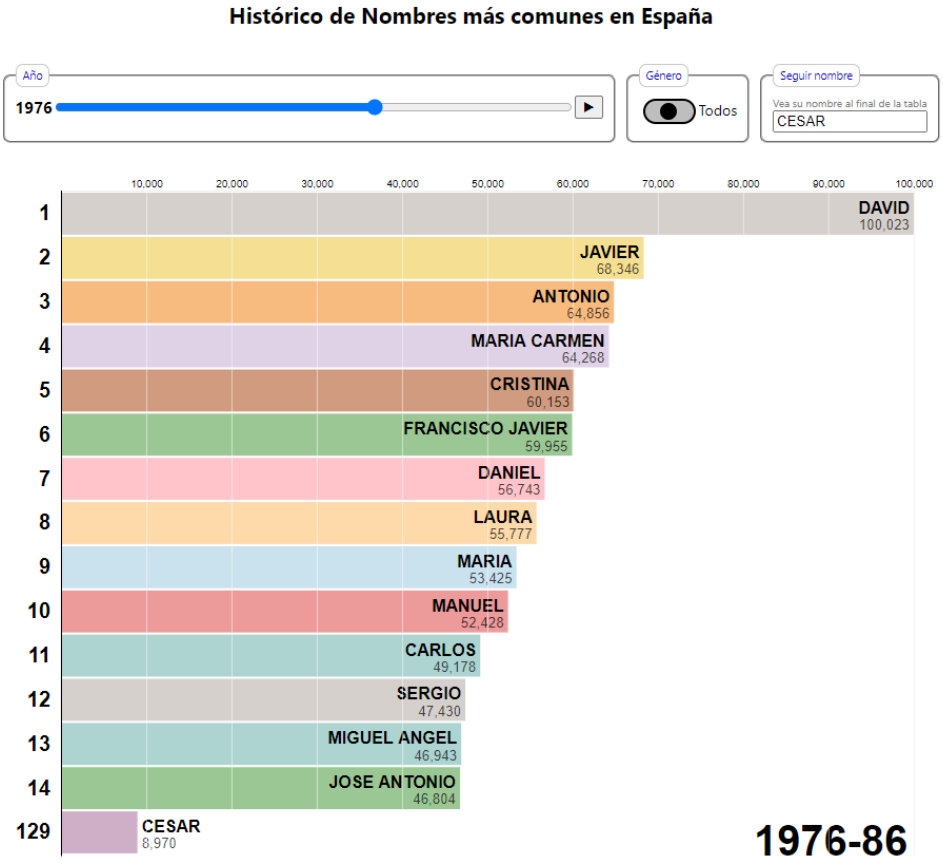
Estructuras de Datos y Algoritmos

Práctica I - Curso 2020/21

Nombres más comunes

1. Introducción

En esta práctica vamos a suponer que queremos analizar si es factible el crear una aplicación que muestre los nombres más comunes en un determinado intervalo de fechas de forma que pueda ser interactiva y susceptible de animación, es decir si a partir de los datos disponibles podemos calcular y mostrar el resultado en centésimas de segundo.



Por supuesto el objetivo final de la práctica no es construir la aplicación, sino escribir un código de prueba con la parte de tratamiento de los datos, analizar su escalabilidad y los factores de los que depende, y detectar aquellas partes del tratamiento de datos que tienen un impacto mayor en el rendimiento. Con esa información, en la segunda práctica se intentarán usar las estructuras de datos adecuadas para mejorar el rendimiento.

2. Descripción del Problema

Los datos de que vamos a disponer son un array con un elemento para cada **persona** nacida en la zona tratada por la aplicación (en nuestro caso dispondremos de dos ficheros, uno para Valladolid y otro para toda España) desde 1920 hasta 2020. Estos datos se han obtenido del Instituto Nacional de Estadística (www.ine.es) y constan de 448.560 registros para Valladolid y 42.195.534 en el caso de España[1].

Los datos disponibles para cada persona son:

- Nombre de pila (en mayúsculas, sin acentos)
- Género (0 -> Masculino, 1 -> Femenino)
- Fecha de nacimiento[2]
- Fecha de defunción[3]

Las fechas de nacimiento y defunción se expresan como números enteros que indican el número de días transcurridos desde el 1 de enero de 1920 (en el anexo se adjunta un trozo de código para poder traducir una fecha expresada como texto a ese número entero).

