

# Explotación de la Información.

## Curso 2024-2025

### Grado en Ingeniería Informática

### Universidad de Alicante

---

## Práctica 3: Buscador

---

### Contenido

Fecha entrega.....	2
Qué se pide.....	2
Modelo de similitud <i>Deviation From Randomness (DFR)</i> .....	2
Modelo de similitud <i>Okapi BM25</i> .....	3
Ejemplo de modo de uso de la cola de prioridad de STL .....	3
Prototipo de la clase “Buscador” .....	5
Aclaraciones .....	8
Evaluación de la práctica.....	8
Herencia pública.....	8
Ejemplo de fichero main.cpp .....	8
Control de tiempos de ejecución .....	9
Fichero makefile.....	10
Aclaraciones comunes a todas las prácticas .....	10
Forma de entrega.....	11
Ficheros a entregar y formato de entrega .....	11
Evaluación .....	14

## Fecha entrega

Del 2 al 6 de junio de 2025

## Qué se pide

Se pide construir la clase *Buscador* a partir de la indexación generada en la práctica anterior (*IndexadorHash*), implementando los modelos DFR y BM25.

Se valorará la eficiencia de la implementación, especialmente cualquier modificación en la representación interna (parte privada de las clases) o el algoritmo utilizado.

## Modelo de similitud *Deviation From Randomness (DFR)*

$$sim(q, d) = \sum_{i=1}^k w_{i,q} * w_{i,d}$$

$$w_{t,d} = \left( \log_2(1 + \lambda_t) + f_{t,d}^* * \log_2 \frac{1 + \lambda_t}{\lambda_t} \right) * \frac{f_t + 1}{n_t * (f_{t,d}^* + 1)}$$

$$w_{t,q} = \frac{f_{t,q}}{k}$$

$$f_{t,d}^* = f_{t,d} * \log_2 \left( 1 + \frac{c * avr\_l_d}{l_d} \right)$$

$$f_t = \sum_{i=1}^N f_{t,i} \quad \lambda_t = \frac{f_t}{N}$$

Siendo el sumatorio para los términos de  $q$  que están en  $d$ :

- $q$ : la query o pregunta que realiza el usuario
- $d$ : el documento del que se calcula su valor de similitud  $sim(q, d)$  respecto a la pregunta  $q$  que realiza el usuario
- $k$ : número de términos (no de parada) de la query  $q$
- $w_{i,q}$ : peso en la query del término  $i$  de la query  $q$
- $w_{i,d}$ : peso en el documento del término  $i$  de la query  $q$
- $f_t$ : número total de veces que el término  $t$  aparece en toda la colección
- $f_{t,d}$ : número de veces que el término  $t$  aparece en el documento  $d$
- $f_{t,q}$ : número de veces que el término  $t$  aparece en la query  $q$
- $n_t$ : número de documentos en los que aparece el término  $t$
- $l_d$ : longitud en palabras (no de parada) del documento
- $avr\_l_d$ : media en palabras (no de parada) de los documentos
- $N$ : cantidad de documentos en la colección
- Valor recomendado de  $c = 2$

- $\lambda_i$  es la razón entre la frecuencia del término en la colección y la cantidad de documentos en la colección

### Modelo de similitud *Okapi BM25*

Dada una consulta  $Q$ , que contiene las palabras clave  $q_1, \dots, q_n$  ([https://en.wikipedia.org/wiki/Okapi\\_BM25](https://en.wikipedia.org/wiki/Okapi_BM25)):

$$\text{score}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}$$

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

Siendo el sumatorio para los términos de  $q$  que están en  $d$ . Los logaritmos serían en base 2 y es posible que algunos valores  $\text{score}(D, Q)$  sean negativos (p.ej. cuando  $\text{IDF}(q)$  lo sea para los  $q_i$  que aparezcan en más de la mitad de los documentos de la colección):

