Árbol_B+

17/4/98

Especificación Arbol_B+

1	**Sintáctica:** creaArbol_B+(Entero+) Arbol_B+, insArbol_B+(Arbol_B+, Reg) Arbol_B+, eliArbol_B+(Arbol_B+, TipoClave) Arbol_B+, conArbol_B+(Arbol_B+, TipoClave) TipoResto, rango(Arbol_B+, TipoClave, TipoClave) ListaEnl[Reg], orden(Arbol_B+) Entero+, altura(Arbol_B+) Entero+, vacíoArbol_B+(Arbol_B+) Lógico, destruyeArbol_B+(Arbol_B+) .	-**creaArbol_B+()**: Crea un árbol_B+ vacío de orden m. -**insArbol_B+()**: Ingresa un nuevo registro al árbol_B+ en la posición que le corresponde según el orden ascendente de su clave. -**eliArbol_B+()**: Elimina el registro de clave dada en el árbol_B+, si existe. Si está vacío no hace ninguna eliminación. -**destruyeArbol_B+()**: Destruye el árbol_B+.
2	**Declaraciones:** Reg: r Entero+ : o TipoClave: c TipoResto: {TipoRestoNoDef}	-**vacíoArbol_B+()**: Regresa Verdadero si el árbol_B+ está vacío. -**conArbol_B+()**: Devuelve el resto del registro si él está en el árbol_B+, de lo contrario regresa {TipoRestoNoDef}.
3	**Semántica:** vacíoArbol_B+(creaArbol_B+(o)) = Verdadero vacíoArbol_B+(insArbol_B+(creaArbol_B+(o), r)) = Falso conArbol_B+(creaArbol_B+(o), c) = {TipoRestoNoDef} eliArbol_B+(creaArbol_B+(o)) = creaArbol_B+(o) altura(creaArbol_B+(o)) = 0 orden(creaArbol_B+(o)) = o	-**rango():** Devuelve una lista de registros que están en el rango solicitado. -**orden():** Regresa el orden del árbol_B+. -**altura():** Regresa la altura actual.

Arbol_B+ Clases: Entero+, Lógico, ApuntadorA, NodoB, TipoClave, TipoResto, Reg, ElePila, PilaEnl[ElePila]		
1	**Estructura**: privado: h : Entero+ = 0 nc : Entero+ = 0 m : Entero+ raiz : ApuntadorA NodoB = Nulo	**-h.** Altura actual del árbol_B+ **-nc**. Número actual de claves en el árbol. **-m.** Orden del árbol_B+ **-raiz**: Apuntador al nodo raíz. **-Arbol_B+()**. *Constructor*. Crea un árbol vacío de orden m. **-~Arbol_B+().** *Destructor.* Destruye el árbol .

