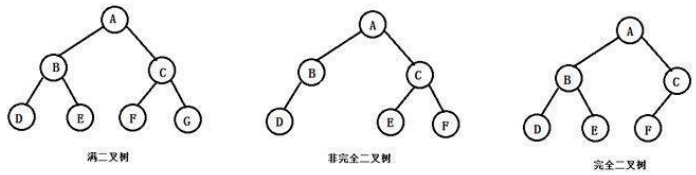
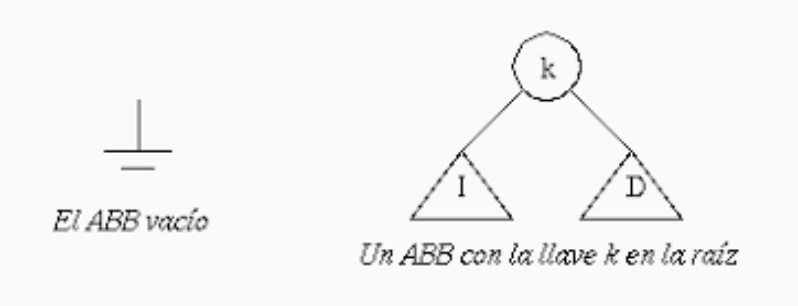
Método del Árbol

Una estructura de datos de árbol se puede definir de forma recursiva como una colección de nodos, donde cada nodo es una estructura de datos con un valor, junto con una lista de referencias a los nodos, con la condición de que ninguna referencia esté duplicada ni que ningún nodo apunte a la raíz.



Un árbol que no tiene ningún nodo se llama árbol vacío o nulo



¿Para qué sirve el método del árbol?

Intuitivamente, podemos visualizar árboles como una forma de organizar información de forma jerárquica, con un único punto de entrada y una serie de caminos que van abriéndose en cada punto hacia sus sucesores

Los árboles binarios se emplean a menudo para la representación de expresiones aritméticas, dado que una operación con dos operandos la podemos representar como un árbol cuya raíz sea el operador, y sus subárboles sean los operandos

Anulables

Se trata de un tipo especial de dato que permite que los tipos base por valor se puedan comportar como valores por referencia nulos cuando sea necesario.

De este modo este nuevo tipo anulable permite representar valores concretos o la ausencia de valor. Por ejemplo, una variable de tipo 'int' siempre contiene un número entero. Aunque no le asignemos explícitamente nada, al ser un tipo base por valor siempre toma un valor por defecto, en este caso concreto un 0. Pero claro, un 0 no es lo mismo que decir 'esta variable no contiene valor alguno'.

En el caso de las clases la ausencia de "valor" se representa asignándole un nulo a la variable, pero en los tipos por valor esto no es posible.

Cálculo de First

El conjunto PRIMERO(a), siendo a una forma sentencia, es el conjunto de Terminales que pueden aparecer los primeros en cadenas derivadas de a.

Para calcular PRIMERO(a) basta con seguir las siguientes reglas para cada regla de producción de la gramática:

1. Si a º Î, donde Î es la cadena vacía:

   Añadir { Î } a PRIMERO(a)

2. Si a º a, donde a es un Terminal:

   Añadir { a } a PRIMERO(a)

3. Si a º A, donde A es un No Terminal:

   Añadir PRIMERO(A) a PRIMERO(a).

4. Si a º AB, donde A y B son No Terminales y PRIMERO(A) incluye { Î }:

   Añadir PRIMERO(A) a PRIMERO(a), pero sin incluir de momento { Î }.

   Además, y puesto que A puede ser vacío, añadir PRIMERO(B) a PRIMERO(a).

   Esta regla puede extenderse a todos los No Terminales que aparezcan detrás de A. Por ejemplo, si a º ABCD y PRIMERO (A), PRIMERO(B) y PRIMERO(C) incluyen { Î }, entonces habrá que añadir a PRIMERO(a) la unión de

PRIMERO(A), PRIMERO(B), PRIMERO(C) y PRIMERO(D), sin incluir { Î }.

Por último se decide si incluir { Î } en PRIMERO(a): sólo debe incluirse si aparece en los conjuntos PRIMERO de todos los No Terminales. Por ejemplo, en el caso anterior, si PRIMERO(D) también incluyese { Î }, entonces PRIMERO(a) incluiría { Î }.

Cálculo de Next

Si A es un símbolo No Terminal de una gramática, SIGUIENTE(A) es el conjunto de Terminales (incluyendo el símbolo de fin de cadena, $) que pueden aparecer justo después de A en alguna forma sentencial derivada del símbolo inicial.

El cálculo de conjuntos SIGUIENTE se realiza a partir de estas reglas:

1. Los conjuntos SIGUIENTE de todos los No Terminales son inicialmente vacíos, excepto el del No Terminal inicial de la gramática, en el que se incluye el símbolo {$}.

   A partir de este punto, para calcular cada uno de los conjuntos deben aplicarse las dos siguientes reglas (no son excluyentes) por cada ocurrencia del No Terminal en la parte derecha de alguna regla de producción:

2. Si A ‡ aBb:

   Añadir a SIGUIENTE(B) los elementos de PRIMERO(b), con la excepción de {Î}: este símbolo nunca se incluirá en los conjuntos SIGUIENTE.

3. Si A ‡ aB, o bien A ‡ aBb donde PRIMERO(b) contiene {Î}:

   Añadir a SIGUIENTE(B) los elementos de SIGUIENTE(A)