- $\text{score}(D, Q)$ : valor de similitud para el documento  $D$  respecto a la pregunta  $Q$  que realiza el usuario
- $n$ : número de términos (no de parada) de la query  $Q$
- $f(q_i, D)$ : la frecuencia del término  $q_i$  en el documento  $D$
- $|D|$ : el número de palabras (no de parada) del documento  $D$
- $\text{avgdl}$ : la media de todas las  $|D|$  en la colección
- $N$ : cantidad de documentos en la colección
- $n(q_i)$ : número de documentos en los que aparece el término  $q_i$
- Constantes para configuración:  $k_1 = 1,2$   $b = 0,75$

### Ejemplo de modo de uso de la cola de prioridad de STL

Un modo de almacenar los resultados de la búsqueda es mediante la cola de prioridad `priority_queue` de STL ([http://www.cplusplus.com/reference/queue/priority\\_queue/push/](http://www.cplusplus.com/reference/queue/priority_queue/push/)). **Se podrá cambiar la forma de implementación con el objetivo de mejorar la eficiencia.** A continuación, se muestra un ejemplo de modo de uso, para el que se crea una cola de objetos `ResultadoIR` que contienen los tríos (valor de similitud del documento, identificador del documento, número de pregunta):

```
#include <iostream>
#include <queue>

using namespace std;

class ResultadoRI {
    friend ostream& operator<<(ostream&, const ResultadoRI&);
public:
    ResultadoRI(const double& kvSimilitud, const long int& kidDoc, const
int& np);
    double VSimilitud() const;
    long int IdDoc() const;
    bool operator< (const ResultadoRI& lhs) const;
```

```

private:
    double vSimilitud;
    long int idDoc;
    int numPregunta;
};

ResultadoRI::ResultadoRI(const double& kvSimilitud, const long int& kidDoc,
const int& np)
{
    vSimilitud = kvSimilitud;
    idDoc = kidDoc;
    numPregunta = np;
}

double
ResultadoRI::VSimilitud() const
{
    return vSimilitud;
}

long int
ResultadoRI::IdDoc() const
{
    return idDoc;
}

bool
ResultadoRI::operator< (const ResultadoRI& lhs) const
{
    if(numPregunta == lhs.numPregunta)
        return (vSimilitud < lhs.vSimilitud);
    else
        return (numPregunta > lhs.numPregunta);
}

ostream&
operator<<(ostream &os, const ResultadoRI &res){
    os << res.vSimilitud << "\t\t" << res.idDoc << "\t" << res.numPregunta
<< endl;
    return os;
}

int main(){
    priority_queue<ResultadoRI> mypq;

    mypq.push(ResultadoRI(30, 1, 1));
    mypq.push(ResultadoRI(100, 2, 1));
    mypq.push(ResultadoRI(100, 5, 1));
    mypq.push(ResultadoRI(25, 3, 2));
    mypq.push(ResultadoRI(40, 4, 2));
    mypq.push(ResultadoRI(200, 4, 2));
    mypq.push(ResultadoRI(250, 4, 2));
    mypq.push(ResultadoRI(400, 4, 3));
    mypq.push(ResultadoRI(40, 4, 3));

    cout << "Mostrando la cola de
prioridad...\nvSimilitud\tNumDoc\tNumPregunta\n";

    while (!mypq.empty())
    {
        cout << mypq.top();
        mypq.pop();
    }
    cout << endl;

    // SALIDA POR PANTALLA:
    //Mostrando la cola de prioridad...
    //vSimilitud NumDoc NumPregunta

```

```

        //100      2      1
        //100      5      1
        //30       1      1
        //250      4      2
        //200      4      2
        //40       4      2
        //25       3      2
        //400      4      3
        //40       4      3

    return 0;
}

