Repaso Cadp Algoritmos

github.com/NahuelArn

Mas Abajo estan los algoritmos de Taller

Algoritmos de carga

```
Procedure agregarAtras(var L,Ult: lista; n: integer);
var
 nue: lista;
Begin
 new(nue);
 nue^.dato:= n;
 nue^.sig:= nil;
 if(L = nil)then
   L:= nue
 else
   Ult^.sig:= nue;
 Ult:= nue;
End;
```

```
Procedure agregarAdelante(var L: lista; n: integer);
Var
 nue: lista;
Begin
 new(nue);
 nue^.dato:= n;
 nue^.sig:= L;
 L:= nue;
End;
```


```
Procedure insertarOrdenado(var L: lista; n: integer);
Var
 nue: lista;
 ant,act: lista;
Begin
 new(nue);
 nue^.dato:= n;
 ant:= L;
 act:= L;
 While(act <> nil) and (n > act^.dato) do //ascendente
   begin
     ant:= act;
     act:= act^.sig;
   end
  if(act = ant) then  //principio o lista vacia
    L:= nue
  else  // al medio o al final
     ant^.sig:= nue;
 nue^sig:= act;
End;
```

A lo sumo: no excede X limite, pero puede ser menor o igual q el. Para ser true

AgregarAtras

```
procedure agregarAtras(var L,Ult: lista; d: cosa);
var
  nue: lista;
begin
  new(nue);
  nue^.dato:= d;
  nue^.sig:= nil;
  if(L = nil)do //si es el primer nodo
    L:= nue;
  else      // si no es el primer nodo
    Ult^.sig:= nue;
  Ult:= nue;
end;
```

AgregarAdelante

```
procedure agregarAdelante(var L: lista; d: cosa);
var
  nue: lista;
begin
  new(nue);
  nue^.dato:= d;
  nue^.sig:= L;
  L:= nue;
end;
```

InsertarOrdenado


```
procedure insertarOrdenado(var L: lista; d: cliente);
var
  nue: lista;
  ant,act: lista;
begin
  new(nue);
  nue^.dato:= d;
  ant:= L;
  act:= L;
  While(act <> nil)and(d.dni > act^.dato.dni);  // > ascendente | < descendente
    begin
      ant:= act;
      act:= act^.sig;
    end;
    if(act = ant)then //al principio o vacio
      L:= nue
    else
      ant^sig:= nue;
    nue^.sig:= act;
end;
```

Elimininar En Listas

github.com/NahuelArn

Hay 4 variantes del eliminar en listas

Desordenado

- Eliminar la Primera Ocurrencia en una Lista Desordenada
- Eliminar todas las Ocurrencias en una Lista Desordenada

Se tiene 1 5 2 10 3 1
Se busca borrar el 2
Queda 1 5 10 3 1

Se tiene 1 5 2 10 3 1
Se busca borrar el 1
Queda 5 2 10 3

Ordenado

- Eliminar la Primera Ocurrencia en una Lista Ordenada
- Eliminar todas las Ocurrencias en una Lista Ordenada

Se tiene 1 5 9 11 32
Se busca borrar el 6
Recorre solo hasta el 9. [6<9]

Se tiene 1 5 9 11 32 32
Se busca borrar el 32
Queda 1 5 9 11. Tambien cheque q el element actual no sea mas grande q el buscado

Elimininar En Listas

github.com/NahuelArn

Desordenado

Primera Ocurrencia

```
procedure eliminarLaPrimeraOcurrenciaEnUnaListaDesordenada(var L: lista; dni: integer; ok: boolean);
var
  ant,act: lista;
begin
  ant:= L;
  act:= L;
  ok:= false;
  While(act <> nil)and(dni <> act^.dato.dni)do;  // mientras no se encuentre el dni y no terminemos la lista
    begin
      ant:= act;
      act:= act^.sig;
    end;
    if(act <> nil)then //no es vacia la lista y encontre el nodo a eliminar
      begin
        ok:= true;
        if(act = L)then //el elemento a eliminar es el primero
          L:= act^.sig
        else  //el elemento es algun otro, pero no el primero
          ant^.sig:= act^.sig;
        dispose(act);  //se elima el actual
      end;
end;
```

TodasLasOcurrencias

```
procedure elimarTodasLasOcurrenciasDesordenado(var L: lista; valor: integer);
var
  ant,act: lista;
begin
  ant:= L;
  act:= L;
  While(act <> nil)do  // mientras no terminemos la lista
    begin
      if(valor <> act^.dato)then
      begin
        ant:= act;
        act:= act^.sig;
      end
      else //no es vacia la lista y encontre un nodo a eliminar
        begin
          if(act = L)then //el elemento a eliminar es el primero
            begin
              L:= act^.sig;
              ant:= L;
            end
          else  //el elemento es algun otro, pero no el primero
            ant^.sig:= act^.sig;
          dispose(act);  //se elima el actual
          act:= ant;
        end;
    end;
end;
```

