Estructuras de datos
Clase teórica 6

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MEDELLÍN

Contenido

- Recursividad
- Árboles binarios

Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Recursividad / Recursión

Definición: "Un algoritmo es recursivo si se encuentra definido en términos de sí mismo".

Un ejemplo clásico de una función que puede definirse de manera recursiva es el factorial de un número:

N! = N*(N-1)!, sabiendo que 1! = 1

Ejemplo: 4! = 4*(3!) = 4*3*(2!) = 4*3*2*(1!) = 4*3*2*1 = 24

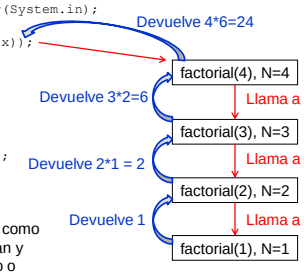
Salida de recursión

Como se observó en el ejemplo anterior, en una función recursiva debe existir por lo menos un caso donde la respuesta se obtiene sin necesidad de hacer más llamadas recursivas, a este caso se le denomina **salida de la recursión** (en el ejemplo anterior 1! = 1) y es necesario para que la función no se quede en un ciclo infinito.

La recursión es muy útil en ciertos problemas que por naturaleza son recursivos. La desventaja sin embargo es la gran cantidad de memoria que se ocupa debido al llamado continuo de funciones que se van acumulando. Cada llamado recursivo implica una copia en memoria de la función, junto con sus argumentos y variables internas.

Ejemplo

```
import java.util.*;

public class Main {

  public static void main(String[] args) {
    int x;
    Scanner entrada = new Scanner(System.in);
    x = entrada.nextInt();
    System.out.println(factorial(x));
  }

  static long factorial(int N) {
    if (N == 1)
      return 1;
    else
      return (N * factorial(N-1));
  }

}
```



*Notemos que los llamados se comportan como una … pila: entran (push) cuando se llaman y salen (pop) cuando devuelven el resultado o terminan

Llamados múltiples

Hacer el "rastreo" a una función recursiva cuando solo se hace un llamado por vez es relativamente simple (como en el ejemplo anterior). Sin embargo, cuando hay más de un llamado es más fácil de entender, no como una "pila", si no como un "árbol" de recursiones.

Miremos otro ejemplo clásico de una función que puede definirse de manera recursiva: la serie de Fibonacci

Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)

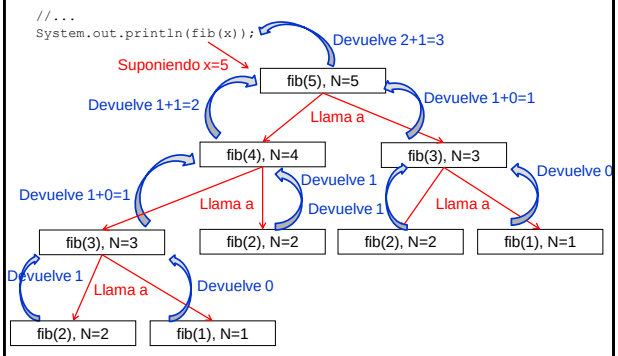
Sabiendo que Fibonacci(1) = 0, Fibonacci(2) = 1

*Recordemos: 0 1 1 2 3 5 8 13 21 34 …

Ejemplo

```
import java.util.*;

public class Main {

  public static void main(String[] args) {
    int x;
    Scanner entrada = new Scanner(System.in);
    x = entrada.nextInt();
    System.out.println(fib(x));
  }

  static long fib(int N) {
    if (N == 1)
      return 0;
    else if (N == 2)
      return 1;
    else
      return fin(N-1) + fib(N-2);
  }

}
```

Ejemplo



Llamados múltiples

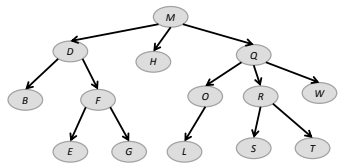
Habiendo entendido como se hacen llamados múltiples dentro de un proceso recursivo, hagamos una prueba de escritorio al siguiente código:

```
import java.util.*;

public class Main {

  public static void main(String[] args) {
    int x=6;
    System.out.println(julianacci(x));
  }

  static long julianacci(int N) {
    if (N <= 3)
      return N;
    else if (N % 2 == 0)
      return julianacci(N-1) + julianacci(N-2) + julianacci(N-3);
    else
      return julianacci(N-1) - julianacci(N-3);
  }

}
```

¿Qué es eso de árboles?

Expectativa

Realidad



Definición y terminología

Un árbol es una estructura de datos con relaciones jerárquicas entre sus componentes. Su principal característica es que tales relaciones son “de uno a muchos”, es decir, no son necesariamente lineales (de uno a uno) como en el caso de las listas enlazadas y generalmente son unidireccionales.

Nodo: De manera similar a como se trabaja en las listas enlazadas, los nodos son contenedores para almacenar los elementos del árbol.

Arista: Son las “líneas” (en realidad las referencias) que conectan a los nodos y que representan las relaciones (jerárquicas) entre ellos. Un nodo puede tener cero, una o varias aristas que salen de él.