```

## Prototipo de la clase “Buscador”

A continuación, se muestra el prototipo de la clase *Buscador*. La parte privada de la clase se podrá modificar a decisión del alumno para mejorar al máximo su eficiencia. La parte pública mantendrá los prototipos aquí indicados, pero se permitirá añadir nuevos enriquecimientos siempre que no hagan referencia a la parte privada de la clase. Esta clase aparecerá en los ficheros *buscador.h* y *buscador.cpp*. Destacar que se podrá modificar el carácter `const` de los métodos/argumentos aquí descritos siempre que se mantengan las características de funcionamiento previstas.

```

class Buscador: public IndexadorHash {

    friend ostream& operator<<(ostream& s, const Buscador& p) {
        string preg;
        s << "Buscador: " << endl;
        if(DevuelvePregunta(preg))
            s << "\tPregunta indexada: " << preg << endl;
        else
            s << "\tNo hay ninguna pregunta indexada" << endl;
        s << "\tDatos del indexador: " << endl << (IndexadorHash) p;
        // Invoca a la sobrecarga de la salida del Indexador

        return s;
    }

public:
    Buscador(const string& directorioIndexacion, const int& f);
    // Constructor para inicializar Buscador a partir de la indexación
    // generada previamente y almacenada en "directorioIndexacion". En caso que
    // no exista el directorio o que no contenga los datos de la indexación se
    // enviará a cerr la excepción correspondiente y se continuará la ejecución
    // del programa manteniendo los índices vacíos
    // Inicializará la variable privada "formSimilitud" a "f" y las
    // constantes de cada modelo: "c = 2; k1 = 1.2; b = 0.75;"

    Buscador(const Buscador&);

    ~Buscador();

    Buscador& operator= (const Buscador&);

    // En los métodos de "Buscar" solo se evaluarán TODOS los documentos que
    // contengan alguno de los términos de la pregunta (tras eliminar las
    // palabras de parada).
    bool Buscar(const int& numDocumentos = 99999);
    // Devuelve true si en IndexadorHash.pregunta hay indexada una pregunta
    // no vacía con algún término con contenido, y si sobre esa pregunta se
    // finaliza la búsqueda correctamente con la fórmula de similitud indicada
    // en la variable privada "formSimilitud".

