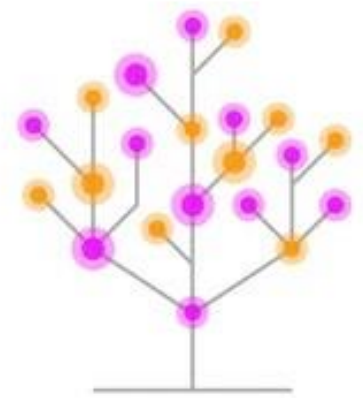

DIVIDE Y VENCERÁS

ETSII - URJC

Sebastián Ramiro Entierros García,
Adrián Martín Martín y
Álvaro Martínez Quiroga

Fundamentos del diseño del software

Índice

Contenido

Ilustraciones

Introducción y ficheros

En esta memoria se va a desarrollar la segunda práctica de la asignatura de *Fundamentos del Diseño del Software* del grado de *Ingeniería de Computadores* de la Universidad Rey Juan Carlos. La práctica consiste en el estudio de los algoritmos de divide y vencerás. El objetivo de esta práctica es desarrollar dos algoritmos sobre un problema de índices y comparar ambos junto con sus gráficas de ejecución, medidas en tiempo en milisegundos. Finalizaremos esta memoria con las conclusiones personales sobre el proyecto.

El paquete entregado, con nombre "Práctica2_FDS", incluye los siguientes ficheros y archivos:

- TrabajoFDS: carpeta en la que vienen los archivos que conforman el maven-project hecho por NetBeans.
- InstruccionesDeUso.txt: un archivo de texto en el que viene las instrucciones básicas de uso del jar.
- PruebaDyV.jar: jar que ejecuta una interfaz simple para hacer las pruebas de los algoritmos.
- Memoria_p2: memoria que resume el transcurso de la práctica.

Descripción del algoritmo

Como hemos descrito en la introducción de la presentación del proyecto, el algoritmo se trata de desarrollar un programa capaz de ejecutar un cálculo sobre el número de inversiones en los elementos de un array. Se dice que dos elementos, 'x' e 'y', están invertidos sobre un array 'E' si x < y, pero la posición E[x] > E[y]. Pongamos un ejemplo para el array [3, 4, 2, 1], el cual tendrá el siguiente grado de inversión por cada elemento del array:

- Elemento 3: grado de inversión de 2, es decir, contiene 2 números menores en posiciones mayores del array.
- Elemento 4: grado de inversión de 2, es decir, contiene 2 números menores en posiciones mayores del array.
- Elemento 2: grado de inversión de 1, es decir, contiene 1 números menores en posiciones mayores del array.
- Grado de inversión de 0, es decir, contiene 0 números menores en posiciones mayores del array.

Con estos valores obtenemos que la inversión total del array puesto como ejemplo es de:

- 2 + 2 + 1 = 5

Desarrollaremos 2 algoritmos diferentes, el primero será un algoritmo puro por fuerza bruta, es decir, no dividirá ni hará uso de técnicas de algoritmia para simplificar resultados, iterará sobre todas las posiciones del array por cada elemento de este. Mientras que el segundo de ellos hará uso de la técnica de divide y vencerás sobre el array de datos.

Generación de datos

El programa es capaz de generar datos para la ejecución del algoritmo mediante la siguiente clase:

Clase de generación

```java
import java.util.concurrent.ThreadLocalRandom;

public class Vectores {
    private int longitud;
    private int[] vector_1; // Gustos usuario 1
    private int[] vector_2; // Gustos usuario 2

    public Vectores(int longitud) {
        this.longitud = longitud;
        this.vector_1 = new int[this.longitud];
        this.vector_2 = new int[this.longitud];
        this.init();
    }

    public void imprime_vectores() {
        System.out.println();
        System.out.print("Primer vector: [");
        imprime(this.vector_1);
        System.out.print("]");
        System.out.println();
        System.out.print("Segundo vector: [");
        imprime(this.vector_2);
        System.out.print("]");
        System.out.println();
    }

    private void imprime(int[] v) {
        for(int i = 0; i < v.length; i++) {
            if(i < v.length - 1) {
                System.out.print(v[i] + ", ");
            } else {
                System.out.print(v[i]);
            }
        }
    }

    public int[] get_vector_1() {
        return this.vector_1;
    }
```

```
    public int[] get_vector_2() {
        return this.vector_2;
    }

    private void init() {
        boolean libre;
        int id;
        for(int i = 0; i < this.longitud; i++) {
            libre = false;
            // Al usuario 1 le gusta ese objeto i+1 en la posicion i:
            this.vector_1[i] = i+1;
            // Al usuario 2 le gusta ese objeto i+1 en una posicion random del
array
            while(!libre){
                // Generamos un id
                id = ThreadLocalRandom.current().nextInt(0, this.longitud);
                // Buscar una posicion libre para insertar
                if(vector_2[id] == 0) {
                    // Insertamos en esa posicion
                    vector_2[id] = i+1;
                    libre = true;
                }
            }
        }
    }

}
```

