHijoIzquierda

El actual hijo más a la izquierda del nodo pasa a ser el hermano a la derecha del nuevo hijo a la izquierda

```
1  template <class T>
2  void ArbolGeneral<T>::InsertarHijoIzquierda (info_nodo<T>* n, info_nodo<T>* &t2) {
3    // El hijo a la izquierda de n pasaria a ser el hermano a la derecha
4    // de t2
5    if (t2 != 0) {
6          t2->hermanodcha = n->hijoizq;
7          t2->padre=n;
8          n->hijoizq=t2;
9          t2=0;
10
11   }
12 }
```



Árboles Generales

InsertarHijoDerecha

El hermano derecha actual pasa a ser el hermano derecha del nuevo nodo

```
1  template <class T>
2  void ArbolGeneral<T>::InsertarHermanoDerecha (info_nodo<T>* n,
3                                                info_nodo<T>* &t2) {
4    if (t2 != 0) {
5          t2->hermanodcha = n->hermanodcha;
6          t2->padre = n;
7          n->hermanodcha = t2;
8          t2 = 0;
9    }
10 }
```

Árboles Generales

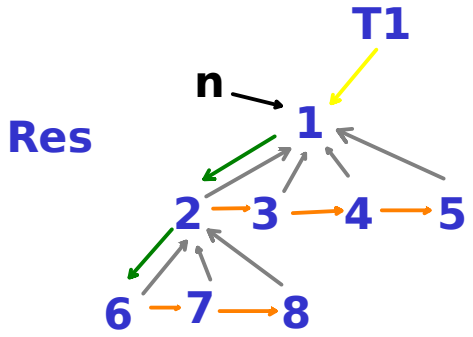
PodarHijoIzquierda

- Elimina del árbol todo lo que cuelga a partir del hijo más a la izquierda de un nodo dado
- El nodo adoptará como nuevo hijo más a la izquierda el hermano del hijo a la izquierda que se quita
- Adicionalmente el subárbol hijo más a la izquierda que se quita se devuelve como un árbol

Árboles Generales

PodarHijoIzquierda

```
1  template <class T>
2  info_nodo<T>* ArbolGeneral<T>::PodarHijoIzquierda (info_nodo<T>* n) {
3    info_nodo<T>* res = 0;  // creamos un nodo auxiliar
4    if (n->hijoizq != 0) {
5      // res apunta al subarbol hijo izquierda
6      res = n->hijoizq;
7      // el hijo a la izquierda del padre
8      // pasa a ser el hermano a la derecha
9      // del que era hijo a la izquierda
10     n->hijoizq = res->hermanodcha;
11     // y el hijo a la izquierda queda como
12     // la raiz del arbol a devolver
13     res->padre = res->hermanodcha=0;
14
15   }
16   return res;
17 }
```



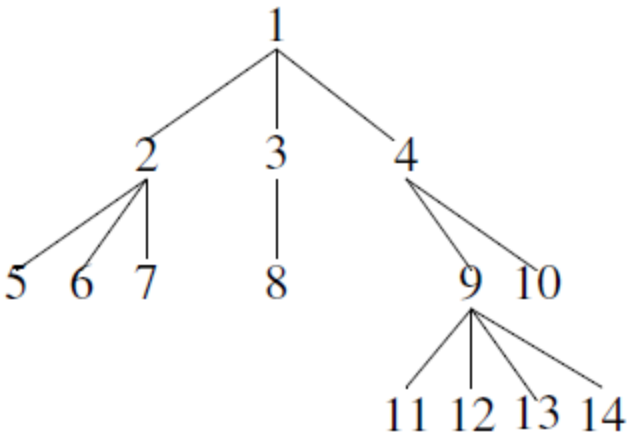
Árboles Generales

Altura

```
1  template <class T>
2  int ArbolGeneral<T>::altura (info_nodo<T>* t) {
3    // las hojas tendran altura cero y la raiz la altura maxima
4    if (t == 0)
5          return -1;
6
7    else {
8          int max = -1;
9          info_nodo<T>* aux;
10         // recorremos los hijos del nodo
11         for (aux=t->hijoizq;aux!=0;aux=aux->hermanodcha) {
12           // comprobamos si la altura de los hijos es mayor a la maxima
13           // que tenemos ya calculada
14
15           //altura del nodo
16          int alturahijo = altura(aux);
17
18          if (alturahijo > max)
19                max = alturahijo;
20        }
21         // la altura de los hijos mas 1 por el padre
22        return max+1;
23   }
24 }
```

