

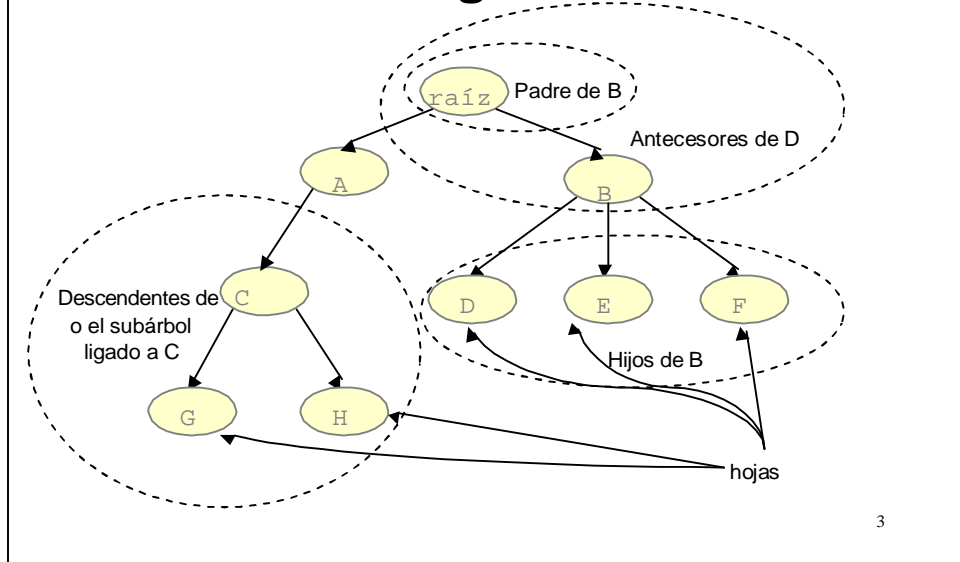
Clase 26

Introducción a los árboles

Árboles

- **Árbol** es el nombre que se le da a un grupo versátil de estructuras de datos.
- Se pueden utilizar para implementar un número de interfaces abstractas, incluida la interfaz `List`, pero las aplicaciones en las que resultan más útiles emplean estructuras de ramas de árboles para representar alguna propiedad de los elementos de los datos o para optimizar ciertos métodos.
- Por ejemplo, los *árboles de juegos Minimax* se suelen utilizar en programas de juegos para representar la forma en que las posiciones de la clasificación se multiplican a partir de una situación original.

Terminología del árbol



3

Árboles, nodos y raíces

- Un árbol consta de *nodos* conectados.
- Cada árbol (salvo un *árbol vacío degenerado*) cuenta con un nodo distinguido llamado *raíz*.
- No puede haber rutas circulares en las conexiones de un árbol, de tal forma que sólo puede existir una ruta única desde cada nodo hasta la raíz.

4

Nodos hijos y padres

- Todos los nodos conectados a un nodo concreto son *hijos* o bien el *padre* de dicho nodo.
- Si el nodo conectado se encuentra en la única ruta a la raíz, dicho nodo recibe el nombre de padre. Todos los nodos, salvo la raíz, tienen un único padre.
- El resto de nodos conectados a un nodo concreto son los hijos del nodo.

5

Antecesoros, descendientes y subárboles

- Los nodos que se encuentran en la ruta que va desde un nodo a la raíz reciben el nombre de antecesoros del nodo e incluyen a su padre, al padre de su padre, etc., hasta llegar a la raíz.
- El conjunto de nodos que incluyen a los hijos del nodo, a los hijos del hijo, etc., reciben el nombre de *descendientes* del nodo.
- Un nodo y sus descendientes forman un *subárbol* enraizado a dicho nodo.
- Un nodo sin hijos recibe el nombre de *hoja*.

6

Árbol binario

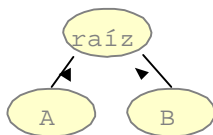
- Un tipo de árbol habitual y de gran utilidad es el llamado *árbol binario*, que permite que un nodo tenga, al menos, dos hijos.
- A continuación se incluye una definición formal del árbol binario que hace hincapié en el carácter recursivo del árbol:

Un árbol binario es un árbol vacío, o bien un nodo raíz con subárboles formados por árboles binarios a la derecha y a la izquierda.

