iRAT #003 UT4

Lecturas Básicas de la Cátedra

Recursividad

$Recursión$: cuando un método se llama a sí mismo.

Decimos que un objeto es recursivo cuando está formado en parte por sí mismo o cuando está definido en función de sí mismo.

Se debe evitar su uso si existe una solución no recursiva aceptable.

Permite definir un conjunto infinito de cálculos mediante un programa recursivo sin repeticiones explícitas.

Los algoritmos recursivos son idóneos cuando el problema a resolver, la función para resolverlo o la estructura de datos a utilizar están definido de forma recursiva.

Si un módulo P contiene una referencia explícita a sí mismo, entonces es directamente recursivo.

Si un módulo P contiene una referencia explícita a un módulo Q, que a su vez tiene una referencia explícita al primer módulo P, se dice que P es indirectamente recursivo. Esto es generalmente un error que debe ser evitado.

Problema de terminación

Un módulo P de tamaño n consiste en una secuencia s de sentencias.

$$P(n) = P[s,\ si\ n > 0 \rightarrow P(n-1)]$$

Cada cadena de llamadas recursivas potencial debe eventualmente alcanzar un caso base. El manejo del caso base no debe usar recursividad.

Forma usual

```
Función A (N: tamaño del problema)
   COM
      <bloque>
      Si <condición> entonces
```

```
        <bloque>
        A(N-1)
        <bloque>
    Sino
        <bloque>
    Fin Si
    <bloque>
FIN
```

Factorial

$$f(n) = \begin{cases} 1 \\ n * f(n-1) \end{cases}$$

```
public static int factorial(int n) {
    if (n == 0) {
        return 1  // caso base
    } else {
        return n * factorial(n -1) // caso recursivo
    }
```

Estructuras de Datos en Java, Mark Allan Weiss

- ✅ ~~18.1.1 Definiciones~~
- ✅ ~~18.1.2 Implementación~~
- ✅ ~~18.1.3 Sistemas de archivos~~
- ✅ ~~18.2 Árboles Binarios~~
- ✅ ~~18.4 Recorrido del Árbol~~

18.1 Árboles Generales

Un árbol puede ser definido de dos maneras: recursivamente o no recursivamente.

18.1.1 Definiciones

Árboles no recursivos

Un árbol está compuesto de nodos y de un conjunto de aristas dirigidas que conectan parejas de nodos.

Un árbol con raíz tiene las siguientes propiedades:

- Uno de los nodos se distingue por ser raíz.
- Para todos los nodos c, excepto la raíz, está conectado por una arista a un único otro nodo p. Decimos que p es padre de, c hijo de p.
- Existe un único camino desde la raíz hasta cada nodo.
- El número de aristas es la longitud del camino

Una arista dirigida conecta al padre con el hijo. La raíz, por tanto, no tiene padre.

Un nodo sin hijo se denomina **hoja** o nodo externo. Un árbol de n nodos tiene$n - 1$ aristas.

La **profundidad** del nodo es la longitud del camino desde la raíz hasta dicho nodo. La **profundidad** de la raíz siempre es 0.

La **altura** de un nodo es la longitud del camino de dicho nodo hasta la hoja más profunda. La **altura** de un árbol es la altura de su raíz.

El **tamaño** de un nodo es el número de descendientes más el propio nodo.

Los nodos del mismo padre son **hermanos**.

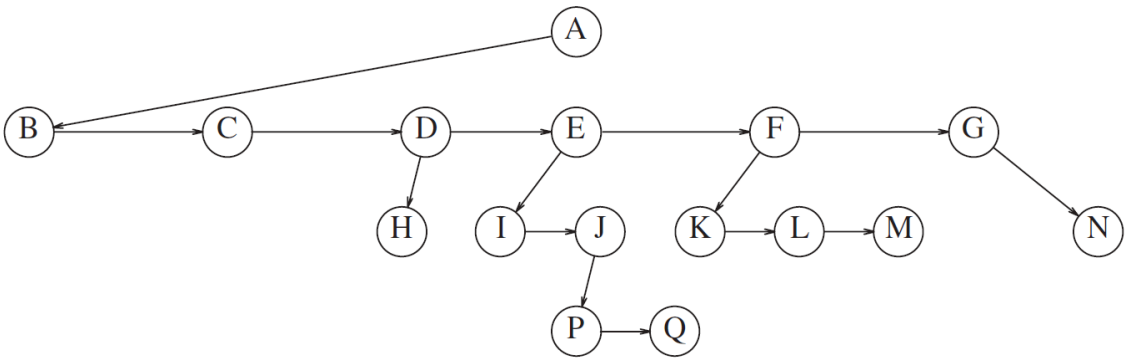
Si existe un camino desde u a v entonces u es **ancestro** de v y v es **descendiente** de u.