```

```

// Por ejemplo, devuelve falso si no finaliza la búsqueda por falta de
memoria, mostrando el mensaje de error correspondiente, e indicando el
documento y término en el que se ha quedado.
// Se guardarán los primeros "numDocumentos" documentos más relevantes
en la variable privada "docsOrdenados" en orden decreciente según la
relevancia sobre la pregunta (se vaciará previamente el contenido de
esta variable antes de realizar la búsqueda). Se almacenarán solo los
documentos que compartan algún término (no de parada) con la query
(aunque ese número de documentos sea inferior a "numDocumentos"). Como
número de pregunta en "ResultadoRI.numPregunta" se almacenará el valor
0. En caso de que no se introduzca el parámetro "numDocumentos",
entonces dicho parámetro se inicializará a 99999)

bool Buscar(const string& dirPreguntas, const int& numDocumentos, const
int& numPregInicio, const int& numPregFin);
// Realizará la búsqueda entre el número de pregunta "numPregInicio" y
"numPregFin", ambas preguntas incluidas. El corpus de preguntas estará
en el directorio "dirPreguntas", y tendrá la estructura de cada pregunta
en un fichero independiente, de nombre el número de pregunta, y
extensión ".txt" (p.ej. 1.txt 2.txt 3.txt ... 83.txt). Esto significa
que habrá que indexar cada pregunta por separado y ejecutar una búsqueda
por cada pregunta añadiendo los resultados de cada pregunta (junto con
su número de pregunta) en la variable privada "docsOrdenados". Asimismo,
se supone que previamente se mantendrá la indexación del corpus.
// Se guardarán los primeros "numDocumentos" documentos más relevantes
para cada pregunta en la variable privada "docsOrdenados". Se
almacenarán solo los documentos que compartan algún término (no de
parada) con la query (aunque ese número de documentos sea inferior a
"numDocumentos").
// La búsqueda se realiza con la fórmula de similitud indicada en la
variable privada "formSimilitud".
// Devuelve falso si no finaliza la búsqueda (p.ej. por falta de
memoria), mostrando el mensaje de error correspondiente, indicando el
documento, pregunta y término en el que se ha quedado.

void ImprimirResultadoBusqueda(const int& numDocumentos = 99999) const;
// Imprimirá por pantalla los resultados de la última búsqueda (un
número MÁXIMO de "numDocumentos" (en caso de que no se introduzca el
parámetro "numDocumentos", entonces dicho parámetro se inicializará a
99999) por cada pregunta, entre los que tienen algún término de la
pregunta), los cuales estarán almacenados en la variable privada
"docsOrdenados" en orden decreciente según la relevancia sobre la
pregunta, en el siguiente formato (una línea por cada documento):
    NumPregunta FormulaSimilitud NomDocumento Posicion PuntuacionDoc
    PreguntaIndexada
// Donde:
    NumPregunta sería el número de pregunta almacenado en
    "ResultadoRI.numPregunta"
    FormulaSimilitud sería: "DFR" si la variable privada
    "formSimilitud == 0"; "BM25" si es 1.
    NomDocumento sería el nombre del documento almacenado
    en la indexación (habrá que extraer el nombre del documento
    a partir de "ResultadoRI.idDoc", pero sin el directorio
    donde esté almacenado ni la extensión del archivo)
    Posicion empezaría desde 0 (indicando el documento más
    relevante para la pregunta) incrementándose por cada
    documento (ordenado por relevancia). Se imprimirá un máximo
    de líneas de "numDocumentos" (es decir, el máximo valor de
    este campo será "numDocumentos - 1")
    PuntuacionDoc sería el valor numérico de la fórmula de
    similitud empleada almacenado en "ResultadoRI.vSimilitud".
    Se mostrarán los decimales con el punto en lugar de con la
    coma.
    PreguntaIndexada se corresponde con IndexadorHash.pregunta
    si "ResultadoRI.numPregunta == 0". En caso contrario se
    imprimirá "ConjuntoDePreguntas"

// Por ejemplo:
1 BM25 EFE19950609-05926 0 64.7059 ConjuntoDePreguntas

```

```

1 BM25 EFE19950614-08956 1 63.9759 ConjuntoDePreguntas
1 BM25 EFE19950610-06424 2 62.6695 ConjuntoDePreguntas
2 BM25 EFE19950610-00234 0 0.11656233535972 ConjuntoDePreguntas
2 BM25 EFE19950610-06000 1 0.10667871616613 ConjuntoDePreguntas
// Este archivo debería usarse con la utilidad "trec_eval -q -o
frelevancia_trec_eval_TIME.txt fich_salida_buscadador.txt >
fich_salida_trec_eval.res", que se consigue compilando "make" del
directorio "trec_eval_8_1_linux" de "materiales_buscadador.tgz", para
obtener los datos de precisión y cobertura

bool ImprimirResultadoBusqueda(const int& numDocumentos, const string&
nombreFichero) const;
// Lo mismo que "ImprimirResultadoBusqueda()" pero guardando la salida
en el fichero de nombre "nombreFichero"
// Devolverá false si no consigue crear correctamente el archivo

int DevolverFormulasimilitud() const;
// Devuelve el valor del campo privado "formSimilitud"

bool CambiarFormulasimilitud(const int& f);
// Cambia el valor de "formSimilitud" a "f" si contiene un valor
correcto (f == 0 || f == 1);
// Devolverá false si "f" no contiene un valor correcto, en cuyo caso no
cambiaría el valor anterior de "formSimilitud"

void CambiarParametrosDFR(const double& kc);
// Cambia el valor de "c = kc"

double DevolverParametrosDFR() const;
// Devuelve el valor de "c"

void CambiarParametrosBM25(const double& k1, const double& kb);
// Cambia el valor de "k1 = k1; b = kb;"

void DevolverParametrosBM25(double& k1, double& kb) const;
// Devuelve el valor de "k1" y "b"

private:
Buscador();
// Este constructor se pone en la parte privada porque no se permitirá
crear un buscador sin inicializarlo convenientemente a partir de una
indexación.
// Se inicializará con todos los campos vacíos y la variable privada
"formSimilitud" con valor 0 y las constantes de cada modelo: "c = 2; k1
= 1.2; b = 0.75"

priority_queue< ResultadoRI > docsOrdenados;
// Contendrá los resultados de la última búsqueda realizada en orden
decreciente según la relevancia sobre la pregunta. El tipo
"priority_queue" podrá modificarse por cuestiones de eficiencia. La
clase "ResultadoRI" aparece en la sección "Ejemplo de modo de uso de la
cola de prioridad de STL"

int formSimilitud;
// 0: DFR, 1: BM25

double c;
// Constante del modelo DFR

double k1;
// Constante del modelo BM25

double b;
// Constante del modelo BM25
};