2	**Operaciones:** público: Arbol_B+(Entero+: orden) ~Arbol_B+() insArbol_B+(Reg: r) eliArbol_B+(TipoClave: cl) conArbol_B+(TipoClave: cl): TipoResto rango(TipoClave: vi, vf): ListaEnl[Reg] vacíoArbol_B+(): Lógico numClaves(): Entero+ altura(): Entero+ orden(): Entero+ despliegue() **privado:** limpiarArbol_B+(ApuntadorA NodoB: pa) recorrer(ApuntadorA NodoB: pa, PilaEnl[ElePila]: p, TipoClave: cl): Lógico DividirNodo(ApuntadorA NodoB: pa, ph, TipoClave: clp): Lógico propagarFision(ApuntadorA NodoB: pa, ph, TipoClave: clp, PilaEnl[ElePila]: p) compactarNodo(ApuntadorA NodoB: pa, PilaEnl[ElePila]: p) Redistribucion(ApuntadorA NodoB: pa, pp, phi, phd, Entero+: p,c)	**-insArbol_B+()**. *Transformador*. Ingresa una nueva entrada en la posición que le corresponde según el orden ascendente de sus claves. **-eliArbol_B+()**. *Transformador*. Elimina una entrada de clave = cl, si el árbol no está vacío. **-conArbol_B+()**. *Observador*. Regresa el resto de la entrada si la clave existe. **-rango().** *Observador.* Regresa la lista de las entradas ∈[vi, vf] **-vacíoArbol_B+()**. *Observador*. Regresa Verdadero si el árbol está vacío. **-numClaves()**. *Observador*. Regresa el número actual de claves. **-altura()**. *Observador*. Regresa la altura actual de la estructura. **-orden()**. *Observador*. Regresa el orden del árbol_B+. **-despliegue().** *Observador*. Despliega el contenido del árbol. **-limpiarArbol_B+().** *Transformador.* Devuelve todos los nodos. **-recorrer().** *Observador.* Desciende en el árbol hasta encontrar la hoja donde debe ser insertado r, regresando el camino de descenso en la pila y la dirección del nodo hoja en pa. **-dividirNodo().** *Observador.* Divide un nodo en dos nodos hermanos, devolviendo el nodo fisionado en pa, su hermano en ph y la clave promocionada en clp. **-propagarFision().** *Observador.* Inserta la clave promocionada en el nodo padre, si éste desborda llama a dividirNodo() hasta que no haya desborde, regresando el nodo actual en pa. **-compactarNodo().** *Observador.* Compacta un nodo semi-vacío. **-redistribucion().** *Observador.* Redistribuye las entradas luego de una eliminación.

Implantación de la clase Arbol_B+

Reg Clases: TipoClave, TipoResto		
1	**Estructura**: privado: clave: TipoClave resto : TipoResto	**-clave:** Clave del registro. **-resto:** Resto del registro.
2	**Operaciones:** público: Reg() Reg(TipoClave: cl, TipoResto: r) Clave(): TipoClave Clave (TipoClave: cl) Resto(): TipoResto Resto(TipoResto: r) despliegue()	-**Reg()**. *Constructor*. Crea un registro con su información definida. -**Clave()**. *Observador y transformador*. Manejo de las claves del nodo. -**Resto()**. *Observador y transformador*. Manejo del resto del registro. -**despliegue()**. *Observador.* Despliega el contenido del registro.

TipoClave debe soportar todos los operadores relacionales { = , ≠ , < , ≤ , > , ≥ }

ElePila Clases: Entero+, ApuntadorA NodoB		
1	**Estructura**: privado: apu: ApuntadorA NodoB sub : Entero+	**-apu:** Puntero en la entrada. **-sub:** Posición de la entrada en el nodo.
2	**Operaciones:** público: ElePila() Apu(): ApuntadorA NodoB Apu (ApuntadorA NodoB: p) Sub(): Entero+ Sub(Entero+: s)	-**ElePila()**. *Constructor*. Crea un elemento de la pila. -**Apu()**. *Observador y transformador*. Manejo de los punteros. -**Sub()**. *Observador y transformador*. Manejo del subíndice.