Árboles recursivos

Un árbol recursivo o bien está vacío o está compuesto de una raíz y cero o más sub-árboles no vacíos $\{T_1, T_2, T_3, \ldots, T_n\}$, cuyas raíces están conectadas a la raíz del árbol con una arista. Algunos sub-árboles pueden estar vacíos especialmente en el caso de árboles binarios.

18.1.2 Implementación Primer Hijo/Siguiente Hermano

Cada nodo mantiene 2 enlaces, uno a su primer hijo y a su siguiente hermano. Es decir, mantenemos a todos los hijos de un nodo dado en una Lista Enlazada, donde el hijo más a la izquierda a es la cabeza de la lista, y cada elemento en la lista tiene un enlace a su siguiente hermano, y también a su primer hijo.



$FirstChildNextSiblingParadygm$

18.1.3 Sistemas de Archivos

La forma más fácil de recorrer un sistema de archivos es la recursión.

Un directorio es simplemente un archivo con todos sus hijos.

Class File in Java.

18.2 Árboles Binarios

Árbol en el que ningún nodo puede tener más de 2 hijos. Puesto que hay solo dos hijos, podemos llamarlos hijo izquierdo e hijo derecho. Recursivamente hablando, un árbol binario puede estar vacío o estar compuestos por un árbol binario izquierdo y un árbol binario derecho, que pueden estar vacíos.

1. Árboles de expresión.
2. Árbol de Codificación de $Huffman$: cada símbolo del alfabeto se almacena en una hoja, el código de la letra se obtiene con la profundidad de dicha hoja.
3. Árboles de Búsqueda Binaria: permiten insertar elementos y acceder a ellos en tiempo logarítmico.

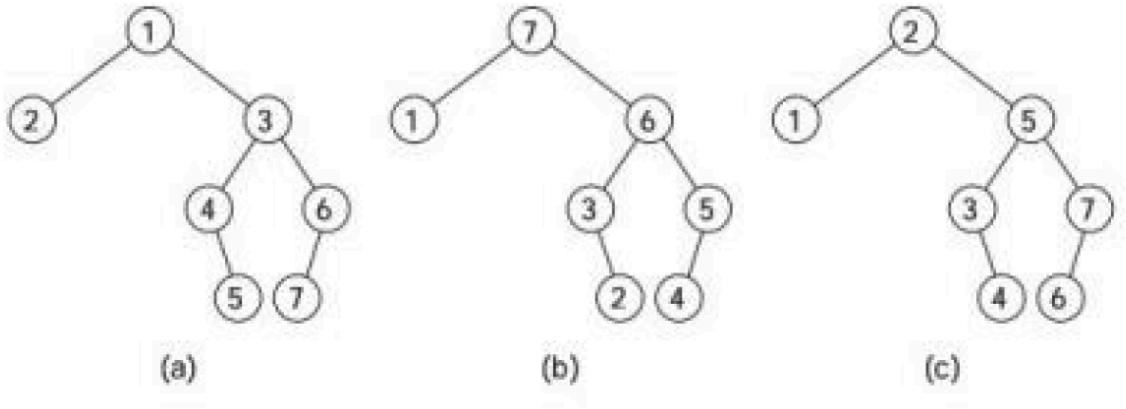
Class Binary Node.

