Algoritmos y Estructuras de Datos

Recursión o Recursividad

1

TIEMPO DE EJECUCIÓN DE UN PROGRAMA

- Al resolver un problema, debemos elegir el algoritmo a utilizar, siguiendo los siguientes objetivos:
  - algoritmo fácil de entender, codificar y depurar.
  - uso eficiente de los recursos y ejecución veloz.
- En la mayoría de los casos, estos objetivos son contrapuestos.
- Depende, en gran forma, de la cantidad de veces que el programa se va a ejecutar.

2

RECURSIÓN – algunas definiciones típicas

- Es la forma en la cual se especifica un proceso basado en su propia definición.
- Definición de un problema en instancias más pequeñas de sí mismo. Conociendo la solución explícita de las instancias más simples (CASOS BASE), se puede aplicar inducción para resolver el caso general.
- Recursión: cuando un método se llama a sí mismo.
- Un objeto es recursivo si:
  - en parte está formado por sí mismo, o
  - está definido en función de sí mismo
- Ejemplos en matemáticas:
  - número natural:
    - 0 pertenece a N
    - Si n pertenece a N, entonces n+1 pertenece a N
  - Factorial , Potencia
  - Números de Fibonacci

3

Ejemplo : Factorial

UCU

- Definición recursiva

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$$

- Como método :

```
// función factorial recursiva
public static int factorial(int n) {
   SI (n == 0) devolver 1;          // caso base
   SINO,
      devolver (n * factorial(n- 1));          // caso recursivo
}
```


RECURSIÓN

UCU

- Una función es recursiva cuando se invoca a sí misma.
- Es útil cuando el problema a resolver o la estructura de datos a recorrer son de naturaleza recursiva.
- Evaluar muy bien para otros casos, y evitar su uso si existe una solución no recursiva aceptable.

RECURSIÓN

UCU

- Permite definir un conjunto infinito de objetos mediante una proposición finita.
- Un número infinito de cálculos puede describirse mediante un programa recursivo finito, sin repeticiones explícitas.
- Los algoritmos recursivos son idóneos principalmente cuando están definidos en forma recursiva:
  - el problema a resolver, o
  - la función por calcular, o
  - la estructura de datos por procesar

RECURSIÓN

- Si un módulo **P** contiene una referencia explícita a sí mismo, se dice que es **directamente recursivo**. Técnica ampliamente usada para la resolución de problemas.
- Si un módulo **P** contiene una referencia a un módulo **Q** que a su vez referencia a **P**, entonces se dice que **P** es **indirectamente recursivo**. Técnica de **cuidadosa aplicación**, o error que debe ser evitado.
- Debe tenerse en cuenta el problema de la t**erminación**; por ejemplo, si un módulo **P** de tamaño "n**"** consiste de una secuencia "**S**" de sentencias:
  - P(n) ≡ P[S, Si n > 0 entonces P(n-1)]

7

RECURSIÓN

- Soluciones a funciones matemáticas recursivas: escribir un algoritmo en seudocódigo para:
  - Hallar el factorial de un número natural "N".
  - Hallar la potencia de un número "N" elevado a un exponente "e".
  - Hallar el número de Fibonacci correspondiente a la posición "N" de la sucesión.

8

USO DE LA RECURSIÓN – una forma usual

Función ***A*** *(N: tamaño del problema)*
COM
  <bloque>
  Si <condición> entonces
    <bloque>
    A(N-1)
    <bloque>
  Sino
    <bloque>
  Fin Si
  <bloque>
FIN

9

Recursión Lineal

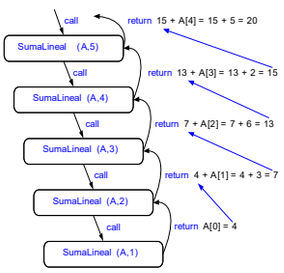
- Probar los casos base.
  - **Comenzar** probando un conjunto de casos base (debe haber **por lo menos uno**).
  - Cada cadena de llamadas recursivas potencial **debe** eventualmente alcanzar un caso base, y el manejo del caso base **NO DEBE USAR RECURSIVIDAD**.
- Recurrir una vez.
  - Realizar una sóla llamada recursiva. (este paso puede involucrar una prueba para decidir cuál de varias posibles llamadas recursivas diferentes ejecutar, pero en última instancia **sólo se ha de ejecutar una**)
  - Definir cada posible llamada recursiva de forma tal que **progrese hacia un caso base**.

Ejemplo simple de recursión lineal

Algoritmo SumaLineal(A, n):
Entrada: Un array de enteros *A* y un entero *n* >= 1, tal que A tiene al menos *n* elementos
Salida:
La suma de los primeros *n* enteros en *A*

```
COM
SI n = 1
  devolver A[0]
SINO
  devolver SumaLineal(A, n - 1) + A[n - 1]
FIN
```

Ejemplo de secuencia recursiva:
A = {4,3,6,2,5} y *n* = 5



Invertir un Array

Algoritmo ***InvertirArray(A, i, j)***
  Entrada: Un array A de enteros e índices enteros no negativos i y j
  Salida: Los elementos de A entre los índices i y j, invertidos

```
COM
  SI i <  j then
      Intercambiar A[i] y A[ j]
      InvertirArray(A, i + 1,  j - 1)
  FINSI
FIN
```

Cálculo de la potencia

- La función potencia, $p(x,n)=x^n$, puede ser definida recursivamente:

$$p(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot p(x, n-1) & \text{else} \end{cases}$$

- Esto conduce a una función potencia que se ejecuta en un tiempo O(n) (puesto que hacemos n llamadas recursivas).