```

## Aclaraciones

### Evaluación de la práctica

La nota se calculará del siguiente modo:

- 30% por la corrección automática con los ficheros de prueba.
- 20% por el cálculo de la eficiencia temporal para una sola pregunta.
- 20% por el cálculo de la eficiencia temporal para todas las preguntas del corpus.
- 30% por la revisión de la memoria entregada, en la que se justificarán/evaluarán las mejoras intentadas/implementadas en la práctica.

### Herencia pública

Al realizarse la herencia pública de la clase *IndexadorHash* nos permite tener acceso a toda la parte pública de dicha clase, además de simplificar los constructores. Por ejemplo:

```
Buscador::Buscador(const string& directorioIndexacion, const int& f):
IndexadorHash(directorioIndexacion)
{
    formSimilitud = f;
    c = 2; k1 = 1.2; b = 0.75;
}

Buscador(const Buscador& busc): IndexadorHash(busc)
{
    formSimilitud = busc.f;
    c = busc.c; k1 = busc.k1; b = busc.b;
}

~Buscador(){ }

Buscador& operator= (const Buscador& busc)
{
    formSimilitud = busc.f;
    c = busc.c; k1 = busc.k1; b = busc.b;
}
```

### Ejemplo de fichero main.cpp

A continuación, se muestra un posible ejemplo de fichero *main.cpp*:

```
#include <iostream>
#include <string>
#include "buscador.h"
#include "indexadorHash.h"

using namespace std;

main() {
    IndexadorHash b("./StopWordsEspanyol.txt", ".", ":", false, false,
    "./indicePrueba", 0, false, false);

    b.Indexar("./listaFicheros_corto.txt");
    b.GuardarIndexacion();

    Buscador a("./indicePrueba", 0);
    cout << "Buscador: " << a;
    cout << a.DevolverTipoStemming () << endl;
    // Estaría accediendo al método de la clase "IndexadorHash" de la
    que "Buscador" está heredando
```



```

        a.IndexarPregunta("documentos sobre la Guerra Civil española");
        if(a.Buscar(20))
            a.ImprimirResultadoBusqueda(10);

        if(a.CambiarFormulaSimilitud(1))
        {
            if(a.Buscar(20))
                a.ImprimirResultadoBusqueda(10);
        }
    }
}