Class Binary Tree Skeleton

```
1  public class BinarySearchTree<AnyType extends Comparable<? super AnyType>>
2  {
3      private static class BinaryNode<AnyType>
4        { /* Figure 4.16 */ }
5
6      private BinaryNode<AnyType> root;
7
8      public BinarySearchTree( )
9        { root = null; }
10
11     public void makeEmpty( )
12       { root = null; }
13     public boolean isEmpty( )
14       { return root == null; }
15
16     public boolean contains( AnyType x )
17       { return contains( x, root ); }
18     public AnyType findMin( )
19       { if( isEmpty( ) ) throw new UnderflowException( );
20         return findMin( root ).element;
21       }
22     public AnyType findMax( )
23       { if( isEmpty( ) ) throw new UnderflowException( );
24         return findMax( root ).element;
25       }
26     public void insert( AnyType x )
27       { root = insert( x, root ); }
28     public void remove( AnyType x )
29       { root = remove( x, root ); }
30     public void printTree( )
31       { /* Figure 4.56 */ }
32
33     private boolean contains( AnyType x, BinaryNode<AnyType> t )
34       { /* Figure 4.18 */ }
35     private BinaryNode<AnyType> findMin( BinaryNode<AnyType> t )
36       { /* Figure 4.20 */ }
37     private BinaryNode<AnyType> findMax( BinaryNode<AnyType> t )
38       { /* Figure 4.20 */ }
39
40     private BinaryNode<AnyType> insert( AnyType x, BinaryNode<AnyType> t )
41       { /* Figure 4.22 */ }
42     private BinaryNode<AnyType> remove( AnyType x, BinaryNode<AnyType> t )
43       { /* Figure 4.25 */ }
44     private void printTree( BinaryNode<AnyType> t )
45       { /* Figure 4.56 */ }
46 }
```

18. 4 Recorrido del Árbol

- Recorrido en $Preorden$: se procesa el dato del padre, luego del hijo izquierdo, finalmente del hijo derecho.
- Recorrido en $Inorden$: se procesa el dato el hijo izquierdo, luego del nodo padre, finalmente del hijo derecho.
- Recorrido en $Postorden$: se procesa el dato del hijo del izquierdo, luego del hijo derecho, finalmente del nodo padre.



a. Pre-orden, b. Post-orden, c. In-orden

1. Si **K = SL + 1**: El $K-ésimo$ menor elemento es la raíz del nodo actual.
2. Si **K ≤ SL**: El $K-ésimo$ menor elemento está en el subárbol izquierdo. Llamamos recursivamente al método.
3. Si **K > SL + 1**: El $K-ésimo$ menor elemento está en el subárbol derecho. Restamos SL + 1 de K. Llamamos recursivamente al método.

Desequilibrio en árboles AVL

Cuando realizamos una inserción en un árbol AVL es posible que un nodo X pierda el equilibrio, es decir, que la diferencia de altura entre su subárbol izquierdo y derecho sea mayor o igual a 2. Este nodo X es el primer nodo desequilibrado desde la hoja hasta la raíz, debemos restaurar el equilibrio para mantener la condición de los árboles AVL.

Cuatro desequilibrios posibles:

- **Izquierda-Derecha**: inserción en el subárbol derecho del hijo izquierdo de X.
- **Derecha-Izquierda**: inserción en el subárbol izquierdo del hijo derecho de X.

- **Derecha-Derecha**: inserción en el subárbol derecho del hijo derecho de X.

Rotaciones simples y dobles

Casos Izquierda-Izquierda y Derecha-Derecha: rotación simple, es decir, el hijo desequilibrado se convierte en el nuevo padre.

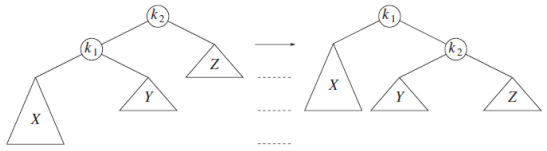
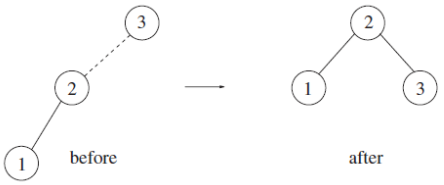


Figure 4.31 Single rotation to fix case 1



Casos Izquierda-Derecha y Derecha-Izquierda: rotación doble, es decir, se realiza una primera rotación sobre el hijo del nodo desequilibrado y luego una segunda rotación sobre el nodo desequilibrado.

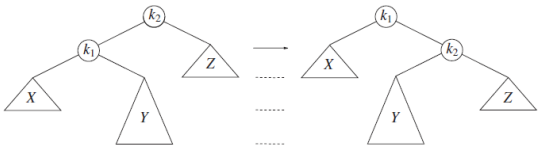


Figure 4.34 Single rotation fails to fix case 2