Este array se leerá de un fichero de texto que tiene el siguiente formato:

- La primera línea contiene el número de registros (el número de personas)
- A continuación aparecen 2 líneas para cada persona, la primera con los tres enteros que indican la fecha de nacimiento, defunción y el género, y la siguiente línea indica el nombre de pila. Las fechas ocupan siempre 5 caracteres (con ceros a la izquierda si es necesario).
- Los registros no están ordenados

```
448560
03791 04091 0
FRANCISCO
28015 57057 0
ANGEL
08241 40575 1
ESPERANZA
…
```

Aunque este formato parezca un poco raro se ha diseñado para optimizar en lo posible la lectura de los ficheros, ya que el fichero de España ocupa casi 1Gb.

Este array es el que **va a servir como base** para todos los cálculos posteriores. **Importante**: No se debe ordenar ni reprocesar el array en esta primera práctica.

Con esos datos el objetivo va a ser encontrar los ***k*** nombres más comunes de las personas nacidas entre un determinado **intervalo de fechas**. Además se deberá calcular el **número de personas** con cada uno de esos nombres y existirá un **nombre del usuario** para el que se calculará además de su número de personas su **posición en el ranking** de popularidad.

El proceso consistirá en generar un nuevo array no de personas, sino de pares [nombre, número de personas que lo tienen asignado], siempre teniendo en cuenta el intervalo de fechas establecido. Este array de nombres se ordenará (por número de personas) para obtener los ***k*** nombres más populares y para poder localizar entre ellos la posición del nombre del usuario.

[1] Estos valores son más pequeños que la población real porque el INE no incluye referencias a nombres asignados a menos de 20 personas ni datos de personas no nacidas en España.

[2] Como en los datos del INE solo aparece la **decada**, la fecha de nacimiento se ha generado al azar entre todas las fechas de esa década.

[3] La fecha de fallecimiento no aparece en el INE. Aquí también (vea el punto anterior) este valor se ha generado al azar teniendo en cuenta la distribución de mortalidad estándar. Por favor, no sea morboso buscándose a si mismo en la tabla para averiguar su fecha de fallecimiento! No son datos reales!!

Parámetros de entrada

Los parámetros necesarios en cada ejecución del proceso son los siguientes:

1. **Intervalo de fechas**: Dos valores (fecha inicial y final) indicando el intervalo de fechas de nacimiento para las que se va a realizar el cálculo. De todas las personas solo se realizará el cálculo para aquellas con una fecha de nacimiento superior o igual a la fecha inicial e inferior o igual a la fecha final.
2. **Número de nombres a mostrar (*k*)**
3. **Nombre del usuario**

La aplicación (en modo consola) pedirá en primer lugar el nombre del fichero de texto con los datos y creará el array principal. A continuación pedirá el número de nombres a mostrar y el nombre del usuario (estos parámetros ya permanecen fijos durante el resto de la ejecución). Por último entrará en un bucle en el que pedirá el intervalo de fechas, realizará el cálculo, mostrará los resultados y preguntará al usuario si desea otra iteración. A continuación se muestra un ejemplo del aspecto que podría tener la ejecución del programa (en azul los datos que introduce el usuario):

```
Nombre del fichero: personas_va.txt
_|____|____|____|____|
********************
Creación array: 0,01573 seg. Lectura fichero: 0,15297 seg.
n = 448560 personas en total.

*** OPCIONES GENERALES ***
Número de nombres a mostrar: 5
Nombre del usuario: CESAR

*** BUCLE DE CONSULTAS ***
Intervalo fechas de nacimiento: 1/1/1974 31/12/1982

    1. DAVID 1.523
    2. JAVIER 943
    3. ALBERTO 827
    4. OSCAR 822
    5. ROBERTO 802
   56. CESAR 303

*** RESULTADOS ***
m = 59338 personas cumplen las condiciones.
p = 1039 nombres distintos.
d = 3287 dias en el intervalo.
FILTRADO     |       448560 comps |            0 movs | 0,00979 seg
INSERCIÓN #1 |      9619612 comps |         3071 movs | 0,05781 seg
INSERCIÓN #2 |       524558 comps |       269569 movs | 0,02617 seg
EXTRACCIÓN   |        26278 comps |        11760 movs | 0,00113 seg
SEGUIMIENTO  |           56 comps |            0 movs | 0,00002 seg

¿Continuar? [S/N] S

*** BUCLE DE CONSULTAS ***
Intervalo fechas de nacimiento: _
```