```

## Control de tiempos de ejecución

Seguidamente se muestra el **fichero `main.cpp`**, con el que se realizará el estudio de **tiempos de ejecución**, el cual se utilizará para evaluar las prácticas:

```

#include <iostream>
#include <string>
#include <list>
#include <sys/resource.h>
#include "tokenizador.h"

using namespace std;

double getcputime(void) {
    struct timeval tim;
    struct rusage ru;
    getrusage(RUSAGE_SELF, &ru);
    tim=ru.ru_utime;
    double t=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;
    tim=ru.ru_stime;
    t+=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;
    return t;
}

main() {
    IndexadorHash b("./StopWordsEspanyol.txt", ". ,:", false, false,
    "./indicePruebaEspanyol", 0, false);
    b.Indexar("ficherosTimes.txt");
    b.GuardarIndexacion();

    Buscador a("./indicePruebaEspanyol", 0);

    a.IndexarPregunta("KENNEDY ADMINISTRATION PRESSURE ON NGO DINH DIEM TO
    STOP SUPPRESSING THE BUDDHISTS . ");
    aa=getcputime();
    a.Buscar(423);
    a.ImprimirResultadoBusqueda(423);

    double bb=getcputime()-aa;
    cout << "\nHa tardado " << bb << " segundos\n\n";

    time_t inicioB, finB;
    time(&inicioB);
    double aaB=getcputime();
    a.Buscar("/home/tad/12-13/eiBuscador/CorpusTime/Preguntas/", 423, 1,
    83);
    a.ImprimirResultadoBusqueda(423);

    double bbB=getcputime()-aaB;
    cout << "\nHa tardado " << bbB << " segundos\n\n";
}

```

## Fichero makefile

La práctica se corregirá utilizando el siguiente fichero makefile:

```
.PHONY= clean

CC=g++
OPTIONS= -g
DEBUG= #-D DEBUG
LIBDIR=lib
INCLUDEDIR=include
_OBJ=      buscador.o      indexadorHash.o      tokenizador.o      stemmer.o
indexadorInformacion.o
OBJ = $(patsubst %, $(LIBDIR)/%, $( _OBJ) )

all: buscador

buscador:      src/main.cpp $(OBJ)
              $(CC) $(OPTIONS) $(DEBUG) -I$(INCLUDEDIR) src/main.cpp $(OBJ) -o
buscador

$(LIBDIR)/%.o : $(LIBDIR)/%.cpp $(INCLUDEDIR)/%.h
              $(CC) $(OPTIONS) $(DEBUG) -c -I$(INCLUDEDIR) -o $@ $<

clean:
      rm -f $(OBJ)
```

## Aclaraciones comunes a todas las prácticas

Se puede utilizar la librería STL, para lo que se puede consultar en <http://en.cppreference.com/w/> o en <http://www.cplusplus.com/>.

Se aconseja el uso de la herramienta VALGRIND para comprobar el manejo correcto de la memoria dinámica. Modo de uso:

```
valgrind - --tool=memcheck - --leak-check=full nombre_del_ejecutable
```

Todas las operaciones especificadas son obligatorias.

Si una clase hace uso de otra clase, en el código nunca se debe incluir el fichero .cpp, sólo el .h.

En la parte PUBLIC no debe aparecer ninguna operación que haga referencia a la representación del tipo, sólo se pueden añadir operaciones de enriquecimiento de la clase.

En la parte PRIVATE de las clases se pueden añadir todos los atributos y métodos que sean necesarios para la implementación de los tipos.

Tratamiento de excepciones:

Todos los métodos darán un mensaje de error (en cerr) cuando el alumno determine que se produzcan excepciones; para ello, se pueden añadir en la parte privada de la clase aquellas operaciones y variables auxiliares que se necesiten para controlar las excepciones.

Se considera excepción aquello que no permite la normal ejecución de un programa (por ejemplo, problemas al reservar memoria, problemas al abrir un fichero, etc.). NO

se considera excepción aquellos errores de tipo lógico debidos a las especificidades de cada clase.

De cualquier modo, todos los métodos deben devolver siempre una variable del tipo que se espera.

Los mensajes de error se mostrarán siempre por la salida de error estándar (cerr). El formato será:

ERROR: mensaje\_de\_error (al final un salto de línea).

### Forma de entrega

La práctica se programará en el Sistema Operativo Linux, y en el lenguaje C++. Deberá compilar con la versión instalada en los laboratorios de la Escuela Politécnica Superior.

