Práctica 3. Análisis experimental de algoritmos de búsqueda.

Índice

1. Objetivo.

El objetivo principal de la práctica es estudiar empíricamente Algoritmos de Búsqueda. Los análisis teóricos están incluidos en los contenidos de la asignatura.

El objetivo específico de la práctica es:

- Estudiar experimentalmente el comportamiento temporal de un algoritmo de búsqueda comparándolo, además, con el de otros procedimientos de búsqueda. Estudiaremos y compararemos los algoritmos de búsqueda **`Secuencial`**, **`Binaria`** y **`Ternaria`**. Al finalizarlo, se dispondrá de los tiempos utilizados para ordenar un vector usando varios algoritmos, de modo que tendremos criterios objetivos para poder elegir la alternativa más adecuada.

2. Descripción del problema

Este estudio de costes se va a realizar sobre *arrays* de enteros y claves generados aleatoriamente.

2.1. Definición de la clase base `ConjuntoInt`.

La definición de la clase **ConjuntoInt**, para que se pueda especificar (y pueda variar) el tamaño máximo del vector, es la proporcionada en la Práctica 2.

2.2. La clase AlgoritmosBusqueda.

La clase **`AlgoritmosBusqueda`** define los Algoritmos de Búsqueda para buscar una clave **`key`** en un vector de enteros, ***`v`***, de un cierto tamaño **`size`**. Define la interfaz de los siguientes métodos de Búsqueda por clave en vectores:

- Secuencial.
- Binaria.
- Ternaria.

```
class AlgoritmosBusqueda
{
public:
    AlgoritmosBusqueda();
    ~AlgoritmosBusqueda();

/*
 * Función busquedaSecuencialIt, implementa el método de búsqueda secuencial Iterativo
 * param v: el array de enteros donde buscar
 * param size: tamaño del array
 * param key: clave o elemento a buscar
 * return posición de la clave en el array
 */
 int busquedaSecuencialIt(int v[], int size,int key);

/*
 * Función busquedaBinariaRc, implementa el método de búsqueda binaria Recursivo
 * param v: el array de enteros donde buscar
 * param size: tamaño del array
 * param key: clave o elemento a buscar
 * return posición de la clave en el array
 */
  int busquedaBinariaRc(int v[], int size,int key);
  int BinariaRc(int A[], int left, int right, int key);
```

```
/*
 * Función busquedaTernariaRc, implementa el método de búsqueda ternaria recursiva
 * param v: el array de enteros donde buscar
 * param size: tamaño del array
 * param key: clave o elemento a buscar
 * return posición de la clave en el array
 */
 int busquedaTernariaRc(int v[], int size,int key);
 int TernariaRc(int A[], int left, int right, int key);
};
```

NOTA IMPORTANTE-> La implementación de los métodos hay que realizarlos siguiendo el seudocódigo dado a continuación:

```
funcion BusquedaSecuencial(V[1..n];size,key:int)
   i←1;
   mientras (v[i]!= key && i<=size)
   {
       i←i+1;
   }
   fmientras
   si(v[i]==key)
       devolver i;   // posición donde se encuentra el elemento en el array
   sino
       devolver -1; // no se encuentra el elemento en el array
   fsi
ffuncion
```

```
funcion BusquedaBinaria(V[1..n];primero,ultimo,clave:int)
    si primero ≥ ultimo  entonces
       devolver V[ultimo]=clave;
    fsi
    mitad ← ((ultimo - primero + 1) / 2);
    si clave = V[mitad] entonces
       devolver cierto;
    sino
       si clave < V[mitad] entonces
              devolver BusquedaBinaria (V, primero, mitad-1, clave);
       sino
              si clave > V[mitad] entonces
                 devolver BusquedaBinaria (V, mitad+1, ultimo, clave);
              fsi
       fsi
    fsi
ffuncion
```

```
funcion BusquedaTernaria(V[1..n];primero,ultimo,clave:int)
    si primero ≥ ultimo  entonces
       devolver V[ultimo]=clave;
    fsi
    tercio ← ((ultimo - primero + 1) / 3);
    si clave = V[primero+tercio] entonces
       devolver cierto;
    sino
       si clave < V[primero+tercio] entonces
              devolver BusquedaTernaria (V, primero, primero+tercio-1, clave);
       sino
              si clave = V[ultimo-tercio] entonces
                     devolver cierto;
              sino
                     si clave < V[ultimo-tercio] entonces
                        devolver BusquedaTernaria (V, primero+tercio+1, ultimo-tercio-1,
                     clave);
                     sino
                            devolver BusquedaTernaria (V, ultimo-tercio+1, ultimo, clave);
                     fsi
              fsi
       fsi
    fsi
ffuncion
```

2.3. La clase Graficas.

La clase **`Graficas`** contiene métodos para guardar las gráficas de los resultados, es decir, crea los ficheros por lo lotes(comandos) para generar los ficheros gráficos correspondientes a:

- Caso medio de un algoritmo de ordenación-búsqueda con ajuste a su Orden de complejidad.
- Casos medios de dos algoritmos de Ordenación-Búsqueda.