Árboles Generales

Recorridos



Preorden: 1 2 5 6 7 3 8 4 9 11 12 13 14 10
Inorden: 5 2 6 7 1 8 3 11 9 12 13 14 4 10
Postorden: 5 6 7 2 8 3 11 12 13 14 9 10 4 1
Niveles: 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Árboles Generales

Recorridos – Preorden

```
1  template <class T>
2  void ListarPreorden (info_nodo<T>* t) {
3        if (t != 0) {
4              cout << t->et << ' ';    // primero listamos la raiz
5              info_ndo<T>* aux;        // y luego sus hijos
6              for (aux=t->hijoizq;aux!=0;aux=aux->hermanodcha)
7                    ListarPreorden(aux);
8        }
9  }
```

Árboles Generales

Recorridos – Inorden

```
11  template <class T>
12  void ListarInorden (info_nodo<T>* n) {
13        if (n != 0) {
14              ListarInorden (n->hijoizq); // listamos el hijo a la izquierda
15              cout << n->et << ' ';       // despues la raiz
16              info_nodo<T>* aux=n->hijoizq;
17              if (aux != 0) {
18                    aux = aux->hermanodcha; // y luego los hijos a la
19                    while (aux!=0) {        // derecha
20                          ListarInorden(aux);
21                          aux = aux->hermanodcha;
22                    }
23              }
24        }
25  }
```

Árboles Generales

Recorridos – Postorden

```
27  template <class T>
28  void ListarPostorden (info_nodo<T>* n) {
29        if (n != 0) {
30              info_nodo<T>* aux;
31              for (aux = n->hijoizq; aux != 0; aux = aux->hermanodcha)
32                    ListarPostorden (aux);
33
34              cout << n->et << ' ';
35        }
36  }
```

Árboles Generales

Recorridos – Por Niveles

```
38 // para la siguiente funcion debemos haber hecho en la cabecera un
39 // #include <queue>
40 template <class T>
41 void ListarNiveles (info_nodo<T>* n) {
42       // imprimimos un nodo y despues guardamos en la cola a sus hijos
43       if (n != 0) {
44             queue<info_nodo<T>* > c;
45             c.push(n);
46             while (!c.empty()) {
47                   info_nodo<T>* aux = c.front();
48                   c.pop();
49                   cout << aux->et << ' ';
50                   for (aux=aux->hijoizq;aux!=0;aux=aux->hermanodcha)
51                         c.push(aux);
52             } // cuando la cola quede vacia
53               //se termina el listado por niveles
54       }
55 }
```

Árboles Generales

Size

```
1  template <class T>
2  int size (info_nodo<T>* n) {
3        if (n == 0)
4              return 0;
5
6        else {
7              int nt = 1; // al menos hay un nodo
8              info_nodo<T>* aux;
9              for (aux=n->hijoizq;aux!=0;aux=aux->hermanodcha)
10                   nt += size(aux);
11
12             return nt;
13       }
14 }
```

Árboles Generales

Iguales

```
15 template <class T>
16 bool iguales (info_nodo<T>* t1, info_nodo<T>* t2) {
17   if (t1==0 && t2==0)
18         return true; // ambos son arboles vacios
19
20   else {
21         if (t1 == 0 || t2 == 0)
22               return false; // uno es vacio y el otro no
23
24         else {
25           if (t1->et != t2->et)
26                 return false;
27
28           else {
29                 info_nodo<T>* aux1, *aux2;
30                 bool igual = true;
31                 for (aux1=t1->hijoizq;
32                 aux2=t2->hijoizq;igual && aux1!=0 && aux2!=0;
33                       aux1=aux->hermanodcha;aux2=aux2->hermanodcha) {
34
35                       igual = iguales(aux1,aux2);
36                 }
```

Árboles Generales

Iguales

```
37                  // ahora bien, puede ser que un arbol este contenido en otro,
38                  // es decir, su tamanio sea diferente, por lo que comprobamos
39                  // si los dos han terminado
40                  return igual && aux1==0 && aux2==0;
41              }
42          }
43      }
44  }
```