Elimininar En Listas

github.com/NahuelArn

Ordenado

Primera Ocurrencia

```
procedure elimarLaPrimeraOcurrenciaListaOrdenada(var L: lista; dni: integer);
var
  ant,act: lista;
  contIteraciones: integer;
begin
  <!-- Writeln('Flag 0'); -->
  ant:= L;
  act:= L;
  <!-- contIteraciones:= 0; -->
  While(act <> nil)and(dni > act^.dato)do //ascendente // mientras no se encuentre el dni y el dni actual no sea mayor al busca
    begin
      ant:= act;
      act:= act^.sig;
      contIteraciones:= contIteraciones+1;
      <!-- Writeln('Flag 1'); -->
    end;
    <!-- Writeln('Flag 2'); -->
    if(act <> nil) and (dni = act^.dato)then //no es vacia la lista y encontre el nodo a eliminar
      begin
        if(act = L)then //el elemento a eliminar es el primero
          L:= act^.sig
        else  //el elemento es algun otro, pero no el primero
          ant^.sig:= act^.sig;
        dispose(act);  //se elima el actual
        //contIteraciones:= contIteraciones+1;
      end;
    <!-- Writeln('Cant Itereaciones ',contIteraciones); -->
  end;
```

TodasLasOcurrencias

```
  procedure buscarOrdenado(var act, ant: lista ; d : integer);
begin
  while (act <> nil) and (act^.dato < d) do begin
    ant := act;
    act := act^.sig;
  end;
end;

procedure eliminarNodo (var pri, act, ant: lista);
var
  aux : lista;
begin
  aux := act^.sig;
  if (act = pri) then
     pri := pri^.sig
  else
     ant^.sig := act^.sig;
  dispose(act);
  act := aux;
end;

procedure elimarTodasLasOcurrenciasOrdenado (var L : lista; d : integer);
var
  act, ant : lista;
begin
  act := L;
  ant := L;
  buscarOrdenado(act, ant, d);

  while (act <> nil) and (act^.dato = d) do
        eliminarNodo(L, act, ant);
end;
```

Vectores

github.com/NahuelArn

EliminarPosVector

```
Procedure eliminarPosVector(var v: vector; var ok: boolean; pos: integer; var dimL: integer);
var i: integer;
Begin
  ok:= ((pos > 0) and (pos <= dimL)); //verifico q la pos sea valida
    if(ok)then
      begin
        for i:= pos to (dimL-1) do //hasta menos 1 porq se "elima una posicion"
          begin
            v[pos]:= v[i+1]; //basicamente haces desplazamientos, 1:= [i+1=2]; 2:= [i+1=3] etc etc..
          end;
          dimL:= dimL-1; //decremento la dimL, ya q "borre un elemento del vector"
      end;
end;
```

InsertarEnVector

```
Procedure insertar(var v: vector; var ok: boolean; pos: integer; var dimL: integer; numOcosa: integer);
var i: integer;
Begin
  ok:= ((pos >= 1) and (pos <= dimL) and (dimL+1 <= dimF)); //verifico q la pos sea valida
    if(ok)then
      begin
        for i:= dimL downto pos do  //Arranco en la posDimL y voy hasta pos
          begin
            v[i+1]:= v[i]; //En la posActual+1, me cargo lo que hay en la posActual, son desplazamientos
          end;
          v[pos] := numOcosa; // Asigno el nuevo valor en la posición indicada
          dimL:= dimL+1; //incremento la dimL, ya q inserte un nuevo valor en el vector
      end;
end;
```

Vectores

github.com/NahuelArn

InsertarOrdenado

```
function buscarPosicion(v: vEmpleado; dimL: integer; cod: integer): integer;
var
  pos: integer;
begin
  pos:= 1;
  //lo va dejar ordenado tipo 1 2 3 4 5
  While(pos <= dimL) and (cod > v[pos].codPais)do
    begin
      pos:= pos+1;
    end;
  buscarPosicion:= pos;
end;

procedure insertarPosicion(var v: vEmpleado; var dimL: integer; pos: integer; e: empleado);
var
  i: integer;
begin
  if((pos >= 1) and (pos <= dimF2k) and (dimL+1 <= dimF2k)) then
    begin
      for i:= dimL downto pos do
        v[i+1]:= v[i];
      v[pos]:= e;
      dimL:= dimL+1;
    end;
end;

procedure insertarOrdenado(var v: vEmpleado; var dimL: integer; e: empleado);
var pos: integer;
begin
  pos:= buscarPosicion(v,dimL,e.codPais);
  insertarPosicion(v,dimL,pos,e);
end;

procedure cargarVector(var v: vEmpleado; var dimL: integer);
var
  e: empleado;
begin
  dimL:= 0;
  While(dimL < dimF2k)do
    begin
      leerEmpleados(e);
      insertarOrdenado(v,dimL,e);
    end;
end;
```