Resultados

En la aplicación real los resultados a mostrar serían únicamente la tabla de nombres. Sin embargo nosotros queremos analizar la eficiencia de nuestro algoritmo y para ello vamos a contar el **número de comparaciones y movimientos** (solo en aquellas operaciones en las que intervenga un elemento de un array) en cada **fase** del proceso. Vamos a identificar 4 fases:

1. **FILTRADO**: El recorrido por el array principal detectando las personas cuya fecha de nacimiento está en el intervalo. Podemos considerar que cada comprobación es una única comparación y por lo tanto en esta fase siempre el número de comparaciones es igual que el de registros (personas) del array principal.
2. **INSERCIÓN**: Durante el recorrido del array en el que se realiza el filtrado tenemos que generar la estructura de datos que almacena el conjunto de pares nombre-número de personas (descrito en el último párrafo de la página 2). Esta estructura debe ser un **array parcialmente lleno** (**APL** de aquí en adelante) que contenga objetos con dos campos/propiedades, el nombre y el número de personas que lo tienen asignado. El algoritmo es muy sencillo: Si una persona está en el intervalo de fechas correcto, se **busca** su nombre en el APL. Si existe, se incrementa el campo número. Si no existe, se **inserta** en el APL (con el campo número igual a uno, claro).
3. **EXTRACCIÓN**: De todos los nombres almacenados en el APL hay que obtener los ***k*** con mayor número. El algoritmo será simplemente **ordenar** el APL por el campo número. Se puede usar cualquier algoritmo de los contemplados en la asignatura, aunque parece evidente que deberíamos escoger uno avanzado.
4. **SEGUIMIENTO**: Esta fase consiste en buscar el nombre del usuario en el APL para averiguar su número y su posición en el ranking.

El objetivo va a ser identificar aquellas fases donde se realicen más operaciones para poder dedicar (en la segunda práctica) la mayor cantidad de esfuerzo en optimizar los algoritmos o estructuras utilizados. Para la fase de **INSERCIÓN** existen **dos alternativas** que vamos a considerar (se pueden implementar las dos en un único programa o bien crear dos programas con cada una de las alternativas):

1. En la primera alternativa el APL está desordenado, la **búsqueda** se realiza de forma **secuencial** y la **inserción** (si no existe el nombre) es siempre **al final**.
2. El la segunda alternativa el APL está ordenado por nombre. La **búsqueda** es ahora **binaria** pero la **inserción** debe realizarse en el **sitio adecuado** para que el APL siga ordenado (desplazando primero los elementos).

Uno de los objetivos de esta primera práctica es detectar cual de estas dos alternativas es mejor y bajo que circunstancias.

Posibles tamaños de las entradas

Para que no sea todo tan sencillo, en este problema el número de operaciones que realiza cada una de las fases puede depender de uno o varios de los siguientes valores (típicamente solo uno de ellos es el relevante):

- ***n***: el número total de personas del array principal
- ***m***: el número de personas cuya fecha de nacimiento está en el intervalo
- ***p***: el número de nombres distintos de las *m* personas tratadas (tamaño del APL)
- ***d***: el número de días del intervalo de fechas

Nota: El parámetro *k* (número de nombres a mostrar) no es relevante en esta primera práctica pero si que puede serlo en la segunda.

Se deberá encontrar para cada fase el parámetro relevante y su dependencia respecto a él en notación asintótica (término más importante exacto). Esta información se solicitará (salvo indicación en contra) en la defensa de la práctica.

Como el único parámetro que podemos controlar es ***d***, el número de días del intervalo de fechas, se recomienda que se ejecute el bucle del programa con varios intervalos obteniendo medidas que luego se puedan representar gráficamente (por ejemplo en Excel) y estimar el mejor ajuste respecto a los posibles parámetros de los que puedan depender.

