ALGORITMIA

Grado en Ingeniería Informática del Software - Curso 2021-2022

GUIÓN DE LA PRÁCTICA 3

OBJETIVO:

- **Divide y Vencerás: modelos recursivos y ejemplos**

Modelos recursivos básicos

El paquete **alg77777777.p3** contiene diez clases siguientes:

Las clases **Sustraccion1.java** y **Sustraccion2.java** tienen un esquema por *SUSTRACCIÓN* con a=1, lo que lleva consigo un **gran gasto de pila**, de hecho se desborda cuando el tamaño del problema crece hasta algunos miles. Afortunadamente los problemas así solucionados, van a tener una mejor solución iterativa (con bucles) que la solución de este tipo (sustracción con a=1).

La clase **Sustraccion3.java** tiene un esquema por *SUSTRACCIÓN* con a>1, lo que lleva consigo un **gran tiempo de ejecución** (exponencial o no polinómico). Esto supone que para un tamaño del problema de algunas decenas el algoritmo no acaba (*tiempo intratable, NP*). La consecuencia es que debemos procurar no plantear soluciones del tipo *SUSTRACCIÓN* con varias llamadas (a>1)

Las clases **Division1.java**, **Division2.java** y **Division3.java** tienen un esquema por *DIVISIÓN*, siendo la primera del tipo $a<b^k$, la segunda del tipo $a=b^k$ y la última del tipo $a>b^k$ (ver cómo se definen estas constantes en teoría).

SumaVector1.java resuelve de tres formas diferentes el sencillo problema de sumar los elementos de un vector y **SumaVector2.java** mide tiempos de esos tres algoritmos para diferentes tamaños del problema.

Fibonacci1.java resuelve de cuatro formas diferentes el problema de calcular el número de Fibonacci de orden n y **Fibonacci2.java** mide tiempos de esos cuatro algoritmos para diferentes tamaños del problema.

Escribir las siguientes clases recursivas para obtener las siguientes complejidades de ejecución:

- **Sustraccion4.java**, método recursivo POR SUSTRACCION con una complejidad $O(3^{n/2})$
- **Division4.java**, método recursivo POR DIVISIÓN con una complejidad $O(n^2)$ y el número de subproblemas = 4.

TRABAJO PEDIDO

Realizar un análisis de las complejidades y empírico de las 10 clases anteriores. Estudiando el código y ejecutándolo. Escribir el código para Sustracción4.java y Division4.java

Las clases que programe las incluirá dentro del paquete alg<dnipropio>.p31 , si utiliza Eclipse llamar al proyecto prac03_Recursivo<UOpropio>
Se entregará en el campus virtual según el calendario previsto.

Problema Divide y Vencerás

Poliominó

En el documento adjunto (http://culturacientifica.com/2014/07/16/embaldosando-con-l-triominos-un-ejemplo-de-demostracion-por-induccion/) tiene un estudio interesante del problema de los poliominós. De su lectura se deduce que para los trominós hay dos casos: L-trominó (ficha de 3 cuadrados en L) e I-trominó (ficha de 3 cuadrados en línea o I).

En el otro documento adjunto (*TrominoDescripcion.pdf*), se analiza cómo colocar L-trominós en un tablero de lado n (n es potencia de 2, esto es: 2, 4, 8, 16, …). El algoritmo expuesto es claramente "Divide y vencerás".

Pues bien, elijamos como problema a resolver ese de colocar L-trominós en un tablero de lado n (obligatoriamente potencia de 2). Queda claro que ese problema solamente es un caso particular de un problema mucho más general expuesto en el primer documento antes apuntado.

Visto lo anterior, se pide:

1. Implementar en Java la solución.
   Para probar el algoritmo puede meter casos sencillos (vía argumentos de ejecución) para n bajas (n=2, 4, 8, 16) y compruebe que la solución hallada es la correcta.

   Por ejemplo, suponiendo que la clase se llama Tromino.java, para un tablero 8x8 con el cuadro inicial "exento" en posición (3,5) podría ser:

   ```
   java Tromino 8 3 5
   ```

2. Una vez probado el funcionamiento correcto. Hacer una clase `TrominoTiempos` que sirva para ir aumentando el tamaño del problema (n= 16, 32, 64, 128, 256, …, hasta que se desborde el HEAP) y mida los tiempos de ejecución para cada tamaño del problema. Para medir los tiempos, la posición inicial del cuadro "exento" se puede establecer aleatoriamente (no se debe escribir la solución por pantalla).

n	*t Trominó*
16	…..
32	…..
64	…..
128	…..
256	…..
…	…..
Oveflow heap	

3. Analizar la complejidad temporal teórica del algoritmo y comprobar si los tiempos obtenidos en el apartado anterior cumplen o no con esa complejidad.

Trabajo optativo

Como parte extra puede extender el problema en alguna de las líneas siguientes:

- Paralelizar el algoritmo presentado, en línea a lo visto al respecto en clase de teoría. Como conclusión mida tiempos, para así ver qué mejora temporal se consigue.

- Intentar resolver el mismo problema antes propuesto (L-trominó en un tablero de lado potencia de 2) mediante otro algoritmo diferente (al aquí propuesto "divide y vencerás"), y como conclusión comparar tiempos.

- Intentar resolver el problema L-trominó para tableros de lado que no sea potencia de 2 (deficientes según terminología del artículo antes citado).

- Intentar resolver el problema I-trominó, o el dominó (fichas de 2 cuadrados) o lo que sería aún más ambicioso: alguno de los tetraminó (fichas de 4 cuadrados) o pentaminó (fichas de 5 cuadrados).

Las clases que programe las incluirá dentro del paquete alg<dnipropio>.p32 Si utiliza Eclipse llamar al proyecto prac03_Tromino<UOpropio>
La respuesta a las preguntas planteadas en el enunciado y las tablas con los tiempos se incluirán en un documento aparte.
En el campus virtual se entregarán dos ficheros: ZIP con el código y PDF documento de tiempos, según el calendario previsto.