OrdenarVectorPorSeleccion

```
{
  este metodo busca en todo el array el minimo y lo va posicionando al principiendo uno a uno,
busco [i] si es mas chico q algun elemento del array me lo guardo en la iteracion que este i
}
procedure ordenarVector(var v: vOrdenar; dimL: integer);
var a,b,i,min: integer;
begin
  for i:= 1 to (dimL-1) do
    begin
      a:= i; //me paro en la [x posicion]
      for b:= i+1 to dimL do
        begin
          if(v[b] < v[a])then {pregunta si i+1 es mayor al primer campo, si es asi, cambia de lugar y asi se
            begin
              a:= b; //en A tengo guardado la posicion del minimo de todo el vector
            end;
          {aca hacen el swap, intercambia los valores en sus posiciones correspondientes
          Minmo lo guarda en la iteracion i, y lo que habia en i posicion lo intercambia en el lugar de la po
          min:= v[a]; //salvo el valor del minimo
          v[a]:= v[i]; //swap de valores
          v[i]:= min; //guardo en la pos i el valor minimo de todo el vector
        end;
    end;
end;
```

Ordena de menor a mayor < ‘
Ordena de mayor a menor > ‘

Busqueda Dicotomica | Busqueda Binaria

Con procedimiento

Con Funcion

```
Function buscarNumero(numAbuscar: integer; v1: vPrimero; dimL: integer): Boolean;
Var
  ini,mitad,fin: integer;
  encontrado: Boolean;
Begin
  encontrado := false;
  ini := 1;
  fin := dimL;
  While ((ini <= fin) And (encontrado <> true)) Do
    Begin
      mitad := (ini+fin) Div 2;
      If (v1[mitad] =numAbuscar) Then
        encontrado := true
      Else
        If (numAbuscar < v1[mitad])Then
          fin := mitad-1
      Else
        If (numAbuscar> v1[mitad])Then
        ini := mitad+1;
    End;
  buscarNumero := encontrado;
End;
```

Encontre el Valor Buscado

Se tira para la izquierda

Se tira para la derecha

Algoritmos de Taller

github.com/NahuelArn

Imperativo

- Ordenacion
- Recursion
- Arboles
- Merge

Si taller hablara: link

Algoritmos de Ordenacion

github.com/NahuelArn

Insercion

```
const
  PosIni = 0;
Procedure ordenarVectorInsercion(var vec: v; dimL: integer);
var
    pos : integer;
    elemAct,i,b: integer;
begin
    pos:= 0;
  for i := 1 to (dimL-1) do
    begin
      elemAct := vec[i];
      pos := i - 1;
      while( (Pos > (PosIni-1) )and (vec[pos] > elemAct)) do  //si el anterior es mas grande que el segundo
        begin
          vec[pos+1] := vec[pos];
          pos := pos - 1;
        end;
        vec[pos+1] := elemAct;
    end;
end;
```

Seleccion

```
procedure ordenarVectorSeleccion(var v: vOrdenar; dimL: integer);
var a,b,i,min: integer;
begin
  for i:= 1 to (dimL-1) do
    begin
      a:= i;
      for b:= i+1 to dimL do
        begin
          if(v[b] < v[a])then
            begin
              a:= b;
            end;
          min:= v[a];
          v[a]:= v[i];
          v[i]:= min;
        end;
    end;
end;
```

Recursion

github.com/NahuelArn

Con recursion(pila)

```
//usando un agregarAdelante con recursion nos queda un agregarAtras
procedure CargarLista (var l: lista);
var
  numero: integer;
  nuevo: lista;
Begin
  write ('Ingrese un numero: ');
  readln(numero);
  if (numero <> -1 )  then
    begin
      CargarLista (l);
      new (nuevo);
      nuevo^.dato:= numero;
      nuevo^.sig:= l;
      l:= nuevo;
    end
  else
    l:= nil
End;
```

Hasta q no llegue al -1, carga en la pila

llega el backtracking y lo deja como un agregarAtras

aca entraria solo si en la primera iteracion te ingresan el -1

Con “recursion”

```
//usando un agregarAdelante con recursion nos queda un agregarAdelante
procedure CargarLista (var l: lista);
var
  numero: integer;
  nuevo: lista;
Begin
  write ('Ingrese un numero: ');
  readln(numero);
  if (numero <> -1 )  then
    begin
        new (nuevo);
        nuevo^.dato:= numero;
        nuevo^.sig:= l;
        l:= nuevo;
        CargarLista (l);
    end;
End;
```

de cierto modo no usa "la pila"

en este caso si sale del if, no pasa nada(no hay recursion como tal)