NodoB Clases: Entero+, ApuntadorA, ArregloDe, TipoClave		
1	**Estructura**: privado: pro: Entero+ = 0 ne : Entero+ = 0 pIzq : ApuntadorA NodoB=Nulo pDer : ApuntadorA NodoB= Nulo cl : Arreglo[2m+1]De TipoClave p : Arreglo[2m+1]De ApuntadorA SinTipo	**-pro:** Profundidad del nodo. **-ne:** Número actual de entradas en el nodo. **-pIzq:** Dirección del hermano izquierdo. -**pDer**: Dirección del hermano derecho. **-m:** Orden del árbol_B+. **-cl:** Conjunto de claves del nodo. **-p:** Conjunto de apuntadores a los registros si pro = 0, ó a los nodos hijos si pro > 0.
2	**Operaciones:** público: NodoB() Pro(): Entero+ Pro(Entero+: p) Ne(): Entero+ Ne(Entero+: n) PIzq(ApuntadorA NodoB: izq) PIzq() : ApuntadorA NodoB PDer(ApuntadorA NodoB: der) PDer(): ApuntadorA NodoB Clave(Entero+: si): TipoClave Clave (Entero+: si, TipoClave: cl) Puntero(Entero+: si): ApuntadorA SinTipo Puntero(Entero+: si, ApuntadorA SinTipo: ptr)	-**NodoB()**. *Constructor*. Crea un nodo vacío. **-Pro()**. *Observador y transformador*. Manejo de la profundidad del nodo en el árbol. **-Ne()**. *Observador y transformador*. Manejo del número actual de entradas en el nodo. **-PIzq().** *Observador y transformador.* Manejo del apuntador al hermano izquierdo. -**PDer()**. *Observador y transformador*. Manejo del apuntador al hermano derecho. -**Clave()**. *Observador y transformador*. Manejo de las claves del nodo. -**Puntero()**. *Observador y transformador*. Manejo de los apuntadores del nodo.

Implementación de la clase NodoB.

27/6/98		
Arbol_B+(Entero+: orden)		
{pre: orden ≠ 0 }		{pos: h = 0 ∧ nc = 0 ∧ m > 0 ∧ raiz = Nulo }
1	h, nc, m, raiz = 0, 0, orden, Nulo	**-h, nc, m, raiz.** Definidos en Arbol_B+.

Constructor vacío. O(1).

27/6/98		
~Arbol_B+()		
{pre: nc ≥ 0 ∧ m > 0 }		
1	Si (raiz ≠ Nulo) entonces limpiarArbol_B+(raiz) fsi	**-raiz, limpiarArbol_B+().** Definidos en Arbol_B+.

Destruye el Arbol_B+. O(n).

27/6/98		
limpiarArbol_B+(ApuntadorA NodoB: pna)		
{pre: nc ≥ 0}		{pos: h = 0 ∧ nc = 0 ∧ raiz = Nulo }
1	(pna → Ne() ≥ 0 ∧ pna → Pro() ≠ 0) [limpiarArbol_B+(pna → Puntero(pna → Ne())) pna → Ne(pna → Ne() - 1)]	**-limpiarArbol_B+().** Definido en Arbol_B+. **-Ne(), Pro(), Puntero().** Definidos en NodoB. **-r.** ApuntadorA NodoB. Variable auxiliar.
2	r = pna	
3	DevuelvaDir r	
4	h, nc, raiz = 0, 0, Nulo	

Devuelve todos los nodos del Arbol_B+. O(n).

27/6/98		
	insArbol_B+(Reg: r)	
{pre: r ≠ {RegNoDef} ∧ nc ≥ 0}		{pos: nc ≥ 0}

1	Si (raiz = Nulo) entonces raiz = NuevaDir NodoB[m*LON +4] / creación de la 1era. raíz Si (raiz = Nulo) entonces regrese fsi raiz → Pro(0) raiz → Ne(1) raiz → PIzq(Nulo) raiz → PDer(Nulo) raiz → Clave(0, r.Clave( )) raiz → Puntero(0, DireccionDe r) nc = nc + 1 sino pa, enc, clp = raiz, Falso, r.Clave() Si (pa → Pro() ≠ 0) entonces enc = recorrer(pa, cd, clp) fsi Si (enc) entonces regrese fsi i = 0 (r.Clave() < pa → Clave(i) ∧ i < pa → Ne()) [i = i + 1] Si (i < pa→ Ne()) entonces [pa → Clave(j, pa → Clave(j – 1)) pa → Puntero(j, pa → Puntero(j – 1))] j = pa → Ne(), i + 1, -1 fsi pa → Clave(i, r.Clave()) pa → Puntero(i, DireccionDe r) pa → Ne(pa → Ne() + 1) Si (pa→ Ne() ≥ 2m) entonces enc = dividirNodo(pa, ph, clp) / El nodo está lleno propagarFision(pa, ph, clp, cd) Si (pa = raiz) entonces / Crecimiento del árbol	**-pa, ph:** ApuntadorA NodoB. Apuntador al nodo actual y al nodo hermano, respectivamente. **-Clave(), Puntero(), PDer(), PIzq(), Pro(), Ne():** Definidos en la clase NodoB. **-nc, h, m, raiz, recorrer(), dividirNodo(), propagarFision()**: Definidos en Arbol_B+. **-clp:** TipoClave**.** Clave promocionada. **-DireccionDe.** Regresa la dirección de memoria de la variable. **-NuevaDir.** Regresa una nueva dirección de memoria. **-enc:** Lógico. Variable auxiliar para detectar si la clave del nuevo registro está en el árbol. **-i, j.** Entero+. Subíndices para recorrer los arreglos. **-MAXCLAVE:** TipoClave. Constante con un valor más grande que el de cualquier otra clave. **-LON:** Entero. Longitud en bytes de una entrada conformada por la clave y un apuntador.

