Práctica 6

Técnicas de Programación

Introducción a la Computación

1^{er} cuatrimestre 2018

DIVIDE & CONQUER

Ejercicio 1.

(a) Escribir un algoritmo *divide and conquer* en pseudocódigo que dados $a, n \in \mathbb{N}$ devuelva a^n. Ayuda: $a^n = a^{\lceil \frac{n}{2} \rceil} \times a^{\lfloor \frac{n}{2} \rfloor}$.

(b) Comparar la complejidad del algoritmo propuesto con uno iterativo que compute la misma función.

(c) Implementar el algoritmo en Python. Sugerencia: usar las funciones `ceil` y `floor` del módulo `math` de Python.

Ejercicio 2.

(a) Escribir un algoritmo *divide and conquer* en pseudocódigo que dado un arreglo de enteros devuelva la posición del elemento más grande.

(b) ¿Cuál es el resultado del algoritmo propuesto si el máximo elemento aparece repetidas veces en el arreglo?

(c) Implementar el algoritmo en Python.

Ejercicio 3. Sea A un arreglo de números enteros cuya longitud es potencia de 2. Decimos que A es *más a la izquierda* si:

1. la suma de los elementos de la mitad izquierda es mayor a la suma de los de la mitad derecha;
2. cada una de las mitades es, a su vez, más a la izquierda.

Por ejemplo, [8, 6, 7, 4, 5, 1, 3, 2] es más a la izquierda, pero [8, 4, 7, 6, 5, 1, 3, 2] no.

(a) Escribir en pseudocódigo un algoritmo *divide and conquer* que determine si un arreglo es más a la izquierda.

(b) Implementar el algoritmo en Python.

Ejercicio 4. Sea A un arreglo de números enteros. Decimos que un par $\langle A[i], A[j] \rangle$ es una *inversión* si $i < j$ y $A[i] > A[j]$.

(a) Escribir en pseudocódigo un algoritmo *divide and conquer* $\in O(|A| \times log(|A|))$ que dado un arreglo de enteros devuelva la cantidad de inversiones. Sugerencia: modificar el algoritmo merge sort.

(b) Implementar el algoritmo en Python.

Ejercicio 5. Un *L-rompecabezas* es un juego que en el que hay *l-fichas* en forma de L formadas por tres cuadrados adyacentes y un tablero de $2^n \times 2^n$ casilleros. El tablero contiene una posición distinguida llamada *agujero negro*. A modo de ejemplo, el siguiente dibujo representa un tablero de 8×8 con 3 *l-fichas*:

	[gray]0.8	[gray]0.8					
	[gray]0.8						
	[gray]0.8						
[gray]0.8	[gray]0.8				[gray]0.1		
						[gray]0.8	[gray]0.8
						[gray]0.8	

El objetivo del juego consiste en, dado un tablero vacío con exactamente un *agujero negro*, usar *l-fichas* para cubrir todas las posiciones del tablero, sin superponer fichas y sin cubrir el *agujero negro*. Se pide diseñar una estrategia *divide and conquer* para resolverlo.

BACKTRACKING

Ejercicio 6.

(a) Escribir un algoritmo en pseudocódigo que dado un arreglo de enteros genere todas sus permutaciones usando *backtracking*.

(b) Implementar el algoritmo en Python.

Ejercicio 7.

(a) Escribir un algoritmo en pseudocódigo que dada una matriz $M \in \{0,1\}^{n \times m}$ determine si existe un camino de '0's desde $M[0][0]$ a $M[n-1][m-1]$. Sólo se permite avanzar en forma horizontal o vertical.

(b) Implementar el algoritmo en Python.

Ejercicio 8.

(a) Escribir un algoritmo en pseudocódigo para determinar si una palabra se encuentra presente en una *sopa de letras*. Es decir, dadas una matriz de $n \times m$ caracteres y una palabra p, determinar si p se encuentra escrita en la matriz, pasando de una letra a otra adyacente, ya sea vertical u horizontalmente. Por ejemplo, en la siguiente sopa de letras de 5×4 se encuentra la palabra *altos*, pero no *dibujo*:

a	l	d	r
q	r	y	e
p	d	b	u
i	s	l	a
f	o	t	e

(b) Implementar el algoritmo en Python.

Ejercicio 9.

(a) Se tienen $n \geq 1$ ciudades distintas representadas como coordenadas (x, y) en un plano. La distancia entre dos ciudades se calcula como la distancia euclideana entre ambas. Un viajante de comercio debe recorrer todas las ciudades exactamente una vez y volver a la ciudad de inicio. Escribir un algoritmo de *backtracking* que dada una lista de ciudades encuentre el recorrido con la menor distancia. Suponer que empieza y termina en la primera ciudad de la lista pasada como parámetro.

(b) Implementar el algoritmo en Python.

Ejercicio 10.

(a) El *problema del caballo* es un antiguo problema matemático en el que se pide que, dado un tablero de $n \times n$ casilleros y un caballo de ajedrez colocado en una posición cualquiera (x, y), el caballo pase por todas las casillas y una sola vez por cada una. Dar un algoritmo en pseudocódigo que dado un tablero y una posición inicial devuelva si existe o no tal recorrido.

(b) Implementar el algoritmo en Python.