Fuerza bruta

El programa calcula la cantidad de inversiones a fuerza bruta de la siguiente manera:

Algoritmo

```
public class AlgorFuerza {
    private int contador; // numero de inversion
    private double tiempo;

    public AlgorFuerza() {
        this.contador = 0;
        this.tiempo = 0.0;
    }

    public String exec_fuerza(int[] v) {
        long t_comienzo = System.nanoTime();

        int contador = 0;
        for (int i = 0; i < v.length; i++) {
            // Comienzo operando por el valor i
            for (int j = i + 1; j < v.length; j++) {
                // Voy a iterar sobre todo el array restante
                if ((v[i] > v[j]) && (i < j)) {
                    // Si el elemento que esta seleccionado por v[j]
                    // es menor que el elemento que ha seleccionado v[i]
                    // entonces podemos afirmar que hay 1 inversion que
                    // agregar al contador
                    contador++;
                }
            }
        }
        this.contador = contador;

        long t_final = System.nanoTime();
        this.tiempo = t_final - t_comienzo;
        return this.toString();
    }

    public String toString() {
        return (
            "La inversion de la ejecucion es de: "
            + this.contador
            + ", ha tardado en ejecutar: "
            + this.tiempo + " ms."
        );
    }

}
```

Como vemos tenemos dos bucles ‘for’ anidados, esto implica a priori que la **complejidad** pertenece a **$O(N^2)$**.

La idea de esta secuencia es sencilla: iterará sobre el array de elementos, comparando la condición de validez para cada uno de ellos y, si esta es cierta, suma 1 al contador de inversiones.

Divide y vencerás

El programa aplica el algoritmo de divide y vencerás para calcular las inversiones de la siguiente manera:

Algoritmo

```
public class AlgorDyV {

	private int contador; // número de inversión
	private double tiempo;

	public AlgorDyV() {
		this.contador = 0;
		this.tiempo = 0.0;
	}

	public String exec_dyv(int[] v) {
		long t_inicio = System.nanoTime();

		this.contador = algorDyV(v);

		long t_final = System.nanoTime();

		this.tiempo = t_final - t_inicio;
		return (this.toString());
	}

	private int algorDyV(int[] array) {
		int inversiones_izquierda, inversiones_derecha, inversiones_union;

		// Lista de menor longitud a 1 no devuelve inversión
		// Ya que no tiene con que dividir y comparar
		if (array.length <= 1) {
			return 0;
		}
		// Preparamos los arrays
		int mid = array.length / 2;
		int[] izquierda = new int[mid];
		int[] derecha = new int[array.length - mid];
		// Copiamos los elementos de cada array en su lado respectivo
		for (int i = 0; i < mid; i++) {
			izquierda[i] = array[i];
		}
		for (int i = 0; i < array.length - mid; i++) {
			derecha[i] = array[mid + i];
		}

		// Contamos recursivamente las inversiones en cada parte
		inversiones_izquierda = algorDyV(izquierda);
		inversiones_derecha = algorDyV(derecha);
```

```
		int[] array_resultado = new int[array.length];
		inversiones_union = inversiones_union(izquierda, derecha, array_resultado);

		for (int i = 0; i < array.length; i++) {
			array[i] = array_resultado[i];
		}
		return (inversiones_izquierda + inversiones_derecha + inversiones_union);
	}

	private int inversiones_union(int[] izquierda, int[] derecha, int[] array_resultado) {
		int a = 0, b = 0, c = 0, suma = 0;

		// Buscamos las inversiones que hay entre los dos arrays
		while ((izquierda.length > a) && (derecha.length > b)) {
			// c sera la posicion donde insertaremos
			if (izquierda[a] <= derecha[b]) {
				// No produce inversiones, no sumamos y
				// metemos el valor en el array resultado
				array_resultado[c] = izquierda[a];
				a++;
			} else {
				// Produce inversiones, metemos en el array
				// resultado el valor y sumamos inversiones
				array_resultado[c] = derecha[b];
				suma += izquierda.length - a;
				b++;
			}
			c++;
		}

		int i;
		// Rellenamos con valores restantes usando la variable
		// c la cual en este momento se encuentra en el ultimo
		// valor + 1 que hemos introducido previamente
		if (a == izquierda.length) {
			// Si he insertado todos los valores de la izquierda
			// inserto los restantes a la derecha
			for (i = b; i < derecha.length; i++) {
				array_resultado[c] = derecha[i];
				c++;
			}
		} else {
			// Si he insertado todos los valores de la derecha
			// inserto los restantes a la izquierda
			for (i = a; i < izquierda.length; i++) {
				array_resultado[c] = izquierda[i];
				c++;
			}
		}

		// Devolvemos el resultado de las inversiones
		return suma;
```

```
        }

        @Override
        public String toString() {
                return (
                                "La inversion de la ejecucion es de: " +
                                this.contador +
                                ", ha tardado en ejecutar: " +
                                this.tiempo +
                                " ms."
                );
        }

}
```

Al ser un algoritmo de divide y vencerás, seguimos un proceso de análisis donde el problema principal se subdivide en dos subproblemas de mitad de tamaño, y el **coste de división y combinación pertenece a O(N)**. Por lo que su **T(n) es** aproximadamente **2T(n/2) + O(N)**. Esto finalmente implica que al ser a = b en aT(n/b) se puede concluir que **pertenece a O(n log n)**.

