UNIVERSITAT JAUME·I

# 1º Ingeniería Informática
# 1º Matemática Computacional

## EI1008/MT1008 Programación II

Práctica 4: Algoritmos y complejidad computacional

## 1. Ejercicios sobre búsqueda

### Ejercicio 1

El fichero de texto `dniCenso.txt` contiene los DNI de las personas censadas en una determinada localidad. Por otra parte, el fichero de texto `dniClientes.txt` contiene los DNI de los clientes de una compañía de ámbito nacional. La primera línea de cada fichero contiene un valor entero que indica la cantidad de DNI que hay en ese fichero.

Nos interesa averiguar cuántas personas censadas en esa localidad son clientes de esa compañía. Sabiendo que *ninguno de los ficheros está ordenado ni contiene datos repetidos*, escribe los siguientes métodos estáticos:

- `crearVectorDni`, que reciba como parámetro el nombre de un fichero de texto con el formato descrito anteriormente y devuelva un vector de cadenas con todos los DNI que aparecen en el fichero.
- `buscarDni`, que reciba como parámetros un DNI y un vector de DNI y devuelva `true` cuando el DNI esté en el vector y `false` cuando no esté. Para ello, el método debe aplicar una búsqueda secuencial.
- `contarCoincidencias`, que reciba como parámetros dos vectores de DNI y devuelva la cantidad de DNI del primer vector que aparecen en el segundo. Para ello, debe llamar al método `buscarDni`.

En el aula virtual de la asignatura tienes disponible un método `main`[1] que crea los vectores de DNI correspondientes a los dos ficheros de texto y muestra cuántos DNI del vector del censo están en el vector de clientes. Además, se encarga de medir el tiempo que tarda en ejecutarse la llamada al método `contarCoincidencias`.

Expresa, empleando la notación $O$, el coste temporal de los métodos `buscarDni` y `contarCoincidencias`.

### Ejercicio 2

Modifica una copia de tu solución al ejercicio anterior para que:

- El método `main` utilice el fichero `dniClientesOrdenado.txt` en lugar del fichero `dniClientes.txt`. Este nuevo fichero contiene los mismos datos que el anterior pero, como su nombre indica, su contenido está ordenado lexicográficamente de menor a mayor DNI.
- El método `buscarDni` realice una búsqueda binaria, teniendo en cuenta que los datos del vector de clientes están ordenados.

Expresa, empleando la notación $O$, el coste temporal de los métodos `buscarDni` y `contarCoincidencias`.

### Ejercicio 3

Modifica una copia de tu solución al ejercicio anterior para que:

[1] Mientras desarrollas los métodos necesarios, te recomendamos utilizar ficheros y/o vectores similares a los descritos, pero con solo unos pocos datos. Una vez que hayas comprobado que todo funciona correctamente, utiliza los ficheros de datos que tienes disponibles en el aula virtual. Si todo es correcto, tu programa debe indicar que 434 personas censadas en esa localidad son clientes de esa compañía.

- El método `main` utilice el fichero `dniCensoOrdenado.txt` en lugar del fichero `dniCenso.txt`. Este nuevo fichero contiene los mismos datos que el anterior pero, como su nombre indica, su contenido está ordenado lexicográficamente de menor a mayor DNI.
- Se elimine el método `buscarDni`.
- El método `contarCoincidencias` tenga en cuenta que ambos vectores están ordenados y, por lo tanto, calcule las coincidencias mediante una variante del algoritmo de mezcla.

Expresa, empleando la notación $O$, el coste temporal del nuevo método `contarCoincidencias`.

## 2. Ejercicio de examen

El siguiente ejercicio apareció en el examen de junio de 2017. En los ficheros auxiliares de la práctica que tienes disponibles en *Aula Virtual* puedes encontrar las clases auxiliares necesarias y un pequeño programa de prueba.

Te aconsejamos que resuelvas también otros ejercicios de examen y prepares tú mismo las clases auxiliares y las pruebas necesarias de manera similar a como hemos hecho para este ejercicio.

### Ejercicio 4

Necesitamos desarrollar una aplicación para gestionar las actas de calificación de asignaturas de una titulación. Ya hemos definido una clase, `NotaEstudiante`, que proporciona los siguientes constructores y métodos públicos:

- `NotaEstudiante(String DNI, int convocatoria)`: constructor de la clase. Almacena el DNI y el número de convocatoria de un estudiante e inicialmente lo marca como no presentado.
- `String getDNI()`: devuelve el DNI del estudiante.
- `int getConvocatoria()`: devuelve el número de convocatoria del estudiante.
- `double getNota()`: devuelve la nota del estudiante.
- `void setNota(double nota)`: establece la nota del estudiante y lo marca como presentado.
- `boolean getPresentado()`: devuelve `true` si se ha establecido una nota para el estudiante y `false` en caso contrario.

Necesitamos definir una nueva clase, `Acta`, que gestione el acta de una asignatura. Considera que ya tenemos la siguiente definición parcial:

```
public class Acta {

    // Atributos
    private String código;          // Código de la asignatura
    private int curso;              // Curso académico
    private NotaEstudiante[] notas; // Datos de los estudiantes que hay en el acta
                                    // (ordenados de menor a mayor por DNI)
```

```
    // Constructor
    public Acta(String código, int curso, NotaEstudiante[] notas) {
        this.código = código;
        this.curso = curso;
        this.notas = notas;
    }

    // Métodos
    ...
}
```

Añade a la clase `Acta` los siguientes métodos públicos:

a) `void calificar(String DNI, double nota)`

Cuando el DNI dado aparezca en el acta, se establecerá la nota indicada para ese estudiante. En caso contrario, se lanzará la excepción `NoSuchElementException`.

El coste temporal de este método debe ser `O(log n)`, siendo `n` el número de estudiantes en el acta.

b) `void enviarSMS(String[] vectorDNI)`

El parámetro `vectorDNI` contiene, ordenados lexicográficamente de menor a mayor, los DNI de todos los estudiantes de la titulación que están suscritos al servicio de comunicación de calificaciones a través de SMS. Para todo estudiante que figure en el acta como presentado y que esté suscrito a este servicio de mensajería, se le debe enviar un mensaje con la calificación obtenida. Para el envío del mensaje, habrá que usar el método estático enviar de la clase `SMS`:

`static void enviar(String DNI, double nota)` // Envía la nota dada al estudiante indicado

Dado que el atributo `notas` y el parámetro `vectorDNI` están ordenados por `DNI`, el coste temporal de este método debe ser `O(n)`, siendo `n` la suma de elementos de los dos vectores.

c) `Acta siguienteConvocatoria()`

El método debe crear y devolver una nueva `Acta`, con el mismo código de asignatura y curso académico, pero en la que no aparezcan los estudiantes que ya han aprobado (es decir, que solo incluya los estudiantes no presentados o suspensos). El número de convocatoria se debe mantener para los no presentados y se debe incrementar en uno para los suspensos. En cualquier caso, el vector del acta resultado debe contener nuevos objetos de la clase `NotaEstudiante` inicialmente no calificados.

Si el coste temporal de este método es mayor que `O(n)`, siendo `n` el número de estudiantes en el acta, se penaliza la puntuación de este apartado.