Imprimiendo normal La lista con recursion(sin)

```
procedure imprimirLista(L: lista);
begin
  if(L <> nil)then
    begin
      Writeln('Lo que hay en la lista ',L^.dato);
      imprimirLista(L^.sig);
    end;
end;
```

```
Ingrese un numero: 1
Ingrese un numero: 2
Ingrese un numero: 3
Ingrese un numero: -1
Lo que hay en la lista 1
Lo que hay en la lista 2
Lo que hay en la lista 3
```

Imprimiendo La lista con recursion(con)

```
procedure imprimirLista(L: lista);
begin
  if(L <> nil)then
    begin
      imprimirLista(L^.sig);
      Writeln('Lo que hay en la lista ',L^.dato);
    end;
end;
```

```
Ingrese un numero: 1
Ingrese un numero: 2
Ingrese un numero: 3
Ingrese un numero: -1
Lo que hay en la lista 3
Lo que hay en la lista 2
Lo que hay en la lista 1
```

Sin

```
procedure imprimirLista(L: lista);
begin
  if(L <> nil)then
    begin
      Writeln('Lo que hay en la lista ',L^.dato);
      imprimirLista(L^.sig);
    end;
end;
```

```
Ingrese un numero: 1
Ingrese un numero: 2
Ingrese un numero: 3
Ingrese un numero: -1
Lo que hay en la lista 3
Lo que hay en la lista 2
Lo que hay en la lista 1
```

Con

```
procedure imprimirLista(L: lista);
begin
  if(L <> nil)then
    begin
      imprimirLista(L^.sig);
      Writeln('Lo que hay en la lista ',L^.dato);
    end;
end;
```

```
Ingrese un numero: 1
Ingrese un numero: 2
Ingrese un numero: 3
Ingrese un numero: -1
Lo que hay en la lista 1
Lo que hay en la lista 2
Lo que hay en la lista 3
```

Recursion

github.com/NahuelArn

Procedimiento

Funcion

```
                                V(integer)    x              1
procedure incrementarVector(var v: vector;dimL: integer; i: integer);
begin
  if( i <= dimL)then
    begin
      incrementarVector(v,dimL,i+1);
      v[i]:= v[i]+1;
    end;
end;
```

Busqueda Dicotomica Recursiva

github.com/NahuelArn

```
procedure busquedaDicotomicaRecursiva (v: vector ; ini: integer ; fin: integer ; valorBuscado: integer ; var pos: integer;var encontrado:Boolean);
var
  mid: integer;
begin
  if (ini > fin) then
    pos:= -1
  else
    begin
      mid:= (ini + fin) div 2;
      if (valorBuscado = v[mid]) then
        begin
          pos:= mid;
          encontrado:= true;
        end
      else
        begin
          if (valorBuscado < v[mid]) then
            busquedaDicotomicaRecursiva(v, ini, (mid - 1), valorBuscado, pos,encontrado)
          else
            busquedaDicotomicaRecursiva(v, (mid + 1), fin, valorBuscado, pos,encontrado);
        end;
    end;
end;
```

Arboles estructura de carga

github.com/NahuelArn

```
type
  arbol = ^nodo;

  nodo = record
    dato: integer;
    hi: arbol;
    hd: arbol
  end;

procedure inicializarLista(var L: arbol);
begin
  L:= nil;
end;
```

```
procedure cargarArbol(var a: arbol; n: integer);
begin
  if(a = nil)then
    begin
      new(a);
      a^.dato:= n;
      a^.hi:= nil;
      a^.hd:= nil;
    end
  else
    begin
      if(n < a^.dato) then
        cargarArbol(a^.hi,n)
      else
        if(n > a^.dato)then
          cargarArbol(a^.hd,n);
    end;
end;
```

```
var
  L: arbol;
  num: integer;
  a,b: integer;
  cantCumplen: integer;
begin
  randomize;
  inicializarLista(L);
  Writeln('Ingresa un numero');
  readln(num);
  While(num <> 0)do
    begin
      cargarArbol(L,num);
      Writeln('Ingresa un numero');
      readln(num);
    end;
  imprimirArbolInOrder(L);
end.
```

