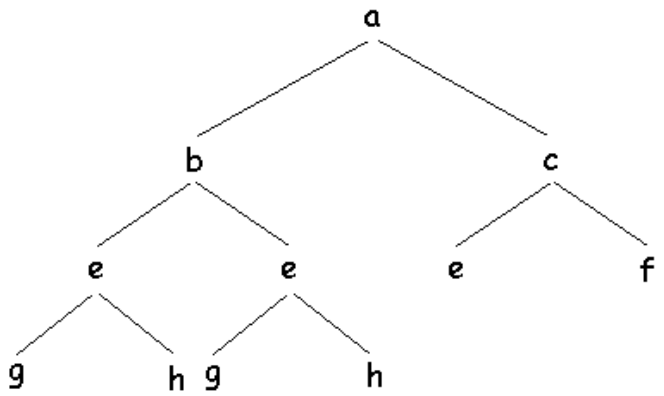
ÁRBOLES PARCIALMENTE ORDENADOS

INTRODUCCIÓN

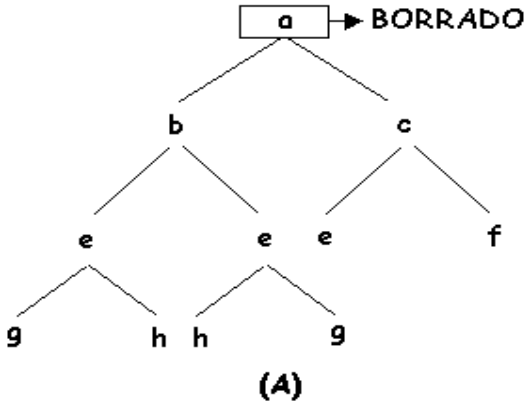
- Un árbol A se dice parcialmente ordenado (APO) si cumple la condición de que la etiqueta de cada nodo es menor (de igual forma mayor) o igual que las etiquetas de los hijos (se supone que el ***tipo_elemento*** base admite un orden) manteniéndose además tan balanceado como sea posible, en el caso óptimo equilibrado.



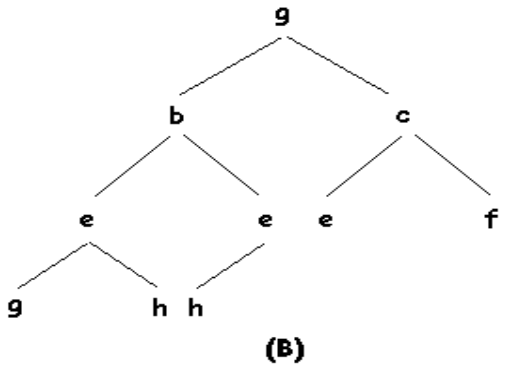
Ejemplo de árbol parcialmente ordenado.

BORRADO EN LOS APO

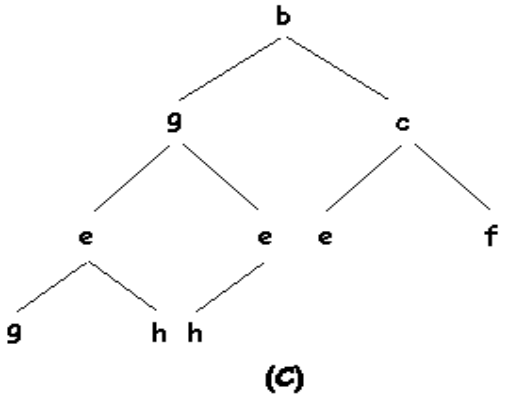
- No se puede quitar el nodo sin más, ya que se desconectaría la estructura. Si se quiere mantener la propiedad de orden parcial y el mayor balanceo posible con las hojas en el nivel más bajo alojadas de izquierda a derecha, lo que podría hacerse es poner provisionalmente la hoja más a la derecha del nivel más bajo como raíz provisional. Empujaremos entonces esta raíz hacia abajo intercambiándola con el hijo de etiqueta menor hasta que no podamos hacerlo más (porque sea ya una hoja o porque la etiqueta sea ya menor que la de cualquiera de sus hijos).



