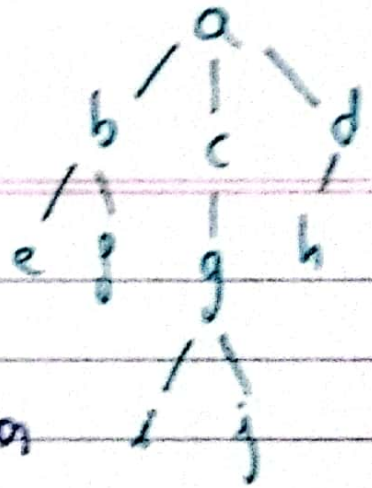
Árboles

Pre: Raíz, Pre(Tiq), Pre(Tdch)

In: In(Tiq), Raíz, In(Tdch)

Post: Post(Tiq), Post(Tdch), Raíz.



abefcgijdh
cbfaigjchd
efbijgchda
abcdefghij

- Grado de un nodo: Número de subárbol (hijos) que tiene el nodo.
- Grado de un árbol: Máximo de los grados de sus nodos.
- Profundidad de un árbol: Máximo de los niveles de los nodos del árbol.

- Altura de un nodo: Longitud del camino más largo entre ese nodo y una hoja.
  - Número de nodos que forman el camino menos 1. (nº de ejes del camino).
- Un árbol vacío es un árbol binario.
- Árbol binario homogéneo: Aquel cuyos nodos tienen grado 0 o 2
- Árbol binario completo: Aquel que tiene todos los niveles llenos excepto, quizá, el último en cuyo caso las hojas deben quedar a la derecha. (El camino más largo de la raíz a las hojas no atraviesa más de $\log_2 n$ nodos. (Árbol binario completo de altura k, el número máximo de nodos es $2^{k+1} - 1$).

En un árbol binario el número máximo de nodos que puede haber en el nivel i es 2^i.
En general un árbol no puede recuperarse con ayuda de su recorrido.
Hijo izquierdo en $2k+1$, hijo derecho en $2k+2$, padre $(k-1)/2$ (nodos se almacenan en

Otros tipos de árboles

APO ⇒ Árbol binario que cumple la condición de que la etiqueta de cada nodo es menor o igual que la etiqueta de sus hijos, manteniéndose lo equilibrado (balanceado) como sea posible (hojas empujadas a la izquierda). No podemos tener ceros en el nivel n sin tener el $n-1$ completo. (Árbol Parcialmente Ordenado). (Se guarda en un heap, un vector cada posición)

ABB ⇒ Árbol binario con la propiedad de que para cualquier nodo, su subárbol izquierdo tiene los elementos menores que la etiqueta de dicho nodo y el subárbol derecho los tiene mayores. El listado ordenado de los elementos es inorden. Búsqueda binaria $O(\log_2 n)$.
(Árbol Binario de Búsqueda)

AVL ⇒ Un árbol es un AVL (o que está equilibrado en el sentido de Addelson-Velski-Landis) si para cada uno de sus nodos se cumple que las alturas de sus dos subárboles difieren como máximo en 1 (altura_izda) - (altura_dcha). AVL (ABB + Equilibrado) ABB condición equilibrio Equilibrio condición geométrica.

Iteradores

Caso fácil

Operadores de desplazamiento:
(Se para referencia):

iterator & operator ++()
iterator & operator --()

Operador * devuelve referencia:

tipito & operator *()

Debemos poner al final de la clase iterator, que Clase es friend:

friend class Clase;

En los comparadores poner iterador por referencia:

bool operator !=/== (const iterador &i);

```
class Clase{
  private:
    Tipo<tipito> datos; // Parte privada clase
  public:
    // Funcs.
    class iterator{
      private:
        Tipo<tipito>::iterator it;
      public:
        // Funciones iterador
        // para desplazarnos:
        iterator & operator ++(){ return it++; };
        iterator & operator --(){ return it--; };
        // Devuelve el elemento
        tipito & operator *(){ return *it; };
        // Comparadores
        bool operator !=(const iterador &i){ return i.it != it; };
        bool operator ==(const iterador &i){ return i.it == it; };
        friend class Clase;
    };
    iterator begin(){
      iterator i;
      i = it.begin();
      return i;
    }
    iterator end(){
      iterator i;
      i = it.end();
      return i;
    }
```

⇒ Begin y end van en la parte pública de la clase.

Caso Medio

Tenemos una base de la STL pero iteramos con condiciones extra. Es igual que en lo solo que hay que tener mucho cuidado con el ++, el begin() y el end(). Generalmente vamos a tener que usar bucles, para saltar los elementos que no interesen, y en parte que tengamos que añadir en la declaración privada otro iterador, final, para apoyarnos para tenerle en cuenta en el ++.

Caso Difícil

Clase cualquiera sin STL ni nada de nada para que iteremos. En caso el caso fácil sobre que ya no podemos apoyarnos en la STL. Probablemente, solo habría que hacer el begin, end y ++, *.

Tablas Hash

Rehashing doble ⇒ Aplicamos la función Hash que mejor venga. Si hay colisión, rehasing:

$h_i(K) = (h_{i-1}(K) + h_0(K)) \% M$

$h_0(K) = 1 + (K \% (M-2))$

$h_1(K) = h(K)$

Cada vez que hay una colisión se llama a $h_i(K)$ donde i es el número de colisiones para esa posición.