Metodos de Impresion en Arboles

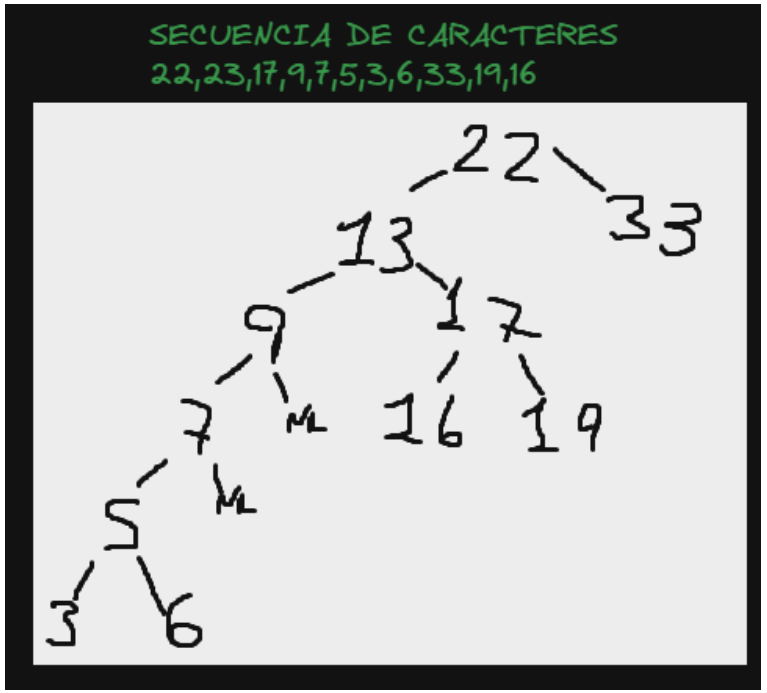
```
procedure imprimirPreOrden(a: arbol);
begin
  if(a <> nil)then
    begin
      Writeln('PreOrden: full izquierda, full Derecha(extremos)',a^.dato);
      imprimirPreOrden(a^.hi);
      imprimirPreOrden(a^.hd);
    end;
end;

procedure imprimirInOrder(a: arbol);
begin
  if(a <> nil)then
    begin
      imprimirInOrder(a^.hi);
      Writeln('In Order: de menor a mayor',a^.dato);
      imprimirInOrder(a^.hd);
    end;
end;
//espejo
procedure imprimirInOrder(a: arbol);
begin
  if(a <> nil)then
    begin
      imprimirInOrder(a^.hd);
      Writeln('In Order: de mayor a menor',a^.dato);
      imprimirInOrder(a^.hi);
    end;
end;


procedure imprimirPosOrder(a: arbol);
begin
  if(a <> nil)then
    begin
      imprimirPosOrder(a^.hi);
      imprimirPosOrder(a^.hd);
      Writeln('PosOrder: full derecha, full izquierda(extremos)',a^.dato);
    end;
end;
```

Impresion con cada Metodo

github.com/NahuelArn




```
0
PreOrden: full izquierda, full Derecha(extremos) 22
PreOrden: full izquierda, full Derecha(extremos) 13
PreOrden: full izquierda, full Derecha(extremos) 9
PreOrden: full izquierda, full Derecha(extremos) 7
PreOrden: full izquierda, full Derecha(extremos) 5
PreOrden: full izquierda, full Derecha(extremos) 3
PreOrden: full izquierda, full Derecha(extremos) 6
PreOrden: full izquierda, full Derecha(extremos) 17
PreOrden: full izquierda, full Derecha(extremos) 16
PreOrden: full izquierda, full Derecha(extremos) 19
PreOrden: full izquierda, full Derecha(extremos) 33
```

```
In Order: de menor a mayor 3
In Order: de menor a mayor 5
In Order: de menor a mayor 6
In Order: de menor a mayor 7
In Order: de menor a mayor 9
In Order: de menor a mayor 13
In Order: de menor a mayor 16
In Order: de menor a mayor 17
In Order: de menor a mayor 19
In Order: de menor a mayor 22
In Order: de menor a mayor 33
```

```
Espejo
In Order: de menor a mayor 33
In Order: de mayor a menor 22
In Order: de menor a mayor 3
In Order: de menor a mayor 5
In Order: de menor a mayor 6
In Order: de menor a mayor 7
In Order: de menor a mayor 9
In Order: de menor a mayor 13
In Order: de menor a mayor 16
In Order: de menor a mayor 17
In Order: de menor a mayor 19
```

```
PosOrder: full derecha, full izquierda(extremos) 3
PosOrder: full derecha, full izquierda(extremos) 6
PosOrder: full derecha, full izquierda(extremos) 5
PosOrder: full derecha, full izquierda(extremos) 7
PosOrder: full derecha, full izquierda(extremos) 9
PosOrder: full derecha, full izquierda(extremos) 16
PosOrder: full derecha, full izquierda(extremos) 19
PosOrder: full derecha, full izquierda(extremos) 17
PosOrder: full derecha, full izquierda(extremos) 13
PosOrder: full derecha, full izquierda(extremos) 33
PosOrder: full derecha, full izquierda(extremos) 22
```