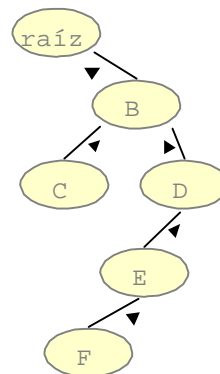
7

Ejemplos de árboles binarios

31.



2.



8

Recorrido del árbol

- Enumerar todos los elementos de un árbol es más complicado que enumerar todos los elementos de una lista enlazada y, además, hay varias formas de hacerlo.
- Decimos que la lista de los nodos de un árbol se puede recorrer si enumera cada nodo del árbol exactamente una vez.
- Las implementaciones de los árboles suelen constar de un iterador que enumera los elementos del árbol siguiendo una secuencia predecible.

9

Preorder, Inorder, Postorder

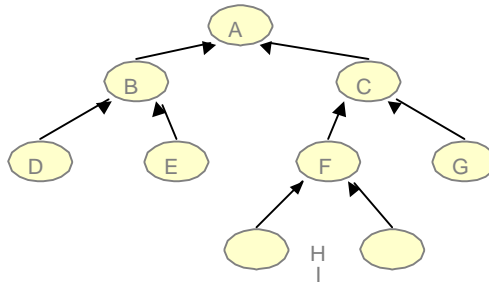
Los tres métodos más comunes para recorrer un árbol se pueden describir, de forma recursiva, del siguiente modo:

- *preorder*: raíz, subárbol izquierdo, subárbol derecho
- *inorder*: subárbol izquierdo, raíz, subárbol derecho
- *postorder*: subárbol izquierdo, subárbol derecho, raíz

10

Recorridos

- *Preorder*: A B D E C F H I G
- *Inorder*: D B E A H F I C G
- *Postorder*: D E B H I F G C A



11

Ejercicio de recorrido de árbol

- Descargue `TreeTraversal.jar` y `TreeTraversal.bat` del sitio web de clase en el *mismo* directorio. Puede encontrar los enlaces en la página del material de clase.
- Haga doble clic en `TreeTraversal.bat` para ejecutarlo.
- Utilice los botones inferiores para analizar la terminología del árbol.
- Utilice los botones superiores para analizar los tres recorridos típicos del árbol: *inorder*, *preorder* y *postorder*.

12

Árboles binarios de búsqueda

- Considere un árbol binario con nodos que contienen un entero llamado *value* que representa el valor del nodo. Haremos que este árbol asuma la propiedad de cada nodo siguiente, el valor de los nodos del subárbol de la izquierda (si existe) siempre es menor o igual que el valor del nodo que es menor o igual que los valores de los nodos del subárbol de la derecha (si existe).
- El árbol se ordena y el recorrido *inorder* generará una lista ordenada del mismo.
- Dichos árboles reciben el nombre de *árboles binarios* y los estudiaremos en profundidad en la siguiente clase.

Árboles analizadores

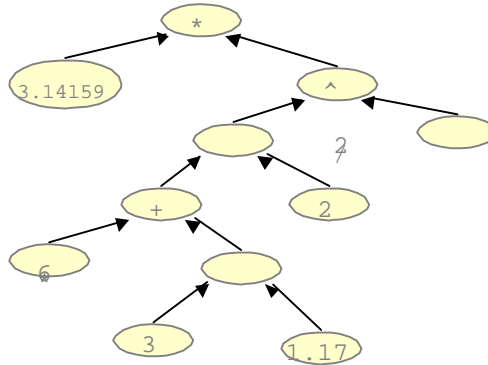
- Considere una expresión aritmética que tal vez desee calcular con una calculadora:

$$3.14159 * ((6 + 3 * 1.17) / 2)^2$$

- Esta expresión se puede ver como un árbol en el que los nodos que no son hojas contienen operadores y sus hijos contienen los operandos, que pueden ser subexpresiones. El árbol resultante recibe el nombre de árbol *analizador* o de *expresión*.

