LECCIÓN 18: Arboles Binarios

Arboles Binarios (AB)

1) Son arboles tal que cada nodo puede tener 0, 1 o 2 hijos.

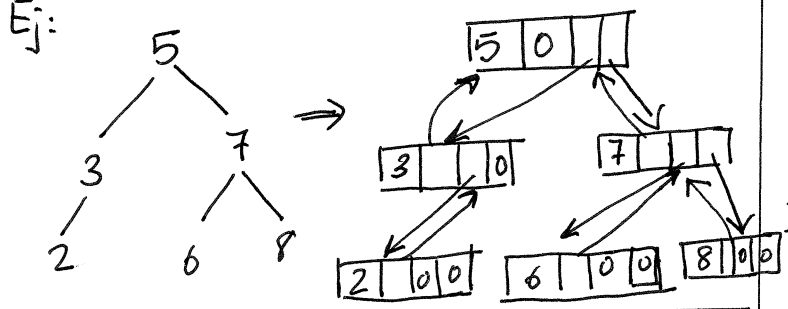
2) El árbol vacío es un AB.

Representacion

- APROXIMACIÓN 1: Sólo con struct_info_nodo

```
template <class T>
struct info_nodo{
    T et;
    info_nodo * padre, * hizq, hder;
    info_nodo(){padre=hizq=hder=0;}
    info_nodo( const T &e) {
et padre hder   et=e;
e|0|0|0   padre=hizq=hder=0
hizq    }
```

Ej:



FUNCIONES

- Get: Padre, Hi, Hd . iguales . borrar . copiar.
- Insertar: Hi, Hd. Versiones con un simple valor o con un subarbol.
- Podar: Hi, Hd. Borrar sin mas o borrar devolviendo el subarbol borrado.
- Recorridos: Preorden, Postorden, Inorden, Por Niveles.
- Leer/Escribir.

```
template <class T>
info_nodo <T>* GetPadre (info_nodo <T>*n)
{
    return n->padre;
}
template <class T>
info_nodo <T>* GetHi (info_nodo <T>*n)
{
    return n->hizq;
}
template <class T>
info_nodo <T>* GetHd (info_nodo <T>*n)
{
    return n->hder;
}
template <class T>
void InsertarHijoIzquierda (info_nodo<T>
                            * n,
                            info_nod <T>
                            * sub) {
    info_nodo <T> * aux = n->hizq;
    if (sub != 0){
        n->hizq = sub;
        sub->padre = n;
    }
    BorrarInfo (aux);
}
else {
    BorrarInfo (aux);
    n->hizq = 0;
}
}
template <class T>
void BorrarInfo ( info_nodo <T>*n){
    if (n != 0){
        BorrarInfo (n->hizq);
        BorrarInfo (n->hder);
        delete n;
    }
}.
```

LECCION = Arboles Binarios

```
template <class T>
void InsertarHijoDerecha(info_nodo<T>*n,
                         info_nodo<T>*sub){
info_nodo<T>*aux=n->hder;
    if (sub!=0){
        BorrarInfo(n->hder);
        n->hder=sub
        n->hder->padre=n;
    }
    else{
        BorrarInfo(aux);
        n->hder=0;
    }
}
template <class T>
void InsertarHijoIzquierda(info_nodo<T>*n,
                           const T&v){
  info_nodo<T>*aux=new info_nodo(v);
  InsertarHijoIzquierda(n,aux);
}
template <class T>
void InsertarHijoDerecha(info_nodo<T>*n,
                         const T&v){
    info_nodo<T>*aux=new info_nodo(v);
    InsertarHijoDerecha(n,aux);
}
template <class T>
void PodarHijoIzquierda(info_nodo<T>*n){
    if (n->hizq!=0){
        BorrarInfo(n->hizq);
        n->hizq=0;
    }
}
template <class T>
info_nodo<T>*PodarHijoIzq-GetSubtree(
                    info_nodo<T>*n){
  info_nodo<T>*aux=n->hizq;
  n->hizq=0;
  if(aux!=0)
  aux->padre=0;
  return aux;
}
template <class T>
void Copiar(info_nodo<T>*s, info_nodo<T>
            *&d){
    if(s==0){
      d=0;
    }
    else{
      d=new info_nodo<T>(s->et);
      Copiar(s->hizq,d->hizq);
      Copiar(s->hder,d->hder);
      if(d->hizq!=0)
      d->hizq->padre=d;
      if(d->hder!=0)
      d->hder->padre=d;
    }
}
template <class T>
void RecorridoPreorden(ostream&os,
                       const info_nodo<T>
                       *n){
  if(n!=0){
    os<<n->et<<" ";
    RecorridoPreorden(os,n->hizq);
    RecorridoPreorden(os,n->hder);
  }
}
```

LECCION : Arboles Binarios

```
template <class T>
void Recondo Inorden(ostream &os,
                     const infonodo<T>*n){

    if (n!=0){
        Recondo Inorden (os, n->hizq);
        os << n->et << ' ';
        Recondo Inorden (os, n->hder);
    }

}
template <class T>
void Recondo Postorden(ostream &os,
                       const infonodo<T>*n){

    if (n!=0){
        Recondo Postorden (os, n->hizq);
        Recondo Postorden (os, n->hder);
        os << n->et << ' ';
    }

}

template <class T>
void Recondo Nwoles(ostream &os,
                    const info_nodo<T>*n){

  if (n!=0){
    queue <const info_nodo<T>*> cola;
    cola.push (n);
    while(!cola.empty()){
      info_nodo<T>* p=cola.front();
      os<< p->et << ' ';
      if(p->hizq != 0)
          cola.push(p->hizq);
      if(p->hder != 0)
          cola.push(p->hder);

      cola.pop();
    }
}
```

```
template <class T>
bool iguales (info_nodo<T> *n1,
              info_nodo<T> *n2){

  if (n1 == 0 && n2 == 0)
      return true;
  if (n1 != 0 || n2 != 0)
      return false;
  if (n1->et == n2->et)
    return iguales(n1->hizq, n2->hizq)
           &&
           iguales(n1->hder, n2->hder);
  else return false.
}

template <class T>
int numero_nodos (info_nodo<T>*n){
    if (n == 0){
        return 0;
    else{
      return 1+numero_nodos(n->hizq)
             +numero_nodo(n->hder);
    }
}
```

```
//LEER Y ESCRIBIR
template <class T>
void Lee (istream &is, infonodo<T>*&n){

  char c;
  c= is.get();
  if (is){
    if (c=='x') n=0;
    else if (c=='n'){
          T e;
          is>>e;
          n= new info_nodo(e);
          Lee(is, n->hizq);
          Lee(is, n->hder);
          if (n->hizq != 0)
              n->hizq->padre=n;
          if (n->hder != 0)
              n->hder->padre=n;
    }
```

3.3

LECCION : Arboles Binarios

```
template <class T>
void Escribe(ostream &os, const infoNodo<T> *n){
    if (n==0)
        os << 'x';
    else {
        os << 'n' << n->et;
        Escribe(os, n->hizq);
        Escribe(os, n->hder);
    }
```

3.