Borrar un nodo en un Arbol

github.com/NahuelArn

```
procedure Borrar(x: elem; var a: arbol; var ok: boolean);
var
  aux: arbol;
begin
  if (a = nil) then
    ok := false
  else
    begin
      if (x < a^.dato) then // BUSCO EN EL SUBARBOL IZQUIERDO
        Borrar (x, a^.HI, ok)
      else
        begin
          If (x > a^.dato) then     // BUSCO EN EL SUBARBOL DERECHO
            Borrar (x, a^.HD, ok)
          else
            begin
              ok := true;
              if (a^.HI = nil) then
                begin                                // SOLO HIJO A DERECHA
                  aux := a;
                  a := a^.HD;
                  dispose(aux)
                end
              else
                if (a^.HD = nil) then
                  begin             // SOLO HIJO A IZQUIERDA
                    aux := a;
                    a := a^.HI;
                    dispose(aux)
                  end
                else
                  begin             // DOS HIJOS. REEMPLAZO CON EL MENOR DE LA DERECHA
                    aux := Minimo(a^.HD);
                    a^.dato = aux^.dato;
                    Borrar(a^.dato, a^.HD, ok);
                  end
            end
        end;
end;
```

Borrar un nodo en un Arbol

github.com/NahuelArn

Caso de q el nodo ha eliminar tenga un hijo en nil, ejemplo se quiere eliminar el 5

```
procedure Borrar(x: elem; var a: arbol; var ok: boolean);
var
  aux: arbol;
begin
  if (a = nil) then
    ok := false
  else
    begin
      if (x < a^.dato) then // BUSCO EN EL SUBARBOL IZQUIERDO
        Borrar (x, a^.HI, ok)
      else
        begin
          if (x > a^.dato) then     // BUSCO EN EL SUBARBOL DERECHO
            Borrar (x, a^.HD, ok)
          else
            begin
              ok := true;
              if (a^.HI = nil) then
                begin                          // SOLO HIJO A DERECHA
                  aux := a;
                  a := a^.HD;
                  dispose(aux)
                end
              else
                if (a^.HD = nil) then
                  begin                        // SOLO HIJO A IZQUIERDA
                    aux := a;
                    a := a^.HI;
                    dispose(aux)
                  end
                else
                  begin                        // DOS HIJOS. REEMPLAZO CON EL MENOR DE LA DERECHA
                    aux := Minimo(a^.HD);
                    a^.dato := aux^.dato;
                    Borrar(a^.dato, a^.HD, ok);
                  end
            end
        end;
end;
```

Busca El Nodo

Ya encontro el x buscado

Si el izquierdo es Nil
se guarda la direc de la pos actual;
y la pos actual cambia de direc a la del HD

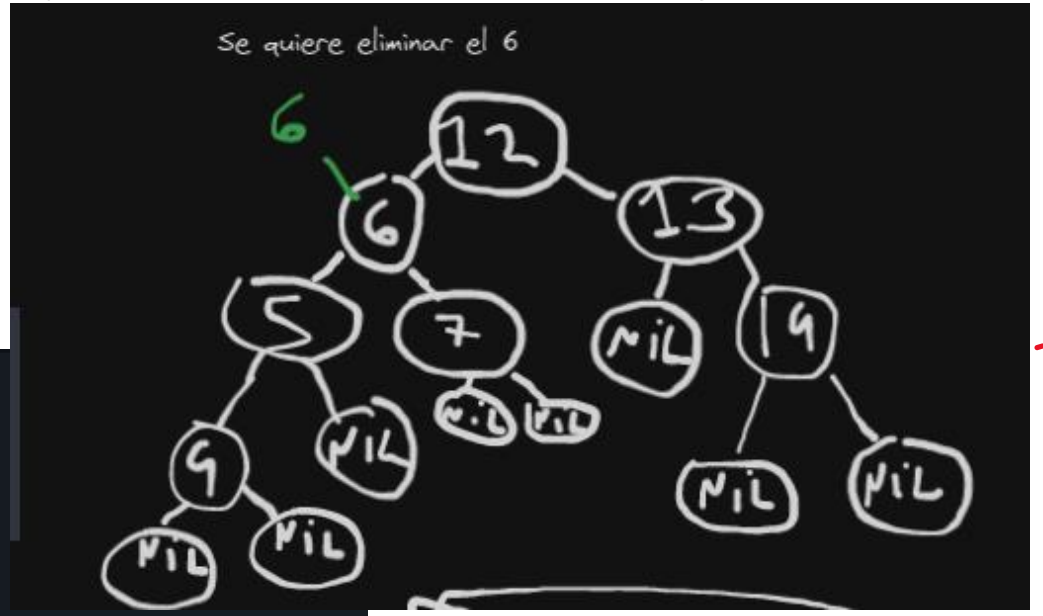
Cubren los 2 casos q un hijo es nil y si los 2 son nil