La entrega de la práctica se realizará:

- En el SERVIDOR DE PRÁCTICAS en <http://pracdlsi.dlsi.ua.es/>
- A título INDIVIDUAL; por tanto requerirá del alumno que conozca su USUARIO y CONTRASEÑA en el Servidor de Prácticas.

Se podrá realizar cuantas entregas quiera el alumno: solo se corregirá la última práctica entregada. Tras cada entrega, el servidor enviará al alumno un INFORME DE COMPILACIÓN, para que el alumno compruebe que lo que ha entregado cumple las especificaciones pedidas y que se ha podido generar el ejecutable correctamente. Este informe también se podrá consultar desde la página web de entrega de prácticas del DLSI (<http://pracdlsi.dlsi.ua.es> e introducir el nombre de usuario y password). En caso de que la práctica esté correctamente entregada, compilada y ejecutada, en este informe debe salir lo siguiente:

=====

DIFERENCIA CON FICHERO DE SALIDA DE REFERENCIA

=====

-----

Significa que la ejecución ha sido correcta y coincide con la salida esperada en el fichero de comprobación web publicado en campus virtual.

No es necesario entregar ni el makefile ni el main.cpp.

Para la entrega y corrección se utilizará el siguiente makefile (previamente mostrado) y el fichero "main.cpp" que se publicará posteriormente en campus virtual.

### Ficheros a entregar y formato de entrega

La práctica debe ir organizada en 3 subdirectorios:

DIRECTORIO 'include': contiene los ficheros (en MINÚSCULAS):

- “buscador.h”
- “indexadorHash.h”
- “tokenizador.h”
- “stemmer.h”
- “indexadorInformacion.h”

DIRECTORIO ‘lib’: contiene los ficheros (NO deben entregarse los ficheros objeto “.o”):

- “buscador.cpp”
- “indexadorHash.cpp”
- “tokenizador.cpp”
- “stemmer.cpp”
- “indexadorInformacion.cpp”

DIRECTORIO ‘src’: contiene **TODOS los ficheros aportados por el alumno para comprobación de la práctica:**

- Por ejemplo: “main.cpp” y “main.sal”
- El fichero **“graficaPrecisionCobertura.pdf”**. Debe tener al menos los siguientes apartados:
  - Introducción: presentación del problema, hipótesis de partida.
  - Análisis de la solución implementada: algoritmo, estructuras de datos, ...
  - Justificación de la solución elegida: detalle de qué soluciones alternativas se han probado, justificando su descarte.
  - Análisis de las mejoras realizadas en la práctica: tanto en el algoritmo como en las estructuras de datos utilizadas.
  - Análisis de eficiencia computacional (**no teórica**: no hay que calcular cotas de complejidad).
  - Análisis de precisión y cobertura:
    - Se incluirá la parte final resumen de todas las preguntas del contenido del archivo “fich\_salida\_trec\_eval.res” que se describen seguidamente, y la gráfica comparativa tras procesar las 83 preguntas del corpus TIME.
    - Habrá que ejecutar el análisis de las 83 preguntas para el corpus Time en inglés **con y sin stemming** (para el inglés) para los dos modelos de similitud (DFR/BM25) **con “quitar acentos y mayúsculas”**, sin *almacenarPosTerm*, y con los delimitadores del tokenizador por defecto:

```
a.Buscar(dirPreguntas, 423, 1, 83);
a.ImprimirResultadoBusqueda(423, "fich_salida_buscador_alumno.txt");
```

- Después habrá que ejecutar la utilidad “trec\_eval” que se consigue compilando “make” del directorio “trec\_eval\_8\_1\_linux” de “materiales\_buscador.tgz”:
 