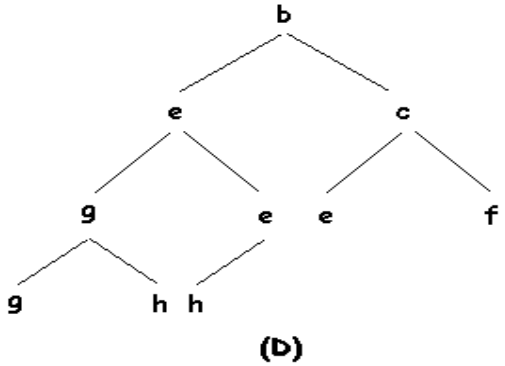
(A)



(B)



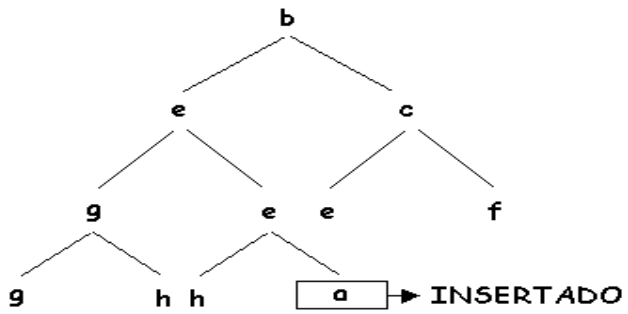
(C)



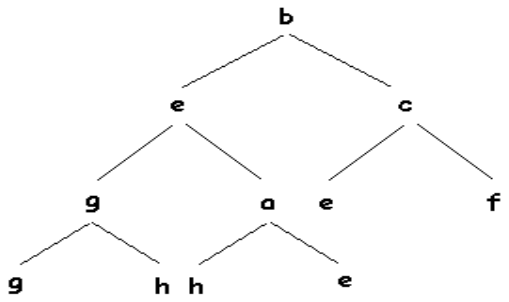
(D)

INSERCIÓN EN LOS APO

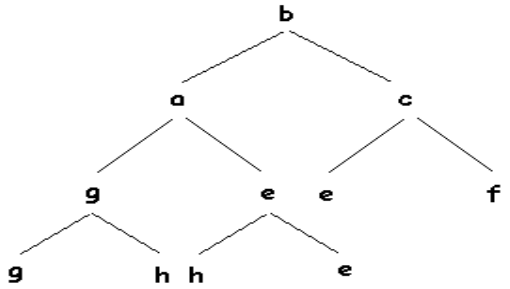
- El nuevo elemento que se inserta, lo podríamos situar provisionalmente en el nivel más bajo tan a la izquierda como sea posible (se comienza en un nuevo nivel si el último nivel está completo). A continuación se intercambia con su padre repitiéndose este proceso hasta que se cumpla la condición de orden parcial (bien porque ya esté en la raíz o porque tenga ya una etiqueta mayor que la de su padre).


b
e
c
g
e
e
f
g
h
h
a
INSERTADO

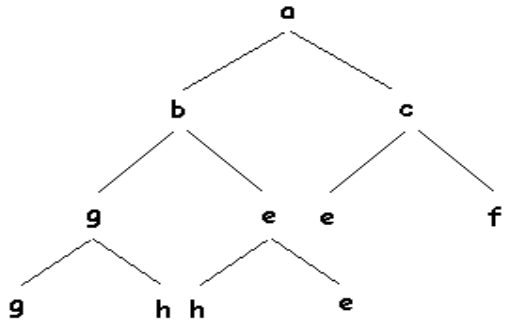
(A)


b
e
c
g
a
e
f
g
h
h
e

(B)


b
a
c
g
e
e
f
g
h
h
e

(C)


