Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2022

Guía Práctica No. 1: Repaso y Fuerza Bruta

Esta guía práctica abarca tanto revisión de conceptos previos como una de las técnicas algorítmicas más básicas, Fuerza Bruta. Refiere por lo tanto a los capítulos 1, 2 y 3 de *Introduction to the Design and Analysis of Algorithms* (Levitin 2003), que son lectura recomendada para el desarrollo de la práctica.

Parte de esta práctica cuenta con esquemas de programación y tests de asistencia a su resolución. Éstos pueden accederse a partir del classroom de github, para el cual es conveniente tener una cuenta en Github (`http://github.com`).

Esta práctica no tiene entrega formal. Su resolución es opcional.

1. Considere la clase `ArraySorter` provista en el repositorio. Implemente los algoritmos de ordenamiento `selectionSort` y `bubbleSort`. Asegúrese que todos los tests pasen (debe completar la implementación de algunos métodos auxiliares). Complete el conjunto de tests de unidad provistos, con otros adicionales, para los algoritmos implementados. Utilice el proyecto para realizar una comparación experimental del tiempo de ejecución de los algoritmos implementados.

2. Complete la implementación de los algoritmos para computar el máximo común divisor presentados en la sección 1.1 de Introduction to the "Design and Analysis of Algorithms" (Levitin 2003). Realice además experimentos para comparar los tiempos de ejecución de los diferentes algoritmos.

3. Implemente en Java el problema de coloreo de grafos. El problema consiste en colorear todos los nodos de un grafo no dirigido, de manera tal que no existan en el grafo dos nodos adyacentes con el mismo color, usando la menor cantidad posible de colores.

4. Para los siguientes problemas, describa en palabras y/o pseudo-código, algoritmos que los resuelvan basándose directamente en la descripción del problema:

   - Decidir si un conjunto de enteros se puede particionar en dos conjuntos de igual suma.
   - Dadas dos cadenas, decidir si las mismas son anagramas (una es permutación de la otra).
   - Dado un número natural n, descomponerlo en sus factores primos.
   - Dadas dos cadenas p y s, decida si p es subcadena de s.
   - Dadas dos cadenas p y s, decida si p es subsecuencia de s (los elementos no necesariamente tienen que aparecer contiguos en s).
   - Dada una secuencia s de números, encontrar el k-ésimo elemento más grande en s.

5. Escriba programas en Java y Haskell que calculen toda las permutaciones de una lista, todos los subconjuntos de un conjunto, y todas las sublistas de una lista. Utilizando estas rutinas, resuelva los siguientes problemas:

   - Decidir si dos secuencias son anagramas.
   - Dado un conjunto s y un valor n, decidir si existe un subconjunto de s cuya suma sea n.
   - Dadas dos cadenas p y s, decida si p es subcadena de s.
   - Dadas dos cadenas p y s, decida si p es subsecuencia de s (los elementos no necesariamente tienen que aparecer contiguos en s).

diferencia entre estas dos?

6. 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. Complete el entorno de algoritmos de ordenamiento para Java con una implementación de *slowSort*.

7. Diseñe, e implemente en Java o Haskell, un algoritmo que elimine elementos repetidos de una lista. Argumente sobre la complejidad en peor caso de su algoritmo.

8. Considere la clase `CaesarCracker`, provista en el repositorio, y cuyo propósito es desencriptar, mediante fuerza bruta, un mensaje ASCII cifrado mediante un cifrado César. Implemente el método `decode`, que dado un mensaje y la clave, desencripta el mismo.

9. Considere nuevamente la clase `CaesarCracker`. Implemente el método `bruteForceDecrypt()`, que intenta desencriptar, por fuerza bruta, un mensaje encriptado, probando con todas las claves posibles de hasta tamaño k (véase atributo `passwordLength`) y utilizando como criterio de desencripción exitosa que el mensaje desencriptado contenga una palabra específica (véase atributo `messageWord`). Argumente sobre la complejidad de su solución.

10. Implemente en Java o Haskell un algoritmo para resolver el problema de, dado un conjunto de puntos del plano, encontrar el polígono convexo más pequeño que incluya todos los puntos del conjunto dado, discutido en Introduction to the Design and Analysis of Algorithms (Levitin).