Inserta un nuevo registro en el Arbol_B+. O(lg(n)).

27/6/98		
	vacíoArbol_B+(): Lógico	
{pre: nc ≥ 0}		{pos: nc ≥ 0}
1	Regresa (nc = 0)	**-nc.** Definido en Arbol_B+.

Verifica si el Arbol_B+ está vacío. O(1).

27/6/98		
	numClaves(): Entero+	
{pre: nc ≥ 0}		{pos: nc ≥ 0}
1	Regresa nc	**-nc.** Definido en Arbol_B+.

Devuelve el número de elementos actuales en el Arbol_B+. O(1).

27/6/98		
	altura(): Entero+	
{pre: nc ≥ 0}		{pos: nc ≥ 0}
1	regrese h	**-h.** Definido en Arbol_B+.

Devuelve la altura actual del Arbol_B+. O(1).

27/6/98		
	orden(): Entero+	
{pre: nc ≥ 0}		{pos: nc ≥ 0}
1	regrese m	**-m.** Definido en Arbol_B+.

Devuelve el orden del Arbol_B+. O(1).

27/6/98

recorrer(ApuntadorA NodoB: pa, PilaEnl[ElePila]: cd, TipoClave: clp): Lógico

{pre: nc ≥ 0} {pos: nc ≥ 0}

1	[i = 0 (clp > pa → Clave(i) ∧ i < pa → Ne() – 1) [i = i + 1] Si (clp = pa → Clave (i)) entonces regrese Verdadero fsi ElePila ep ep.Apu(pa) ep.Sub(i) cd.insElePila(ep) pa = pa → Puntero (i)] j = 0, h	**-Apu(), Sub().** Definidos en ElePila. **-Pro(), Clave(), Ne(), Puntero().** Definidos en NodoB. **-ep:** ElePila. Elemento nuevo para ingresar a cd. **-insElePila().** Definido en PilaEnl[ElePila]. **-i.** Entero+. Subíndice
2	regrese Falso	

Recorre el Arbol_B+. O(2m*h).

27/6/98

dividirNodo(ApuntadorA NodoB: pa, ph, TpoClave: clp): Lógico

{pre: nc ≥ 0} {pos: nc ≥ 0 }

1	nepa = pa → Ne()	**-Ne(), Clave(), Pro(), NodoB, PIzq(), PDer(), Puntero().** Definidos en la clase NodoB.
2	neph = Si (mod(nepa, 2) = 0) entonces 1 + nepa/2 sino nepa/2 fsi	**-LON:** Entero. Longitud en bytes de una entrada conformada por la clave y un apuntador.
3	clp = pa → Clave(nepa - 1)	**-NuevaDir.** Regresa una nueva dirección de memoria.
4	ph = NuevaDir NodoB[m*LON+4]	**-nepa, neph:** Entero+. Número de entradas que quedarán en el nodo desbordado y en su nodo hermano, respectivamente.
5	Si (ph ≠ Nulo) entonces ph → Pro(pa → Pro()) ph → PIzq(pa) ph → PDer(pa → PDer()) j = nepa [ph → Clave(i, pa → Clave(j)) ph → Puntero(i, pa → Puntero(j)) j = j + 1] i = 0, neph – 1 pa → Ne(nepa) ph → Ne(neph) pa → PDer(ph) regrese Verdadero fsi	**-i, j.** Entero+. Subíndices para recorrer los arreglos.
6	regrese Falso	

Divide un nodo. O(2m/2).