Diagrama de árbol analizador

$3.14159 * ((6 + 3 * 1.17) / 2)^2$



15

Árboles analizadores y *postorder*

Un recorrido *postorder* del árbol analizador generará una versión postfijo de la expresión original:

$3.14159 * ((6 + 3 * 1.17) / 2)^2$



3.14159 6 3 1.17 * + 2 / 2 ^ *

16

Ejercicio TreeEditor, 1

- Descargue `TreeEditor.jar` y `TreeEditor.bat` desde el sitio web de clase en el *mismo* directorio. Puede encontrar los enlaces en la página del material de clase.
- Haga doble clic en `TreeEditor.bat` para ejecutarlo.
- Cree un árbol binario de búsqueda agregando un nodo raíz de valor 7 y, a continuación, hijos con la siguiente secuencia de nodos: 1,3,4,5,8,9,10,15. Compruebe que el árbol cumple la propiedad del árbol binario de búsqueda. Compruebe los valores de los nodos haciendo clic en recorrido *inorder*. Si ha construido un árbol binario de búsqueda correcto, los nodos se mostrarán en orden. Ahora seleccione el nodo raíz, borre la pantalla y vuelva a crear el árbol con los mismos nodos, pero esta vez convierta al nodo 3 en nodo raíz.

17

Ejercicio TreeEditor, 2

- Seleccione el nodo raíz y borre la pantalla. Ahora cree un árbol analizador para la expresión aritmética $(1+2.718)*2 / 2*3.14$ utilizando +, -, * y /.
- Haga clic en los botones *inorder* y *postorder* de recorrido. Si está seguro de haber creado un árbol analizador correcto, haga clic en el botón de cálculo para calcular la expresión.
- El editor mostrará la respuesta "please check your input for each node" (compruebe los datos de entrada para cada nodo) si el árbol no es válido. Por el contrario, si ha creado un árbol correcto, se calculará el valor de la expresión.

18

Implementación de recorrido *inorder*, 1

```
public class BinaryTree
{
    private Object value;
    private BinaryTree left = null;
    private BinaryTree right = null;

    public BinaryTree( Object o, BinaryTree l,
                     BinaryTree r )
    {
        value = o; left = l; right = r;
    }
}
```

19

Implementación de recorrido *inorder*, 2

```
public Vector getInorder() {
    Vector vec = new Vector();
    return traverseInorder( this, vec );
}

private Vector traverseInorder( BinaryTree b,
                               Vector v ) {
    if ( b != null ) { traverseInorder(
        b.left, v ); v.addElement(
        b.value ); traverseInorder(
        b.right, v );
    }
    return v;
}
```

20

La eficacia de los algoritmos

- Gran parte de la motivación para el diseño de árboles se debe a la compatibilidad que presentan con algoritmos eficaces.
- A continuación, haremos un pequeño alto en el camino para presentar los conceptos que utilizan los informáticos para analizar la eficacia de los algoritmos.

21

Búsquedas en una lista ordenada

- Comencemos con un ejemplo concreto: buscar un elemento específico en una lista ordenada.
- No hay forma más eficaz de hacerlo que comenzar por la primera entrada y analizar cada una hasta encontrar un elemento coincidente o hasta llegar al final de la lista.
- Si la lista está ordenada de forma ascendente, podremos reconocer una ausencia tan pronto como encontremos una clave mayor que la que buscamos. ¿Qué eficacia tiene este método?
- Por regla general, si la lista contiene n elementos, habrá que esperar a recorrer la mitad para encontrar la entrada.
- Si la lista contiene $2n$ elementos, de forma intuitiva habría que esperar el doble que si contuviese n elementos.

22

Análisis del tiempo de ejecución

- El análisis de un algoritmo se inicia dividiendo el cálculo en pasos bien definidos que tarden el mismo tiempo siempre que se ejecuten.
- Si hay bucles que se puedan ejecutar un número variable de veces y existen condiciones que puedan afectar al desarrollo del algoritmo, el análisis debería identificarlos para que se pueda calcular número medio y el peor número de repeticiones.

