

1.00Clase 10

Recursión

Recursión

- La recursión es un enfoque orientado a la resolución de problemas, basado en la división y conquista (o división y combinación):

```
method Recurse(Arguments)
  if (SmallEnough(Arguments))           // Terminación
    return Answer
  else                                   // "Dividir"
    Identity= Combine( SomeFunc(Arguments),
                      Recurse(SmallerArguments))
  return Identity                       // "Combinar"
```

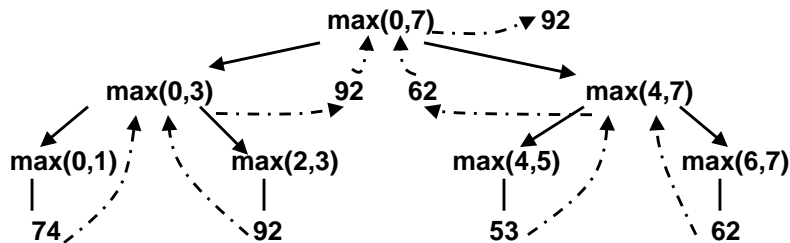
- Si puede escribir un problema como si fuera la combinación de pequeños problemas, puede implementarlo a través de un algoritmo recursivo de Java.

Hallando el valor máximo de un *array*

Suponga que sólo podemos hallar el valor máximo de 2 números cada vez. Imagine que queremos averiguar el valor máximo de un conjunto de 8 números.

35 74 32 92 53 28 50 62

El método recursivo *max* se llama a sí mismo:



Código para el método max

```
public class MaxRecurse {
    public static void main(String[] args) {
        int[] AData= {35, 74, 32, 92, 53, 28, 50, 62};
        System.out.println("Max: " + maxArray(0, 7, AData));
    }

    public static int combine(int a, int b) {
        return ( a >= b ) ? a : b;
    }

    public static int maxArray( int i, int j, int[] Arr) {
        if ( (j - i) <= 1) { // Suficientemente pequeño
            if (Arr[j] >= Arr[i])
                return Arr[j];
            else
                return Arr[i]; }
        else //Dividir y combinar
            return (combine(maxArray(i, (i+j)/2, Arr),
                maxArray((i+j)/2+1, j, Arr)));
    }
}
```

Código con más salida:

```
public class MaxRecurse2 {
    public static void main(String[] args) {
        int[] AData= {35, 74, 32, 92, 53, 28, 50, 62};
        System.out.println("Main Max:" + maxArray(0, 7, AData)); }
    public static int combine(int a, int b) {
        return ( a >= b) ? a : b; }
    public static int maxArray( int i, int j, int[] Arr) {
        System.out.println("Max(" + i + "," + j + ")");
        if ( (j - i) <= 1) {
            if (Arr[j] >= Arr[i]) { // Suficientemente pequeño
                System.out.println(" " + Arr[j]);
                return Arr[j]; }
            else {
                System.out.println(" " + Arr[i]);
                return Arr[i]; } }
        else { // Dividir y combinar
            int aa= (combine(maxArray(i, (i+j)/2, Arr),
                maxArray((i+j)/2+1, j, Arr)));
            System.out.println("Max(" + i + "," + j + ")= " + aa);
            return aa;
        } } }
```

Exponenciación

- La exponenciación, hecha "de forma simple", no es eficaz.
 - Elevar x a la potencia de y puede admitir y multiplicaciones:
 - Ej., $x^7 = x * x * x * x * x * x * x$
 - Calcular el cuadrado consecutivo es mucho más eficiente, pero hay que ser cauteloso en su implementación.
 - Por ejemplo: $x^{48} = (((x * x * x)^2)^2)^2$ utiliza 6 multiplicaciones en vez de 48.
- Informalmente, la exponenciación simple $O(n)$.
 - El cuadrado es $O(\lg n)$, dado que elevar un número a la enésima potencia conlleva $\lg n$ operaciones (base 2).
 - $\lg(48) = \log_2(48) =$ sobre 6
 - $2^{32} = 5; 2^{64} = 6$
 - Para hallar $x^{1,000,000,000}$, calcular el cuadrado conlleva 30 operaciones, mientras que con el método tradicional serían 1,000,000,000.

Exponenciación, cont.

- Los exponentes impares requieren un poco de más esfuerzo:
 - $x^7 = x \cdot (x \cdot x \cdot x)^2$ utiliza 4 operaciones en vez de 7.
 - $x^9 = x \cdot (x \cdot x)^2 \cdot 2$ utiliza 4 operaciones en vez de 9.
- Podemos generalizar estas observaciones y diseñar un algoritmo que utilice el cálculo de cuadrados para exponenciar más rápidamente.
- Sería posible escribir esto con iteración y seguir la pista a los exponentes pares e impares, pero puede resultar difícil.
- Es más lógico escribir un algoritmo recursivo:
 - Escribimos una serie de 3 identidades y las implementamos posteriormente como una función de Java.