27/6/98

propagarFision(ApuntadorA NodoB: pa, ph, TipoClave: clp, PilaEnl[ElePila]: cd)

{pre: nc ≥ 0} {pos: nc ≥ 0 }

1	(pa ≠ raiz) [ep = cd.conPila() cd.eliElePila() pa = ep.Apu() p = ep.Sub() [pa → Clave(i + 1, pa → Clave(i)) pa → Puntero(i + 1, pa → Puntero(i))] i = pa → Ne(), p + 1, -1 pa → Clave(p + 1, clp) pa → Clave(p + 1, ph) pa → Ne(pa → Ne() + 1) Si (pa → Ne() > 2m) entonces dividirNodo(pa, ph, clp) fsi]	**-Ne(), Clave(), Puntero().** Definidos en la clase NodoB. **-m, raiz, dividirNodo().** Definidos en la clase Arbol_B+. **-ep:** ElePila. Elemento de la pila cd. **-conPila(), eliElePila().** Definidos en PilaEnl[ElePila]. **-i, p.** Entero+. Subíndice y punto de inserción, respectivamente. **-Apu(), Sub().** Definidos en ElePila.
2	regrese	

Propaga la división de nodo hacia arriba del Arbol_B+. O(2m*h).

27/6/98

conArbol_B+(TipoClave: cl): TipoResto

{pre: nc ≥ 0}

{pos: nc ≥ 0 }

```
1  Si (raiz ≠ Nulo ) entonces
     pa = raiz
     (pa → Pro() ≠ 0) [ i, ne = 0, pa → Ne()        / Descenso hasta la hoja
                        (i < ne ∧ cl > pa → Clave( i )) [ i = i + 1 ]
                        pa = pa → Puntero ( i )
                       ]
     i = 0
     ( i < pa → Ne() ∧ cl ≠ pa → Clave( i ))  [ i = i + 1  ]
     Si (cl = pa → Clave( i )) entonces
        regrese ContenidoDe pa → Puntero( i )
     fsi
   fsi
2  regrese {TipoRestoNoDef}
```

-raiz. Definido en la clase Arbol_B+.

-Ne(), Pro(), Clave(), Puntero(). Definidos en la clase NodoB.

-i, ne. Entero+. Subíndice y número de entradas del nodo actual, respectivamente.

-ContenidoDe. Regresa el contenido de la variable apuntada.

Consulta al registro que contiene la clave. O(2m*h).

27/6/98

rango(TipoClave: vi, vf): ListaEnl[Reg]

{pre: nc ≥ 0} {pos: nc ≥ 0}

```
2  Si (raiz ≠ Nulo ) entonces
     pa = raiz
     (pa → Pro() ≠ 0) [ i, ne = 0, pa → Ne()        / Descenso hasta la hoja
                        (i < ne ∧ vi > pa → Clave( i )) [ i = i + 1 ]
                        pa = pa → Puntero ( i )
                       ]
     i = 0
     ( i < pa → Ne() - 1 ∧ vi ≠ pa → Clave( i ))  [ i = i + 1  ]
     Si (vi = pa → Clave( i )) entonces
       lr.insLis(ContenidoDe  pa → Puntero( i ) )
       lr.cursorAlProximo()
       (vf > pa → Clave( i ))  [ (i < pa → Ne() – 1 ∧ vf > pa → Clave( i ))
                                   [ i = i + 1
                                     lr.insLis(ContenidoDe  pa → Puntero( i ) )
                                     lr.cursorAlProximo()
                                   ]
                                 pa, i = pa → PDer( ), 0    ]
     fsi
   fsi
1  ListaEnl[Reg]  lr
3  regrese  lr
```