a
b
c
g
e
e
f
g
h
h
e

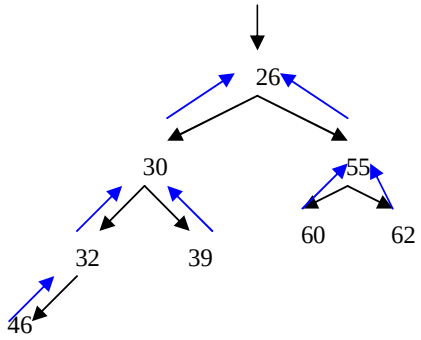
(D)

TDA APO

- Una instancia del tipo de dato abstracto APO sobre un dominio Tbase es un árbol binario con etiquetas en Tbase y un orden parcial que consiste en que la etiqueta de un nodo es menor o igual que la de sus descendientes. Para poder gestionarlo, debe existir la operación menor (<) para el tipo Tbase.

- El espacio requerido para el almacenamiento es O(n), donde n es el número de elementos del árbol.

- En un APO tenemos un vector de tipo Tbase llamada vec donde alamacenaremos los elementos del APO. Tenemos nelementos que nos da el número de elementos del APO. Por último tenemos maxelementos que nos indica la cantidad de posiciones reservadas en vec, como es natural debe ser mayor o igual a nelementos.



Nº elementos

8

max elementos

8

Vector

26	30	55	32	39	60	62	46

CLASE ABSTRACTA

```
Class APO
       {

       Public:

           Apo();
           /* Constructor por defecto
              Efecto: reserva los recursos e inicializa el árbol a  vacío */
           Apo (int tam);
           /* Constructor con tamaño
              Efecto: reserva los recursos para el arbol de tamaño tam e inicializa el árbol a vacío.
              PARÁMETROS: tam -> número de elementos que se espera pueda llegar a tener el árbol.

           Apo(const Apo<Tbase>& a);
           /* Constructor de copia
              Efecto: construye el árbol duplicando el contenido de a en el árbol receptor.
              PARÁMETROS: a -> Apo a copiar.

           Apo<Tbase>& operator = (const Apo<Tbase> &a);
            /* Asignación
              Efecto: asigna el valor del árbol duplicando el contenido de v en el árbol receptor.
              Devuelve la referencia al árbol receptor.
              PARÁMETROS: v -> Apo a copiar. */

           const Tbase& minimo () const;
           /* Mínimo elemento almacenado.
              Efecto: Devuelve una referencia al elemento más pequeño de los almacenados en el árbol
              receptor.
              PRECONDICIÓN: el árbol no está vacío.
           */
           void borrar_minimo ();
```

```
            /* Elimina el mínimo
               Efecto:elimina del árbol receptor el elemento más pequeño.
               PRECONDICIÓN: el árbol no está vacío.  */
            void insertar (const Tbase& el);
            /* Insertar un elemento
              Efecto: inserta el elemento el en el árbol receptor.
              PARÁMETROS: param -> nuevo elemento a insertar.
           */
            void clear ();
            /* Borra todos los elementos
              Efecto: borra todos los elementos del árbol receptor, cuando termina, el árbol está
            vacío.
            */
            int size () const;
            /* Número de elementos
             Efecto: devuelve el número de elementos del árbol   receptor.
            */
            bool empty () const;
            /* vacío
              Efecto: devuelve true si el número de elementos del árbol  receptor es cero, false en
             otro caso.
            */
            ~Apo ();
            /*Destuctor
            Efecto: libera los recursos ocupados por el árbol receptor */

        Private:

            Tbase *vec;
            int nelementos;
            int maxelementos;

            void expandir (int nelem);

    }
```