Recursión de cola

- Ocurre cuando un método linealmente recursivo hace su llamada recursiva como último paso.
- El método de inversión de array es un ejemplo.
- **Estos métodos pueden ser fácilmente convertidos en métodos no-recursivos (lo que ahorra algunos recursos).**
- Ejemplo:
- Algoritmo ***InvertirArrayIterativo(A, i, j)***
  Entrada: Un array A de enteros e índices enteros no negativos i y j
  Salida: Los elementos de A entre los índices i y j, invertidos

```
COM
    MIENTRAS i <  j do
      INTERCAMBIA A[i] con A[j]
      i  = i + 1
      j  = j – 1
    FINMIENTRAS
FIN
```


Recursión Binaria

- Ocurre cada vez que hay **dos llamadas recursivas** para cada caso no-base
- Ejemplo: Números de Fibonacci
- Los números de Fibonacci se definen recursivamente:
  - F0 = 0
  - F1 = 1
  - Fi = Fi-1 + Fi-2 para todo i > 1.
- Algoritmo recursivo (primer versión):

Algoritmo *FibonacciBinario(k)*
Entrada: entero k no negativo
Salida: el k-esimo número de Fibonacci
COM
SI k = 0 o k = 1
devolver k
SINO
devolver ***FibonacciBinario (k - 1)*** + ***FibonacciBinario (k - 2)***
FINSI
FIN

Análisis del Algoritmo Recursivo Binario de Fibonacci

n_k indica el número de llamadas recursivas hechas por ***FibonacciBinario (k)***. Entonces

- $n_0 = 1$
- $n_1 = 1$
- $n_2 = n_1 + n_0 + 1 = 1 + 1 + 1 = 3$
- $n_3 = n_2 + n_1 + 1 = 3 + 1 + 1 = 5$
- $n_4 = n_3 + n_2 + 1 = 5 + 3 + 1 = 9$
- $n_5 = n_4 + n_3 + 1 = 9 + 5 + 1 = 15$
- $n_6 = n_5 + n_4 + 1 = 15 + 9 + 1 = 25$
- $n_7 = n_6 + n_5 + 1 = 25 + 15 + 1 = 41$
- $n_8 = n_7 + n_6 + 1 = 41 + 25 + 1 = 67$.

```
Algoritmo FibonacciBinario(k)
COM
SI k = 0 o k = 1
 devolver k
 SINO
    devolver
          FibonacciBinario (k - 1) +
          FibonacciBinario (k - 2)
FINSI
FIN
```


Un mejor algoritmo de Fibonacci Utilizando recursión lineal

Algoritmo *FibonacciLineal(k)*

Entrada: Un entero no negativo k
Salida: Par de números de Fibonacci (F_k, F_{k-1})

```
COM
    si k = 1
          devolver (k, 0)
      sino
          (i, j) = FibonacciLineal (k - 1)
          devolver (i +j, i)
FIN
```

- Ejecuta en tiempo $O(k)$.

RECURSIÓN

TDALista. Invertir

```
COM
    Si (no Vacía) entonces
        x ← Primero
        Eliminar (Primero)
        Invertir
        InsertarÚltimo(x)
    Fin Si
FIN
```

RECURSIÓN – definición recursiva de LISTA

- Problemas y estructuras de datos abstractas recursivas.
- Una Lista es vacía o está formada por un elemento llamado “primero”, seguido de una lista:
  - L(n) = {}, n=0;
  - L(n) = {primero, L(n-1)}, n > 0

PROBLEMAS Y EJERCICIOS

- El problema de las 8 reinas – Wirth, 3.5
- Ejercicios Wirth
  - Torres de Hanoi
  - Permutaciones de elementos
- Ejercicios Weiss
  - 7.3.1 impresión de números en cualquier base
  - ¿qué problemas podemos tener con la recursión?
  - Máximo común divisor (7.4.3)
- Resuelve los ejercicios de fin del capítulo 7 del libro de Weiss (cuestiones breves, Problemas teóricos y Problemas prácticos)

Ejercicios

Implementar en java:

- Fibonacci recursivo
- Imprimir la lista al revés.
- Invertir la lista

Links sobre recursividad

UCU

http://es.wikipedia.org/wiki/Recursividad
http://es.wikipedia.org/wiki/Algoritmo_recursivo
http://es.wikipedia.org/wiki/Factorial
http://es.wikipedia.org/wiki/Sucesion_de_Fibonacci
http://es.wikipedia.org/wiki/Torres_de_Hanoi

- Animación de algoritmos:

http://users.encs.concordia.ca/~twang/WangApr01/RootWang.html
http://www.mazeworks.com/hanoi/index.htm
http://www.raptivity.com/Demo Courses/Interactivity Builder Sample Course/Content/Booster Pack/HTML Pages/Towers of Hanoi.html