Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2016

Guía Práctica No. 2: Fuerza Bruta

Ej. 1. Lea el capítulo 3 de "Introduction to the Design and Analysis of Algorithms" (Levitin 2003).

Ej. 2. Escriba programas Haskell que realicen las siguientes tareas:

- dada una lista, genere todas las permutaciones de la misma,
- dado un conjunto representado como una lista, genere todos los subconjuntos del mismo,
- dada una lista, genere todas las sublistas de la misma.

Ej. 3. Escriba un programa Haskell que implemente el algoritmo de ordenamiento *SlowSort*, que ordena una lista de elementos (por ejemplo, de enteros) mediante la generación de la lista de todas las permutaciones de la lista original, y filtrando aquella que está ordenada.Utilice su cuenta en codeboard para acceder al template de este algoritmo donde podrá encontrar también una serie de test para probar su programa.

Ej. 4. De los algoritmos para computar el máximo común divisor descriptos en el capítulo 1 de "Introduction to the Design and Analysis of Algorithms" (Levitin 2003), cuál consideraría correspondiente a la técnica de fuerza bruta?

Ej. 5. Considere el problema de las *8 Reinas*, el cual consiste en ubicar ocho reinas en un tablero de ajedrez, de manera tal que éstas no se "ataquen" mutuamente. Escriba un programa Java que resuelva este problema mediante fuerza bruta (búsqueda exhaustiva). Podrá encontrar el template de este algoritmo en codeboard y una serie de test para probar su solución.

Ej. 6. Escriba un programa Java que, mediante la técnica de fuerza bruta encuentre los dos números de cinco dígitos (incluyendo ceros no significativos) que sean capicúas, y cuya distancia entre ellos sea la mínima posible, entre todos los pares de capicúas distintos. Podrá encontrar el template de este algoritmo en codeboard y una serie de test para probar su solución.

Ej. 7. Escriba un programa Java que encuentre todas las soluciones al problema "send+more=money", que asignando dígitos a cada uno de los caracteres de las palabras send, more y money se cumpla que send+more=money. (Ejemplo: send $\rightarrow$ 9789, more $\rightarrow$ 1087, money $\rightarrow$ 10876.)

Ej. 8. El algoritmo de ordenamiento *BubbleSort* puede pensarse como una ligera mejora al algoritmo *SlowSort*, en la cual en lugar de generar todas las permutaciones, sólo se permutan items de la lista si éstos están desordenados. Compare empiricamente *SlowSort* con *BubbleSort*, para evaluar las consecuencias de esta sencilla modificación. Podrá encontrar el template de este algoritmo en codeboard y una serie de test para probar su solución.