Exponenciación, cont.

- Tres identidades:
 - $x^1 = x$ (lo bastante pequeño)
 - $x^{2n} = x^n \cdot x^n$ (reduce el problema)
 - $x^{2n-1} = x \cdot x^n$ (reduce el problema)

Exponenciación recursiva

```
method ExpResult(x, y)                //Pseudocódigo, aún no es código en Java
  if (y == 1)                          //y es lo suficientemente pequeño para escribir respuesta
    result = x
  else if (y mod 2 == 0)                 // ypar
    result = square( ExpResult(x, y/2))
  else                                   // yimpar
    result = x * ExpResult(x, y-1)
  return result
```

. Una identidad (condición) devuelve una respuesta real.

Un algoritmo recursivo debe poseer un caso lo bastante pequeño como para terminar la recursión final.

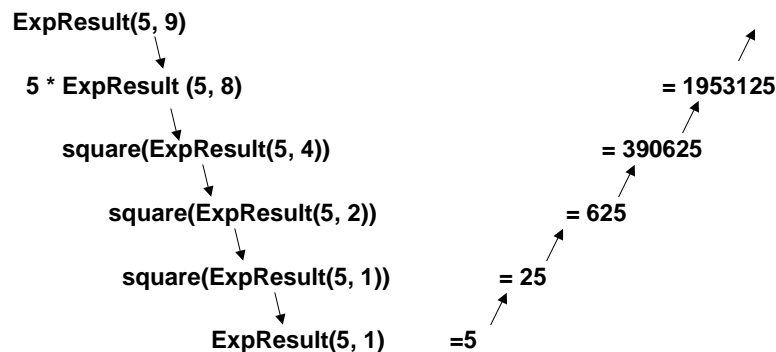
. Dos de las identidades (condiciones) disminuyen el problema en cada paso.

Un algoritmo recursivo ha de reducir el problema en cada paso.

. Estas son nuestras tres identidades.

Cómo funciona la recursión

x= 5, y= 9



Código de exponenciación

```
import javax.swing.*;

public class Exponentiation {
    public static void main(String[] args) {
        int z;
        String input= JOptionPane.showInputDialog("Introduzca x:");
        int x= Integer.parseInt(input);
        input= JOptionPane.showInputDialog("Introduzca y:");
        int y= Integer.parseInt(input);
        z= ExpResult(x, y);
        System.out.println(x + " elevado a " + y + " es: " + z);
    }
    // Utilice BigInteger para manejar números mayores. Algo tosco.
```

Código de exponenciación, 2ª parte

```
public static int ExpResult(int x, int y) {
    int result;
    if (y == 1) // y lo bastante pequeño
        result= x;
    else if (y % 2 == 0) { // y par
        int term= ExpResult(x, y/2); // No haga 2 llamadas recursivas
        result= term * term; } // cuadrado(ExpResult(x, y/2))
    else // y impar
        result= x * ExpResult(x, y-1);
    return result; // Añada println para trazar
}

// "Combinar"
```

Recursión e iteración

- **Escribir la exponenciación iterativamente es una tarea difícil.**
 - Inténtelo si está interesado y dispone de tiempo.
- **Normalmente, es más fácil ver una implementación recursiva correcta.**
 - La recursión con frecuencia se acerca más a la matemática subyacente.
- **Existe un modo mecánico, empleado por compiladores y diseñadores de algoritmos, encargado de convertir la recursión en iteración. Es complejo y se utiliza para mejorar la eficiencia.**
 - El consumo de las llamadas a métodos es evidente, y la conversión de recursión a iteración dentro de un método acelera la ejecución.
 - Los métodos pequeños o utilizados con poca frecuencia pueden quedar como recursivos.

Ejercicio: Serie de Fibonacci

- **Si ha traído su ordenador portátil, vaya a la página Web del curso:**
web.mit.edu/1.00/www/Lecture10/recursionExercise.html
- **Si no lo ha traído consigo, siga la pantalla.**

Resumen

- El método recursivo se llama a sí mismo, directa o indirectamente(en la práctica , casi siempre directamente).
- La recursión está basada en dividir y conquistar:
 - Reducir un problema original a una secuencia de instancias menores, hasta que éstas sean lo suficientemente pequeñas como para resolverse directamente.
 - Luego, combinar las soluciones de los subproblemas para construir la solución del problema original.
- La recursión es uno de los aspectos de cómo funciona un grupo de lenguajes no procedimentales.
 - Javaesproced imental: se le indica al ordenador qué hay que hacer y el modo de hacerlo(comoC++,VisualBasic ...).
 - Lenguajes no procedimentales: se le indica al ordenador qué hay que hacer y éste resuelve el modo de hacerlo (Prolog).