3. Restricciones al programar la aplicación

En esta primera práctica no estamos interesados en obtener una aplicación eficiente, por lo tanto, se restringe el número de técnicas y elementos de programación que se pueden utilizar de la forma siguiente:

- Se debe seguir estrictamente el esquema de aplicación y fases indicado.
- Para representar el array principal y el APL se deben utilizar **únicamente arrays**. **No se puede usar ArrayList** (Java). En el caso de Python solo se pueden usar las listas estándar (está prohibido usar diccionarios, conjuntos, arrays de NumPy, etc.)
- Las operaciones de **búsqueda**, **inserción** y **ordenación** deben ser programadas completamente por el alumno (salvo la inserción si se usa Python)
- Si tiene dudas sobre que elementos del lenguaje o entorno puede utilizar consulte primero al profesor

4. Presentación y Evaluación de la práctica

La práctica puede ser codificada en Java o Python, aunque se recomienda el uso de Java ya que esta práctica puede ser muy exigente respecto a los recursos (tiempo y espacio de almacenamiento).

Para una correcta evaluación de la práctica el alumno deberá:

1. Presentar electrónicamente (por el Aula Virtual de la Escuela o por correo electrónico), antes de la fecha límite (que se establecerá más adelante), un fichero comprimido que contenga el código fuente de la aplicación descrita en el enunciado. En el código fuente debe aparecer (como comentario) en las primeras líneas el nombre de quienes han realizado la práctica.
2. Presentarse a la sesión de evaluación que le corresponda según su grupo de laboratorio en la semana establecida. En esta sesión se probará la aplicación, se pedirá una modificación sencilla, y se indicarán los datos de su análisis de eficiencia en un cuestionario.

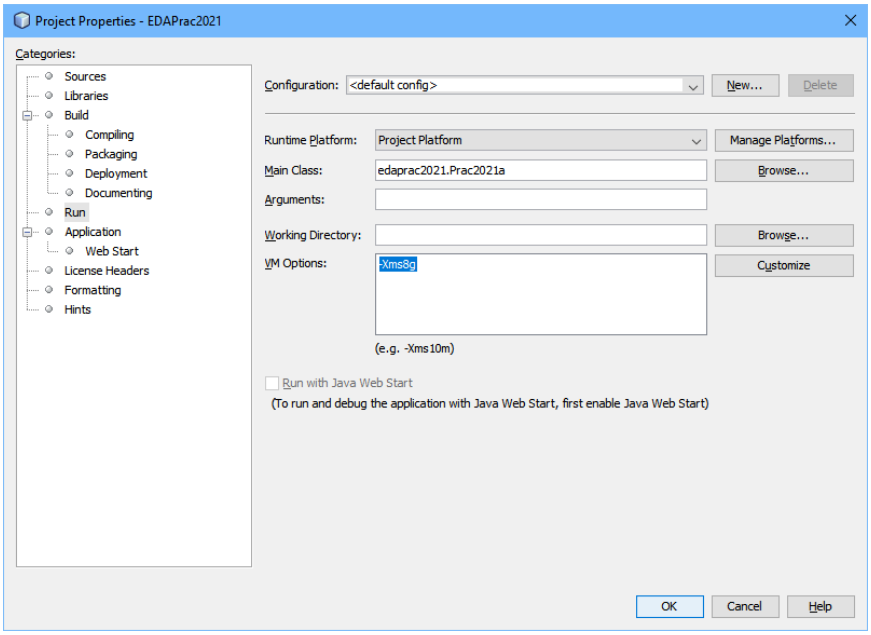
Es en la defensa de la práctica donde se produce la evaluación, la presentación electrónica es simplemente un requisito previo para garantizar la equidad entre subgrupos y la comprobación preliminar de autoría.