IMPLEMENTACIÓN

```
template <class Tbase>
//constructor por defecto( de APO de tamaño 0)
Apo<Tbase>::Apo()
    {
    vec= new Tbase;
    nelementos= 0;
    maxelementos= 1;
    }
```

```
//constructor de APO de tamaño tam

template <class Tbase>
  Apo<Tbase>::Apo(int tam)
    {
    vec= new Tbase[tam];
    nelementos= 0;
    maxelementos= tam;
  }
```

```
//constructor de copia
template <class Tbase>
  Apo<Tbase>::Apo (const Apo<Tbase>& a)
    {
    int i,aux;

    aux=a.nelementos;
    if (aux==0)
        aux=1;
    vec= new Tbase[aux];
```

```
    nelementos= a.nelementos;
    maxelementos= aux;
    for (i=0;i<nelementos;i++)
          vec[i]= a.vec[i];
   }
```

```
//expandir el arbol(aumenta la capacidaddel arbol)
template <class Tbase>
  void Apo<Tbase>::expandir (int nelem)
    {
    Tbase *aux;
    int i;

    if (nelem > maxelementos)
       {
      aux= new Tbase[nelem];
      for (i=0;i<nelementos;i++)
            aux[i]=vec[i];
      delete[] vec;
      vec= aux;
      maxelementos=nelem;
      }
  }
```

```
//operador de asignación
template <class Tbase>
Apo<Tbase>& Apo<Tbase>::operator = (const Apo<Tbase> &a)
    {
    int i;

    if (this!=&a)
       {
       delete[] vec;
```

```
       vec= new Tbase[a.nelementos];
       nelementos= a.nelementos;
       maxelementos= a.nelementos;
       for (i=0;i<nelementos;i++)
            vec[i]= a.vec[i];
       }
    return *this;
    }
```

```
//devolver el minimo
template <class Tbase>
inline const Tbase& Apo<Tbase>::minimo() const
    {
    assert(nelementos>0);
    return vec[0];
    }
```

```
//borrar minimo
template <class Tbase>
  void Apo<Tbase>::borrar_minimo()
    {
    int pos,pos_min, ultimo;
    bool acabar;
    Tbase aux;
    assert(nelementos>0);
    vec[0]=vec[nelementos-1];
    nelementos--;
    if (nelementos>1)
       {
       ultimo= nelementos-1;
       pos=0;
       acabar= false;
       while (pos<=(ultimo-1) / 2 && !acabar)
```

```
            {
            if (2*pos+1 == ultimo)
             pos_min=2*pos+1;
            else if (vec[2*pos+1] < vec[2*pos+2])
                                pos_min= 2*pos+1;
                    else pos_min= 2*pos+2;
            if (vec[pos_min] < vec[pos])
                {
                aux= vec[pos];
                vec[pos]=vec[pos_min];
                          vec[pos_min]= aux;
                pos=pos_min;
                }
            else acabar= true;
            }
        }
  }
```

```
//insertar un elemento
template <class Tbase>
  void Apo<Tbase>::insertar (const Tbase& el)
    {
    int pos;
    Tbase aux;
     if (nelementos==Maxelementos)
        expandir(2*Maxelementos);
     nelementos++;
     pos=nelementos-1;
     vec[pos]=el;
     while ((pos>0) && (vec[pos]<vec[(pos-1)/2]))
         {
         aux= vec[pos];
         vec[pos]=vec[(pos-1)/2];
         vec[(pos-1)/2]= aux;
```

```
          pos= (pos-1)/2;
          }
    }
```

```
//vacia el árbol
template <class Tbase>
inline void Apo<Tbase>::clear()
    {
    nelementos=0;
  }
```

```
//devuelve el tamaño del arbol
template <class Tbase>
inline int Apo<Tbase>::size() const {
    return nelementos;
    }
```

```
//devuelve true si el arbol es un arbol vacio y false en otro caso
template <class Tbase>
inline bool Apo<Tbase>::empty() const
    {
    return nelementos==0;
    }
```

```
//destructor
template <class Tbase>
inline Apo<Tbase>::~Apo()
    {
    delete[] vec;
  }
```