Mirar las imagens de izquierda a derecha para seguir como funciona

Borrar un nodo en un Arbol

github.com/NahuelArn

Caso de q el nodo ha eliminar sus hijos sean <> de nil




```
procedure Borrar(x: elem; var a: arbol; var ok: boolean);
var
  aux: arbol;
begin
  if (a = nil) then
    ok := false
  else
    begin
      if (x < a^.dato) then // BUSCO EN EL SUBARBOL IZQUIERDO
        Borrar (x, a^.HI, ok)
      else
        begin
          If (x > a^.dato) then     // BUSCO EN EL SUBARBOL DERECHO
            Borrar (x, a^.HD, ok)
          else
            begin
              ok := true;
              if (a^.HI = nil) then
                begin                              // SOLO HIJO A DERECHA
                  aux := a;
                  a := a^.HD;
                  dispose(aux)
                end
              else
                if (a^.HD = nil) then
                  begin             // SOLO HIJO A IZQUIERDA
                    aux := a;
                    a := a^.HI;
                    dispose(aux)
                  end
                else
                  begin             // DOS HIJOS. REEMPLAZO CON EL MENOR DE LA DERECHA
                    aux := Minimo(a^.HD);
                    a^.dato = aux^.dato;
                    Borrar(a^.dato, a^.HD, ok);
                  end
            end
        end;
end;
```

estas parado aca

```
IF (HI and HD no son Nil) them          aux: arbol;
  begin              // DOS HIJOS. REEMPLAZO CON EL MENOR DE LA DERECHA
    aux := Minimo(a^.HD);
    a^.dato = aux^.dato;
    Borrar(a^.dato, a^.HD, ok);
  end
nd
```

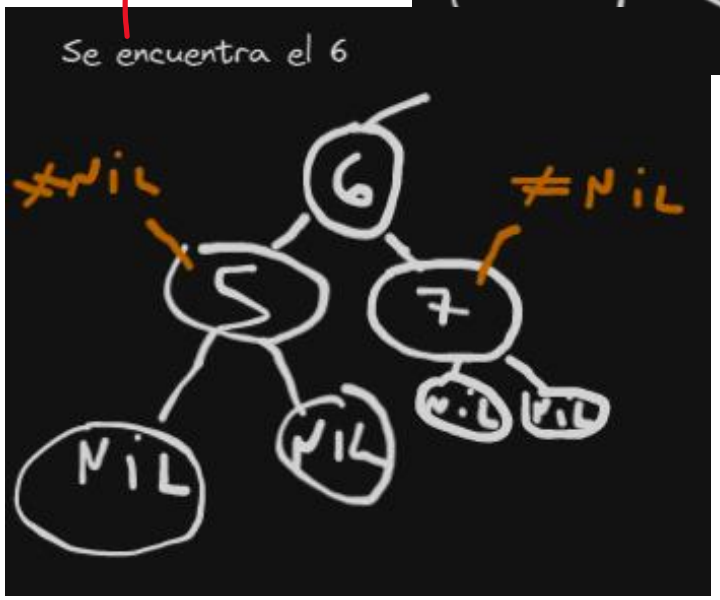
Mirar las imagens de izquierda a derecha para seguir como funciona