Terminología

Padre: *X* es padre de *Y* si hay una relación directa que va de *X* a *Y*. En la figura de ejemplo *F* es padre de *E*. Nótese que todo nodo tiene máximo un solo padre.

Hijo: Consecuentemente, *Y* es hijo de *X* si hay una relación directa que va de *X* a *Y*. En el ejemplo anterior, *E* es hijo de *F*, así como *S* es hijo de *R*. Nótese que un solo nodo puede tener varios hijos.

Hermano: *Y* y *Z* son hermanos si tienen el mismo padre. En el ejemplo anterior, *E* es hermano de *G*.

Terminología

Nodo raíz: Es el primer nodo del árbol. Se caracteriza por ser el único nodo que no tiene padre.

Nodos hojas: Son aquellos nodos que no tienen hijos.

Camino simple: El camino simple entre dos nodos es la secuencia de nodos que hay que recorrer para llegar de uno a otro. En la figura de ejemplo, el camino entre los nodos *M* y *L* es *M - Q - O - L*

Longitud de camino: Se refiere al número de nodos que este contiene (también es igual al número de aristas recorridas mas uno). En el ejemplo anterior la longitud es 4.

Terminología

Antecesores: Si *X* es el nodo raíz entonces no tiene antecesores, de otro modo el padre de *X* es su antecesor, así como todos los antecesores del padre de *X*. Otra forma de verlo es que los antecesores de *X* son aquellos nodos que se encuentran en el único camino simple que va desde *X* (sin incluirlo) hasta la raíz del árbol. En la figura de ejemplo los antecesores de *F* son *D* y *M*

Sucesores: Si *Y* es un nodo hoja entonces no tiene sucesores. De otro modo, cada hijo de *Y* es su sucesor, así como todos los sucesores los hijos de *Y*. En la figura de ejemplo los sucesores de *D* son *B, F, E* y *G*.

Terminología

Subárbol: Dado algún nodo de un árbol, dicho nodo junto con todos sus sucesores, forman un subconjunto del árbol.

Nivel o profundidad: El nivel de un nodo se define como la longitud del camino simple entre él y la raíz. La raíz tiene un nivel de 1. En la figura de ejemplo el nodo *W* está en el nivel 3.

Altura del árbol: Se define como el máximo nivel presente en el árbol. En la figura de ejemplo la altura del árbol es 4.

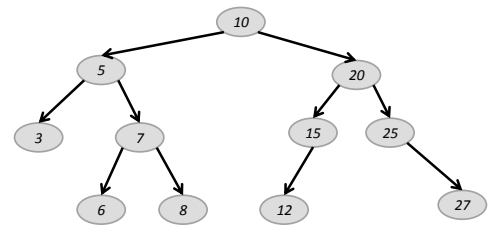
Árbol binario

Existen varios tipos de árboles, cada uno de ellos con propiedades especiales y usos específicos, pero en general todos se ajustan a la definición de árbol vista previamente.

Un árbol binario por ejemplo tiene las siguientes propiedades distintivas:

- Cada nodo puede tener como máximo 2 subárboles, y por tanto no puede tener más de dos hijos.
- Cada subárbol se identifica como el subárbol izquierdo o el subárbol derecho de su padre.
- Puede ser vacío.

Árbol binario



Recorridos típicos en árboles binarios

Preorden**:** Se procesa primero la raíz del subárbol, luego el subárbol izquierdo y por último el subárbol derecho.

```
//Este pseudo-código se limita a mostrar los elementos
void preOrder(nodo n){

   System.out.prinln(n.elm);

   if (n.hijoIzquierdo != null)
      preOrder(n.hijoIzquierdo);

   if (n.hijoDerecho != null)
      preOrder(n.hijoDerecho);
}
```

En la figura de ejemplo: 10, 5, 3, 7, 6, 8, 20, 15, 12, 25, 27

Recorridos típicos en árboles binarios

Enorden**:** Se procesa primero el subárbol izquierdo, luego la raíz del subárbol y por último el subárbol derecho.

```
//Este pseudo-código se limita a mostrar los elementos
void inOrder(nodo n){

  if (n.hijoIzquierdo != null)
      inOrder(n.hijoIzquierdo);

  System.out.prinln(n.elm);

  if (n.hijoDerecho != null)
      inOrder(n.hijoDerecho);
}
```

En la figura de ejemplo: 3, 5, 6, 7, 8, 10, 12, 15, 20, 25, 27

Recorridos típicos en árboles binarios

Posorden: Se procesa primero el subárbol izquierdo, luego el subárbol derecho y por último la raíz del subárbol.

```
//Este pseudo-código se limita a mostrar los elementos
void posOrden(nodo n){

   if (n.hijoIzquierdo != null)
      posOrder(n.hijoIzquierdo);

   if (n.hijoDerecho != null)
      posOrder(n.hijoDerecho);

   System.out.prinln(n.elm);
}
```

En la figura de ejemplo: 3, 6, 8, 7, 5, 12, 15, 27, 25, 20, 10