Comparación de resultados

Ilustración 1: resultados algoritmo fuerza bruta

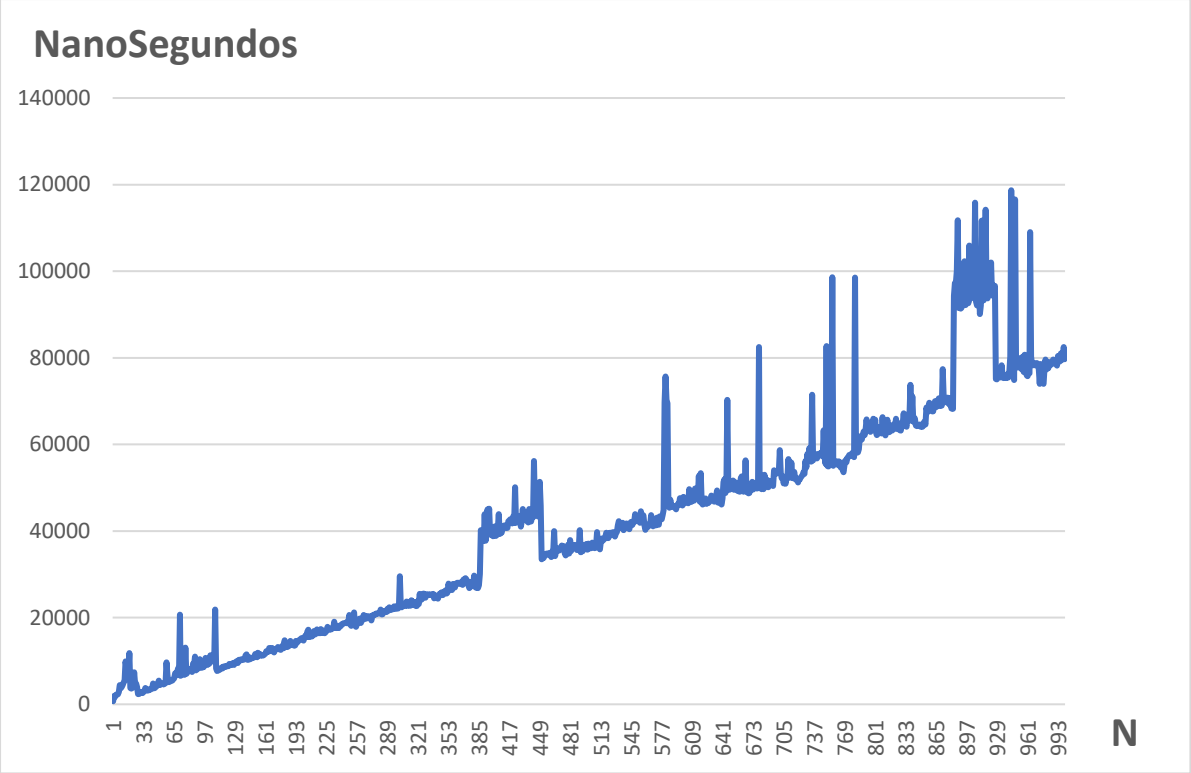

Ilustración 2: resultados algoritmo divide y vencerás

Tras visualizar estas dos gráficas y hacer pruebas con ambos algoritmos hemos llegado a la conslusión de que el algoritmo de divide y vencerás es muchísimo mas eficiente en comparación con la técnica de la fuerza bruta. Sin embargo, cabe destacar que esta mejora se da en arrays de datos extremadamente grandes, es decir, para un array de 5 a 50 elementos tarda menos o igual la técnica de fuerza bruta, ya que en comparación con el algoritmo de divide y vencerás, no tiene que copiar arrays de datos constantemente. Hemos observado tiempos, con arrays de datos de 5 elementos, igual a 900ns en la técnica de fuerza bruta, mientras que con el mismo array y la técnica de divide y vencerás, los tiempos rondaban los 1400ns a 2000ns(en el peor caso).

Conclusiones de la práctica

Consideramos esta práctica fundamental para entender como funcionan los algoritmos de divide y vencerás. Nos ha costado en un principio entender la idea del algoritmo, pero una vez desarrollada la técnica de fuerza bruta, hemos entendido este mejor y así hemos sido capaces de desarrollar el algoritmo de divide y vencerás.

También nos ha parecido muy interesante comprobar como el algoritmo de divide y vencerás mejora a raíz del crecimiento del array de datos.

Referencias

- Temario del aula virtual.