2.4. Definiciones de constantes, fichero "`Constantes.h`".

Contiene las definiciones de las constantes utilizadas en la aplicación.

3. Actividades a desarrollar.

Para el desarrollo de esta práctica habrá que realizar las siguientes actividades:

3.1. Programa Principal.

Se deberá modificar el método ***main*** para incluir un nuevo menú para la búsqueda tal y como se muestra al final del enunciado e invocar las 3 opciones del menú para verificar que los métodos de búsqueda implementados funcionan correctamente, calcular sus costes en el caso medio y comparar o estudiar cual de los métodos es el más eficiente.

3.2. Comprobación de los métodos de búsqueda

En los ficheros suministrados se encuentra también la clase **TestBusqueda**. En esta clase está implementado un método, **comprobarMetodosBusqueda()**, que se encargará de comprobar si los algoritmos de búsqueda funcionan correctamente. Tenéis implementar los correspondientes métodos de búsqueda, ver cómo funcionan y comprobarlos.
La clase **TestBusqueda** dispone de un método que es utilizado para esta tarea:

- **`buscaEnArrayDeInt`**: busca un elemento solicitado por pantalla en un *array* de enteros para una talla determinada utilizando el método de búsqueda indicado, y devuelve la posición del elemento en el array mediante el parámetro posición y el tiempo empleado en el proceso (el tiempo no se necesita para esta tarea).

3.3. Cálculo del caso medio para los métodos de búsqueda

La clase `TestBusqueda`, también calcula el coste para el caso medio de los métodos de búsqueda y muestra el resultado por pantalla del método elegido de forma tabulada, permite guardar los mismos en un fichero de datos y realizar la gráfica de los resultados y guardarla en un fichero junto con el ajuste de la correspondiente función de orden O prevista de forma teórica. Tenéis que implementar esta tarea en el método **`casoMedio`** el cual recibe el nº del método de ordenación a estudiar como parámetro. La implementación debe seguir los siguientes pasos:

1. Almacenar y mostrar por pantalla el tiempo de ejecución en promedio de dicho método para vectores de enteros generados aleatoriamente, y con tallas sucesivamente crecientes: 1000, 2000, …, 10000.,
2. La clase `TestBusqueda` dispone de un método que es de gran utilidad para esta tarea:

- **`buscaEnArrayDeInt`**: busca un elemento generado aleatoriamente utilizando el método generaKey() de la clase ConjuntoInt en un *array* de enteros para una talla determinada utilizando el método de búsqueda indicado, y devuelve el tiempo empleado en el proceso, para esta tarea no nos hace falta la posición. Para cada una de estas tallas se ejecutará el algoritmo de búsqueda un cierto número de veces definido por una constante, NUMREPETICIONES=100 (por ej.). De esta forma se podrá obtener el coste medio como la media aritmética de las medidas tomadas. La salida del programa deberá ser tabulada, con un formato legible, similar al que se muestra en el ejecutable.

3. Se mostrará el nombre del fichero en el que se han grabado los datos, que tiene que ser el mismo nombre que el algoritmo al que corresponda.
4. Finalmente, mostrará una opción para dibujar y grabar la gráfica resultante ajustada a la función correspondiente según coste teórico e invocando al método **`generarGraficaMEDIO`** de la clase Graficas grabará los datos en el fichero de comandos correspondiente (CmdMedio.gpl) y ejecutará el script con una llamada al sistema tras lo cual se guarda el resultado gráfico en un fichero pdf con el nombre el método que se les pasó por parámetro.