5. ANEXO: Recomendaciones

- El fichero con los datos de España es muy grande y es posible que no pueda ejecutarse en su equipo si tiene limitaciones de memoria. En cualquier caso, es conveniente ampliar al máximo la memoria disponible usando el parámetro -XMS de la máquina virtual. Si lo ejecuta en consola escriba:

  `Java -Xms8g` *`miprograma`*

  Si usa Netbeans, haga click con el botón derecho del ratón en su proyecto, y pulse "Propiedades", escribiendo la opción en "VM Options":



- Si de todas formas ve que no puede trabajar con ese fichero utilice el de los datos de Valladolid.
- En Java, la comprobación de igualdad entre strings debe hacerse siempre mediante el método **equals** (*cad1.equals(cad2)*), nunca utilice el operador de igualdad (==). En Python, sin embargo, es seguro usar el operador.
- Para comprobar si un string es menor o mayor que otro utilice el método **compareTo**.
- No es necesario calcular el tiempo en segundos ni se debe utilizar para los análisis.
- Para contar el número de operaciones use dos variables, una para las comparaciones y otra para los movimientos, definidas en la clase que representa al APL o bien definidas como variables estáticas en cualquier otro sitio, e increméntelas cada vez que se vaya a realizar una operación del tipo adecuado. No se olvide ponerlas a cero cuando cambie de fase.

6. ANEXO: Código

La siguiente función está optimizada para leer el fichero de texto con los datos en un tiempo razonable (entre 10 y 30 segundos el fichero de España):

```
public static Persona[] leeFichero(String nomfich) throws IOException {
    BufferedReader br = new BufferedReader(
                        new FileReader(new File(nomfich)), 131072);
    // La primera linea contiene el número de entradas
    int n = Integer.parseInt(br.readLine());
    System.out.println("_|____|____|____|____|");
    // Optimización: Creación previa de array y de objetos
    long t0, t1, t2;
    t0 = System.nanoTime();
    Persona[] dat = new Persona[n];
    for(int i = 0; i < n; i++) { dat[i] = new Persona(); }
    t1 = System.nanoTime()-t0;
    System.out.print("*");
    int lim = n/20;
    t0 = System.nanoTime();
    for(int i = 0; i < n; i++) {
        dat[i].parse(br.readLine(), br.readLine());
        if(i % lim == lim-1) { System.out.print("*"); }
    }
    t2 = System.nanoTime() - t0;
    System.out.printf("\nCreación array: %.5f seg.
                       Lectura fichero: %.5f seg.\n", 1e-9*t1, 1e-9*t2);
    System.out.printf("n = %d personas en total.\n\n", dat.length);
    return dat;
}
```

```
public class Persona {
    // Las fechas son días transcurridos desde el 1/1/1920
    int nac;    // Fecha de nacimiento
    int fac;    // Fecha de defunción
    int gen;    // Género (0 -> hombre, 1 -> mujer)
    String nom; // Nombre de pila

    // Extracción de datos de 2 lineas del fichero
    // Optimizado para procesamiento más rápido
    public void parse(String lin1, String lin2) {
        nac = (lin1.charAt(0)-48)*10000 + (lin1.charAt(1)-48)*1000 +
              (lin1.charAt(2)-48)*100 + (lin1.charAt(3)-48)*10 +
              (lin1.charAt(4)-48);
        fac = (lin1.charAt(6)-48)*10000 + (lin1.charAt(7)-48)*1000 +
              (lin1.charAt(8)-48)*100 + (lin1.charAt(9)-48)*10+
              (lin1.charAt(10)-48);
        gen = lin1.charAt(12)-48;
        nom = lin2;
    }
}
```

La traducción de fecha a número se puede hacer con la siguiente función:

```
public static int traduceFecha(String fec) {
    String[] trozos = fec.split("/");
    int dia = Integer.parseInt(trozos[0]);
    int mes = Integer.parseInt(trozos[1]);
    int año = Integer.parseInt(trozos[2]);
    return 367*año - (7*(año+5001+(mes-9)/7))/4 +
           (275*mes)/9 + dia - 692561;
}
```

Para copiar texto al portapapeles (útil para traspasar rápidamente datos a Excel) puede usar la siguiente función (si usa Linux dependiendo de la versión puede no funcionar):

```
import java.awt.Toolkit;
import java.awt.datatransfer.*;

public void Portapapeles(String txt) {
    StringSelection selection = new StringSelection(txt);
    Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    clipboard.setContents(selection, selection);
}
```

Para que los datos aparezcan en celdas contiguas sepáreles con el carácter tabulador (\t).