Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2017

Guía Práctica No. 3: Divide & Conquer

Ej. 1. Lea el capítulo *Divide & Conquer* de *Introduction to the Design and Analysis of Algorithms* (Levitin).

Ej. 2. Dado un arreglo `A[0..n-1]` de números reales distintos, llamaremos *inversión* a un par de valores `(A[i],A[j])` almacenados en el arreglo tales que `A[i]` $>$ `A[j]`, con `i` $<$ `j`. Diseñe, utilizando Divide & Conquer, un algoritmo que calcule el número de inversiones en un arreglo dado. Calcule el tiempo de ejecución en peor caso de su algoritmo, e impleméntelo en Java.

Ej. 3. Aplicando el Teorema Maestro, decida cuál es la tasa de crecimiento de las siguientes ecuaciones de recurrencia:

- $T(1) = 1; T(n) = 4T(\frac{n}{2}) + n$
- $T(1) = 1; T(n) = 4T(\frac{n}{2}) + n^2$
- $T(1) = 1; T(n) = T(\frac{n}{2}) + 1$
- $T(1) = 1; T(n) = 2T(n-1) + 2n$
- $T(1) = 1; T(n) = T(\frac{n}{2}) + 2^n$
- $T(1) = 1; T(n) = 2T(\frac{n}{2}) + \log n$

Ej. 4. Diseñe usando Divide & Conquer e implemente en Java un algoritmo para resolver el problema de encontrar la subsecuencia de suma mínima, de una secuencia dada. Realice el análisis de tiempo de ejecución para el peor caso de su algoritmo. En caso de tener complejidad superior a $O(n \times \log\ n)$, optimice su algoritmo para conseguir tal eficiencia.

Ej. 5. Diseñe usando Divide & Conquer e implemente en Java un algoritmo para resolver el problema de multiplicar dos matrices (ver Strassen Matrix Multiplication en *Introduction to the Design and Analysis of Algorithms*).

Ej. 6. Implemente en Java o Haskell el algoritmo para resolver el problema de, dado un conjunto de puntos del plano, encontrar el par de puntos (distintos) cuya distancia es la menor entre todos los puntos del plano, en $O(n \log n)$, discutido en *Introduction to the Design and Analysis of Algorithms* (Levitin). Explique además las razones por las cuales no es necesario comparar *todos* los pares de puntos en las inmediaciones de la línea media, sino que para cada punto a uno de los lados de esta línea, sólo hace falta considerar a lo sumo 6 puntos del otro lado.

Ej. 7. Una solución Divide & Conquer (D&C) a un problema determinado consta de:

- un criterio *isBase* de distinción para las entradas, que separe los casos base y los casos complejos,
- un proceso directo *base* de resolución de los casos base,

- un proceso *split* que, dada una entrada compleja (no base), descomponga a la misma en un número de entradas más simples,
- un proceso *join* que, dadas las soluciones a las entradas más simples creadas por split, construya una solución al problema original.

Dados estos elementos, se puede construir un algoritmo D&C para resolver el problema. En Haskell, se puede sintetizar este proceso mediante una función de alto orden, que tome por entradas todos los elementos anteriores que caracterizan una solución D&C, y los combine para conseguir implementar la función correspondiente:

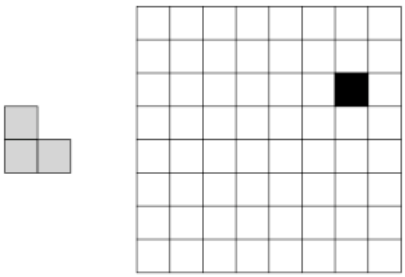
```
evalDC::Eq a⇒(a→Bool)→(a → b)→(a→[a])→([b]→b)→a→b
evalDC isBase base split join x | isBase x = base x
                                | otherwise = join (map ( evalDC isBase base split join) (split x ))
```

Usando esta función, se pueden construir soluciones D&C simplemente identificando e implementando adecuadamente *isBase*, *base*, *split* y *join*. Por ejemplo, la función que computa números de Fibonacci puede definirse usando *evalDC* de la siguiente manera:

```
fibonacci = evalDC (\x→x<=1) (\x→1) (\x→[x-1,x-2]) (\l→(head l)+(last l))
```

Considere el problema de, dada un par de secuencias p y t, determinar si p es subsecuencia (de elementos contiguos) de t. Identifique las funciones isBase, base, split y join para resolver este problema mediante D&C, y utilice evalDC para conseguir el programa que resuelve el problema.

Ej. 8. Un *trominó* es una figura en forma de L formada por tres casillas adyacentes. El problema de los trominós consiste en cubrir una grilla de $2^n \times 2^n$ casillas, con una única casilla ya cubierta (que puede estar en cualquier posición en la grilla), con trominós. Los trominós deberán cubrir toda el área de la grilla, con excepción del punto ya cubierto, sin solaparse unas con otras.



Diseñe una solución Divide & Conquer para el problema de los trominós, e implemente su solución en Haskell.

Ej. 8. Expliqué cuál es la relación entre *inducción* y *recursión*. Si le parece que hay una correspondencia entre estos conceptos, a qué le parece que corresponde la *hipótesis* inductiva en la definición de una función recursiva?