3.4. Comparación de los métodos de búsqueda

La clase **TestBusqueda**, también se encarga de comparar el coste de dos de los métodos de búsqueda que recibe como parámetros, mostrando el resultado por pantalla de forma tabulada así como de grabar los datos y generar el fichero de comandos para gnuplot que grabará la gráfica en un fichero pdf. Esta tarea la realiza el método **comparar** de la clase **TestBusqueda**, que tenéis que implementar, el cual:

1. Almacena y muestra por pantalla el tiempo de ejecución en promedio de dichos métodos para *vectores* de enteros generados aleatoriamente, y con tallas sucesivamente crecientes: 1000, 2000, …, 10000.
2. El método buscaEnArrayDeInt también se utiliza para esta tarea: busca un elemento generado aleatoriamente utilizando el método generaKey() de la clase ConjuntoInt en un *array* de enteros para una talla determinada utilizando el método de búsqueda indicado, y devuelve el tiempo empleado en el proceso, para esta tarea no nos hace falta la posición. Para cada una de estas tallas se ejecutará el algoritmo de búsqueda un cierto número de

veces y que está definido en la constante NUMREPETICIONES. De esta forma se podrá obtener el coste medio como la media aritmética de las medidas tomadas. La salida del programa deberá ser tabulada, con un formato legible, similar al que se muestra en el ejecutable.

3. A continuación mostrará la opción de grabar la gráfica resultante en cuyo caso, utilizando el método **`generarGraficaCMP`** de la clase `Graficas` grabará los datos en el fichero de comandos correspondiente (CmdComparar.gpl) y ejecutará el script con una llamada al sistema tras lo cual se guarda el resultado gráfico en un fichero pdf con el nombre asociado al número del método1+método2 que se les pasó por parámetro.

4. Entrega de la práctica.

La fecha de **entrega** es el domingo **26 de mayo**.

La entrega de la práctica se realizará por Moodle en la tarea puesta al efecto y tendrá el siguiente formato:

Crear y subir la carpeta ***Apellido1Apellido2.rar*** la cual debe contener:

- Una subcarpeta con los fuentes del programa (.cpp y .h)
- Una subcarpeta con el ejecutable y los datos.
- La memoria, de **10 páginas como máximo**, en formato pdf. con el esquema especificado a continuación.

4.1. Esquema para la memoria.

Índice

1. Portada (todos los autores)
2. Índice
3. Introducción. Algoritmos de búsqueda.
4. Cálculo de los tiempos teóricos:
   4.1. Pseudócódigos y análisis de coste
   4.2. Tablas(ficheros) y Gráficas de coste
   4.3. Conclusiones
5. Cálculo del tiempo experimental:
   5.1. Tablas(ficheros) y Gráficas de coste
   5.2. Conclusiones
6. Comparación de los resultados teórico y experimental.
7. Diseño de la aplicación.

(Mostrar un esquema gráfico global de la estructura de tipos de datos existentes. Detallar la descomposición modular del programa, qué módulos existen, cuál es la responsabilidad de cada uno y la relación de uso. Documentar cualquier otra decisión de diseño que pueda resultar de interés. Funcionamiento y explicación de los métodos implementados en la práctica.)

8. Conclusiones y valoraciones personales de la práctica.

4.2. Consideraciones para la memoria.

En las gráficas de coste habrá que tener en cuenta los siguientes puntos:

- Representación gráfica de los tiempos de ejecución obtenidos en la implementación del algoritmo y comparación con el coste teórico esperado analizado.
- Las gráficas deben tener el formato adecuado: leyenda de la gráfica, nombre de los ejes, etc.
- En el caso del tiempo promedio para un método la gráfica de tiempos debe ir acompañada de la curva teórica la cual, se obtendrá del análisis teórico realizado y con la ayuda de la función 'fit' de gnuplot
- Las gráficas deben ir explicadas: qué se observa en la gráfica, si coincide con lo esperado en teoría y de no ser así, por qué, es decir, un contraste teórico/experimental.

5. Resumen de los pasos a seguir para realizar la práctica 3.

En la carpeta FicherosPractica3.rar tenéis las siguientes clases que tendréis que incorporar a vuestra la práctica_2 y crear el proyecto ***ApellidoPractica3***:

- la clase **`AlgoritmosBusqueda`**, ficheros **`AlgoritmosBusqueda.h`** y **`AlgoritmosBusqueda.cpp`**
- la clase **`TestBusqueda`**, ficheros **`TestBusqueda.h`** y **`TestBusqueda.cpp`**
- fichero "**`Constantes.h`**"

1. Tomar los fuentes de la práctica_2.
2. Modificar el programa principal de la práctica_2 incorporando los menús como podéis ver a continuación.
3. Completar los métodos `busquedaSecuencialIt`, `busquedaBinariaRc` y `busquedaTernariaRc` de la clase `AlgoritmosBusqueda` el cual lo tenéis que adaptar de los seudocódigos especificados en esta memoria.
4. Completar los métodos **`casoMedio`** y **`comparar`** de la clase **TestBusqueda.**
5. Modificar el método **`generarGraficaMEDIO`** de la clase **Graficas** para los nuevos órdenes de complejidad.