Borrar un nodo en un Arbol

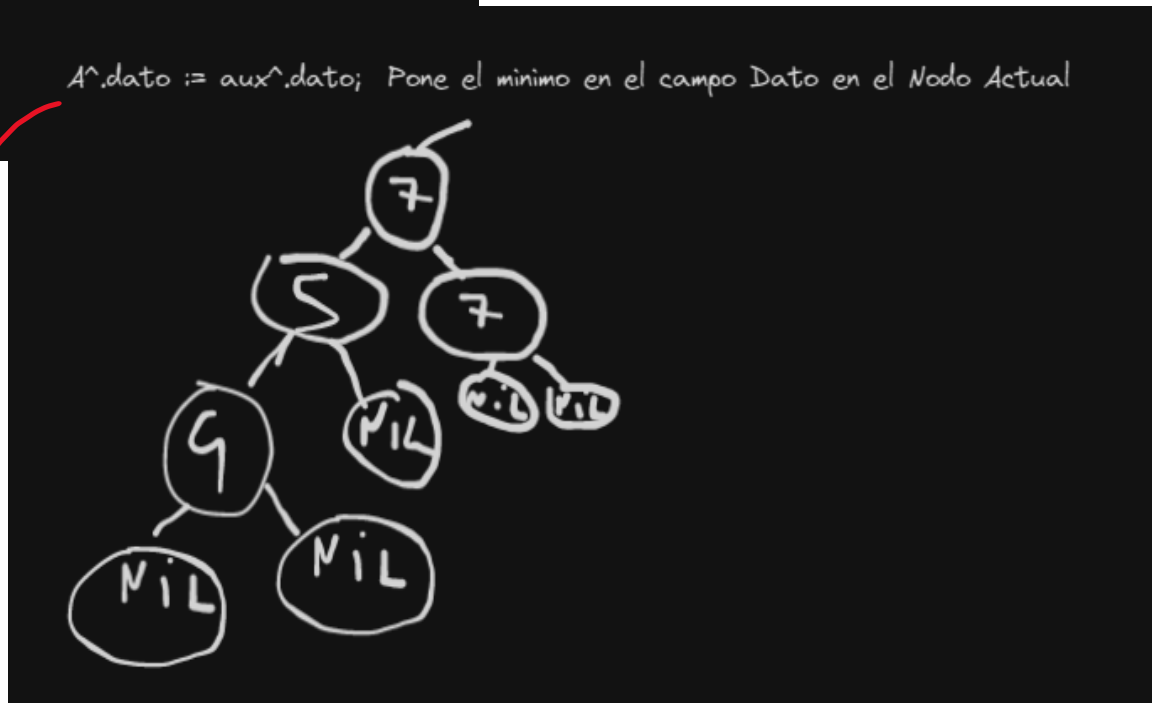
github.com/NahuelArn

Caso de q el nodo ha eliminar sus hijos sean <> de nil


```
function minimo(a: arbol): arbol;
begin
    if (a = nil) then
      minimo:= nil
    else
      begin
        if(a^.hi = nil)then
          minimo:= a
        else
          minimo:= minimo(a^.hi)
      end;
end;
```




```
begin                    // DOS HIJOS
  aux := Minimo(a^.HD);
  a^.dato = aux^.dato;
  Borrar(a^.dato, a^.HD, ok);
end
```



Mirar las imagens de izquierda a derecha para seguir como funciona

Borrar un nodo en un Arbol

Caso de q el nodo ha eliminar sus hijos sean <> de nil

github.com/NahuelArn

a^.hd = 7

7 } TRUE

```
procedure Borrar(x: elem; var a: arbol; var ok: boolean);
var
  aux: arbol;
begin
  if (a = nil) then
    ok := false
  else
    begin
      if (x < a^.dato) then // BUSCO EN EL SUBARBOL IZQUIERDO
        Borrar (x, a^.HI, ok)
      else
        begin
          If (x > a^.dato) then     // BUSCO EN EL SUBARBOL DERECHO
            Borrar (x, a^.HD, ok)
          else
            begin
              ok := true;
              if (a^.HI = nil) then
                begin
                  aux := a;
                  a := a^.HD;
                  dispose(aux)
                end
              else
                if (a^.HD = nil) then
                  begin            // SOLO HIJO A IZQUIERDA
                    aux := a;
                    a := a^.HI;
                    dispose(aux)
```

7 < 7

7 > 7

Busca El Nodo

NO ENTRA EN ESTAS 2

Estas paradado aca

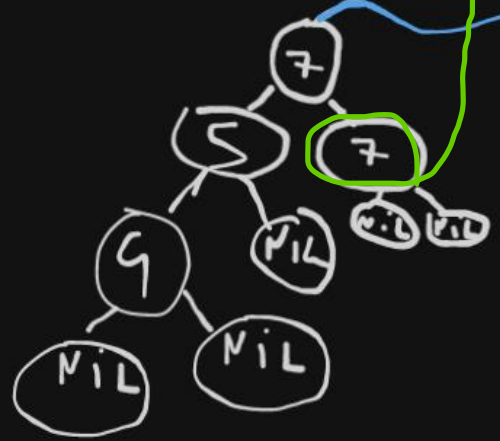
Ya encontro el x buscado

hi es = nil
Lo ELIMINA y listo!

Cubren los 2 casos q un hijo es nil y si los 2 son nil

// SOLO HIJO A DERECHA
Si el izquierdo es Nil
se guarda la direc de la pos actual;
y la pos actual cambia de direc a la del HD

a =
borrar(A^.dato,a^.hd,ok) = borrar(7, a^.hd,ok)




```
begin                    // DOS HIJOS.
  aux := Minimo(a^.HD);
  a^.dato = aux^.dato;
  Borrar(a^.dato, a^.HD, ok);
end
```