23

Costes de tiempo de ejecución

- El resultado de este análisis suele ser una suma de términos. Cada uno consta del número de veces que se repite una constante que representa el tiempo estimado de cada subtarea.
- La suma de la búsqueda lineal de una lista enlazada sencilla puede expresarse como $c_s + c_c * k$, donde
 - c_s = coste constante de definir una búsqueda, p.ej., invocar la máquina virtual de Java, analizar argumentos, etc;
 - c_c = coste de comprobar un elemento de la lista;
 - $k = 1$, en el mejor de los casos, esto es, si el elemento que se busca es el primero de la lista;
 $n/2$ en el caso habitual donde n = el número de elementos de la lista;
 n en el peor de los casos.

24

Tiempo de ejecución absoluto y relativo

- No es fácil calcular a priori el tiempo preciso que se tardará en ejecutar las subtarear. Depende de la CPU, la configuración del equipo, la eficacia del compilador, etc.
- Sin embargo, el recuento de repeticiones sí puede ser muy preciso. Se suele expresar como la función de una variable que representa algún aspecto de la envergadura del problema. En el caso de buscar en una lista enlazada simple, tal como vimos, la cantidad clave es la longitud de la lista en la que estamos buscando.
- Nuestro objetivo aquí *no* es adivinar el tiempo de ejecución exacto de un algoritmo concreto aplicado a un conjunto de datos específico, sino prever qué pasará con el tiempo de ejecución a medida que crece la envergadura del problema.

25

Eficacia algorítmica

- En nuestro ejemplo de búsqueda en una lista enlazada, el término c_s no nos importa demasiado, ya que no cambia a medida que la lista crece. Lo que sí es relevante es que podemos esperar un crecimiento lineal del tiempo de ejecución directamente proporcional a la envergadura del problema (la longitud de la lista).
- Si nuestro algoritmo tuviese dos o más bucles dependientes del número de elementos, la fórmula para el peor de los casos sería una suma de los términos dependientes de la variable n .

Por ejemplo,

$$n^2 + 5n$$

- En este caso, el término elevado al cuadrado es el importante. Decimos que el término n^2 *domina* al término $5n$ a medida que n crece.

26

Dominancia

- Algunos términos dominarán a otros a medida que n crece. Por ejemplo, en una suma de términos polinómicos, el término con mayor exponente es el que domina al resto.
- De la misma forma, n dominará a $\log n$ y 2^n dominará a n^a para cualquier constante a .

27

Notación $O()$

- Teniendo la dominancia en cuenta, el aumento del tiempo de ejecución de un algoritmo suele reducirse a un único término sin una constante.
- Así, el tiempo de búsqueda para una búsqueda lineal en una lista enlazada sencilla aumentará proporcionalmente a n , la longitud de la lista.
- De modo más formal, lo denominamos algoritmo $O(n)$ (leído “de orden n ”). El tiempo de ejecución de un algoritmo $O(n^2)$ aumentará más rápidamente (proporcionalmente al cuadrado de n) y en un algoritmo $O(\log n)$ lo hará más despacio.

28

Eficacia de los árboles

Uno de los muchos motivos por los que estudiamos los árboles reside en que, aunque las listas enlazadas ofrecen tiempos de búsqueda de $O(n)$, la mayoría de las implementaciones de árboles admiten tiempos de búsqueda de $O(\log n)$.

29

Ejercicio 3, Eficacia de los árboles

- Consideremos la búsqueda en una lista enlazada sencilla donde el tiempo medio de búsqueda viene dado por $time = .01 + 0.0001n$ segundos.
- Ahora, comparemos con un árbol donde el tiempo de búsqueda viene dado por $time = 0.1 + 0.001\log_2 n$ segundos.
- A pesar de que el término constante y el coeficiente son de un orden y magnitud mayores en el caso del árbol, calcule en qué árbol de tamaño n el rendimiento supera el de la lista.
- ¿Qué ocurre si aumentamos n una vez pasado ese punto en tres órdenes de magnitud?

30