6. Ejemplos de ejecución

```
*** FAA. Practica 3. Curso 17/18 ***
                                        Prof. Teresa Santos

                *** MENU PRINCIPAL ***

                1.- MENU ORDENACION
                2.- MENU BUSQUEDA
                0.- Salir
                ---------
                Elige opcion: 2
```

```
                *** MENU BUSQUEDA ***
                1.- Probar los metodos de busqueda
                2.- Obtener el caso medio de un metodo de busqueda
                3.- Comparar dos metodos
                0.- Volver al menu principal
                ----------
                Elige opcion: 2
```

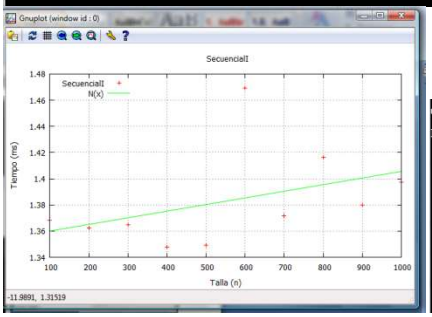
```
        *** Metodo a estudiar para el caso medio ***
        0: Busqueda Secuencial Iterativa
        1: Busqueda Binaria Recursiva
        2: Busqueda Ternaria Recursiva
        ---------
        Elige opcion: 0
```

```
Busqueda SecuencialI. Tiempos de ejecucion promedio

        Talla           Tiempo (mseg)

        100                     1.4
        200                     1.4
        300                     1.4
        400                     1.3
        500                     1.3
        600                     1.5
        700                     1.4
        800                     1.4
        900                     1.4
        1000                    1.4

Datos guardados en el fichero SecuencialI.dat
Generar grafica de resultados? (s/n): s
```



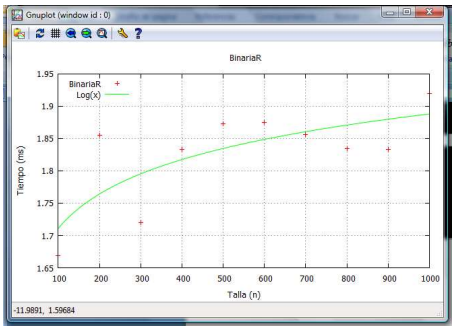

```
Grafica guardada en el fichero SecuencialI.pdf
Presione una tecla para continuar . . .
```

```
Busqueda BinariaR. Tiempos de ejecucion promedio

        Talla           Tiempo (mseg)

        100                     1.7
        200                     1.9
        300                     1.7
        400                     1.8
        500                     1.9
        600                     1.9
        700                     1.9
        800                     1.8
        900                     1.8
        1000                    1.9

Datos guardados en el fichero BinariaR.dat
Generar grafica de resultados? (s/n): s
```



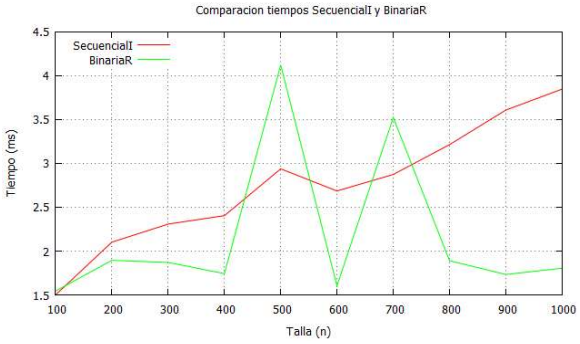

```
Grafica guardada en el fichero BinariaR.pdf
Presione una tecla para continuar . . .
```

```
*** COMPARACION DE DOS METODOS DE BUSQUEDA ***
Selecciona los dos metodos a comparar
0: Busqueda Secuencial Iterativa
1: Busqueda Binaria Recursiva
2: Busqueda Ternaria Recursiva
---------
Elige metodo 1: 0
Elige metodo 2: 1
```

```
usqueda SecuencialI y BinariaR. Tiempos de ejecucion promedio

                    SecuencialI                 BinariaR
     Talla          Tiempo (mseg)          Tiempo (mseg)
     100                 1.5                    1.5
     200                 2.1                    1.9
     300                 2.3                    1.9
     400                 2.4                    1.7
     500                 2.9                    4.1
     600                 2.7                    1.6
     700                 2.9                    3.5
     800                 3.2                    1.9
     900                 3.6                    1.7
     1000                3.8                    1.8

atos guardados en los ficheros SecuencialI.dat y BinariaR.dat
enerar grafica de resultados? (s/n): s
```




```
Grafica guardada en el fichero SecuencialIBinariaR.pdf
Presione una tecla para continuar . . .
```