-raiz. Definido en la clase Arbol_B+.
-Ne(), Pro(), Clave(), Puntero(). Definidos en la clase NodoB.
-i, ne. Entero+. Subíndice y número de entradas del nodo actual, respectivamente.
-ContenidoDe. Regresa el contenido de la variable apuntada.
-lr. ListaEnl[Reg]. Lista de registros que responden a la consulta por rango de claves.
-insLista(), cursorAlProximo(). Definidos en la clase ListaEnl[TipoEle].

Devuelve la lista de registros cuyas claves están entre vi y vf. O(2m*h*nc).

27/6/98

eliArbol_B+ (TipoClave: cl)

{pre: nc ≥ 0} {pos: nc ≥ 0}

```
1   pa = raiz
2   Si (¬ recorrer(pa, cd, cl) ) entonces    regrese
    fsi
3   i = 0                                                    / pa es una hoja
4   (cl > pa → Clave( i ) ∧ i < pa → Ne( ))  [ i = i + 1]
5   Reg  r
6   r = ContenidoDe pa → Puntero( i )
7   [pa → Clave(j, pa → Clave( j + 1))
     pa → Puntero(j, pa → Puntero( j + 1) ) ]  j = i, pa → Ne( ) - 1
8   pa → Ne(pa → Ne( ) - 1)
9   nc = nc - 1
10  Si (pa → Ne( ) < m + 1) entonces   / El nodo tiene menos entradas que
      compactarNodo(pa, cd)            / las permitidas
    fsi
    Si (pa = raiz ∧ pa → Puntero( 0 ) = Nulo ) entonces
11    raiz = Nulo                                / Vaciado del árbol
      devuelvaDir pa
      h = h - 1
    fsi
12  regrese
```

-pa. ApuntadorA NodoB. Variable auxiliar para indicar el nodo actual.
-cd. PilaEnl[ElePila]. Pila que contiene el camino de descenso.
-Ne(), Pro(), Clave(), Puntero(). Definidos en la clase NodoB.
-m, nc, raiz, recorrer(), compactarNodo(): Definidos en Arbol_B+.
-r. Reg. Variable auxiliar para guardar el registro eliminado.
-ContenidoDe. Regresa el contenido de la variable apuntada.
-devuelvaDir: Regresa la dirección a la memoria disponible.
-i, j: Entero+. Subíndices.

Elimina una entrada del árbol_B+. O(2m*h).

27/6/98

compactarNodo(ApuntadorA NodoB: pa, PilaEnl[ElePila]: cd)

{pre: nc ≥ 0} {pos: nc≥ 0}

```
1  ep = cd.conPila()
2  cd.eliElePila()
3  pp = ep.Apu()
4  pos = ep.Sub()
5  Si (pos = 0 ) entonces
     phd, c = pp → Puntero(pos + 1), 1
   sino  Si (pos = pp → Ne() – 1) entonces
            phi, c = pp → Puntero(pos – 1), 2
         sino
            phi, phd, c = pp → Puntero(pos – 1), pp → Puntero(pos + 1), 3
         fsi
   fsi
6  eli = redistribucion(pa, pp, phi, phd, pos, c)
7  Si (eli) entonces
     [ pp → Clave(j, pp → Clave(j + 1))
       pp → Puntero(j, pp → Puntero(j + 1))   ]  j = pos, pp → Ne() – 1
     pp → Ne(pp → Ne() – 1)
     Si (pp → Ne() < m + 1 ∧ pp ≠ raiz) entonces
        compactarNodo(pp, cd)
     fsi
   fsi
8  regrese
```

-ep: ElePila. Elemento de la pila cd.
-conPila(), eliElePila(). Definidos en PilaEnl[ElePila].
-j, pos. Entero+. Subíndice y punto de eliminación, respectivamente.
-Apu(), Sub(). Definidos en ElePila.
-compactarNodo(), redistribución(). Definidos en Arbol_B+.
-Puntero(), Clave(), Ne(). Definidos en NodoB.
-pp, phi, phd. ApuntadorA NodoB. Nodo padre, nodo hermano izquierdo y nodo hermano derecho, respectivamente.
-c. Entero+. Variable auxiliar para indicar el caso de redistribución.

Recorre recursivamente el Arbol_B+ mientras haya que compactar. O(2m*h).

27/6/98

redistribucion(ApuntadorA NodoB: pa, pp, phi, phd, Entero+: p, c)

{pre: nc ≥ 0} {pos: nc ≥ 0}

```
En caso que c =
 1: na, nhd = pa → Ne(), phd →Ne()      / Solo hermano derecho
    Si (na + nhd ≤ 2m) entonces                    / Eliminación de pa
       [phd → Clave(j + na –1, phd → Clave(j – 1))
        phd → Puntero(j + na – 1, phd → Puntero(j – 1)) ] j = nhd, 1, -1
       [phd → Clave(j, pa → Clave(j))
         phd → Puntero(j, phd → Puntero(j))  ] j = 0, na
       phd → Ne(na + nhd)
       devuelvaDir  pa
       regrese  Verdadero
    sino                                  / Equilibrio en número de claves,
       ned = (na + nhd)/2                 / sin eliminación de pa
       Si ((na + nhd) mod 2 = 0 ) entonces          / na + nhd par
          [pa → Clave(j, phd → Clave(j - na))
           pa → Puntero(j, phd → Puntero(j - na))  ] j = na, ne
          pa → Ne(ned)
          [phd → Clave(j, phd → Clave(j + na))
           phd → Puntero(j, phd → Puntero(j + na))   ] j = 0, na
          phd → Ne(ned)
       sino                                      / na + nhd impar
          ne = ned + 1
          [pa → Clave(j, phd → Clave(j - na))
           pa → Puntero(j, phd → Puntero(j - na))  ] j = na, ne
          pa → Ne(ne)
          [phd → Clave(j, phd → Clave(j + ne - na))
           phd → Puntero(j, phd → Puntero(j + ne - na))   ] j = 0, na
          phd → Ne(ned)
       fsi
```

-pp, phi, phd. ApuntadorA NodoB. Nodo padre, nodo hermano izquierdo y nodo hermano derecho, respectivamente.
-c. Entero+. Variable auxiliar para indicar el caso de redistribución.
-Puntero(), Clave(), Ne (). Definidos en NodoB.
-j, p. Entero+. Subíndice y punto de eliminación, respectivamente.
-posordenArbol_B+(). Definido en Arbol_B+.
-Info(), PIzq(), PDer(). Definidos en NodoB.

Redistribuye las claves del nodo actual entre sus hermanos. O(6m).

27/6/98 despliegue ()		
{pre: nc ≥ 0}		{pos: nc ≥ 0}

1	Si (nc = 0) entonces
	  Despliegue “El árbol_B+ está vacío”
	sino
	  Despliegue “Elementos del árbol_B+”
	  pa = raiz → PIzq( )
	  (pa ≠ Nulo) [[pa → Puntero(i) → despliegue()] i = 0, pa → Ne() – 1
	  pa = pa → PDer()]
	fsi
		-nc, raiz. Definidos en la clase Arbol_B+.
		-PIzq(), Ne(), PDer(), Puntero (). Definidos en la clase NodoB.
		-pa. ApuntadorA NodoB. Dirección del nodo actual.
		-i. Entero+. Subíndice.

2	regrese	**-despliegue().** Definida en la clase Reg.

Despliega todos los registros indexados con el árbol. O(nc)