“trec_eval	-q	-o
------------	----	----

frelevancia\_trec\_eval\_TIME.txt fich\_salida\_buscador\_alumno.txt >  
fich\_salida\_trec\_eval.res", para obtener los datos de precisión y cobertura. **IMPORTANTE:** el fichero "frelevancia\_trec\_eval\_TIME.txt" espera los documentos sin el camino donde están almacenados ni la extensión .tim. La salida esperada para el archivo "fich\_salida\_buscador\_alumno.txt" debería ser:

```
1 BM25 126 0 64.7059 ConjuntoDePreguntas
1 BM25 56 1 63.9759 ConjuntoDePreguntas
...
2 BM25 34 0 0.11656233535972 ConjuntoDePreguntas
2 BM25 6 1 0.10667871616613 ConjuntoDePreguntas
...
```

- Del fichero "fich\_salida\_trec\_eval.res", concretamente del apartado "All" se extraerán los valores de precisión interpolada:

Queryid (Num): All

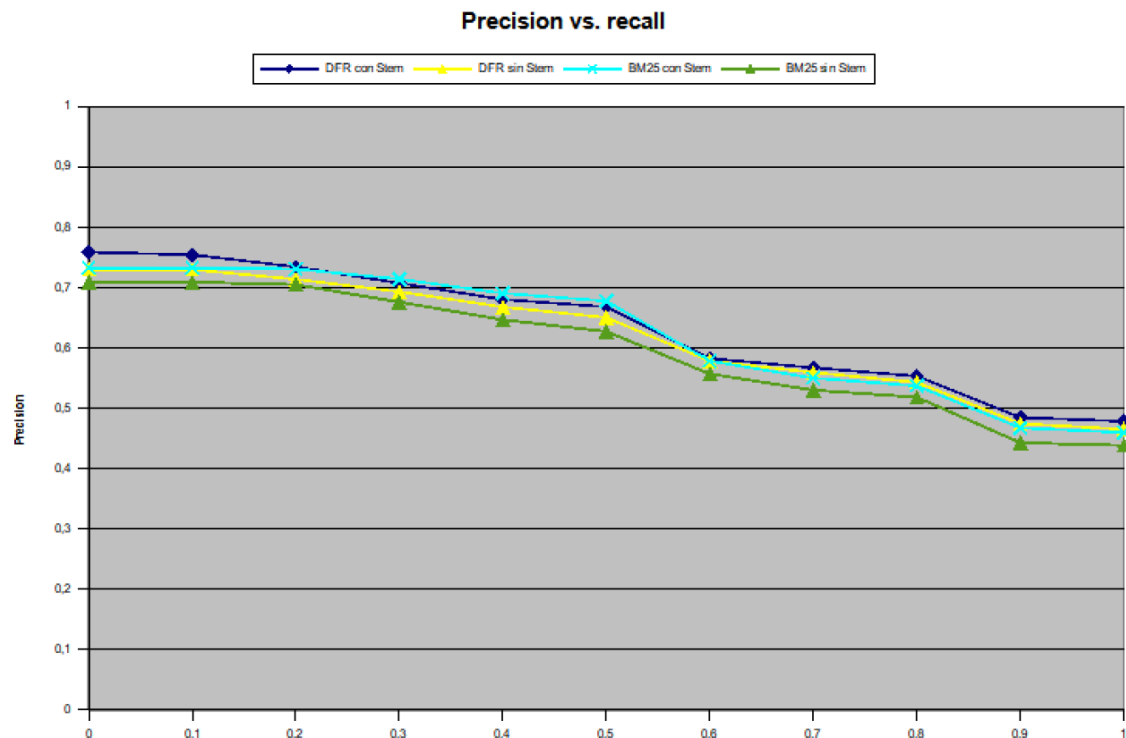
Interpolated Recall - Precision Averages:

at 0.00 **0.2857**  
at 0.10 **0.2857**  
at 0.20 **0.2857**  
at 0.30 **0.2798**  
at 0.40 **0.2798**  
at 0.50 **0.2798**  
at 0.60 **0.2332**  
at 0.70 **0.2332**  
at 0.80 **0.1977**  
at 0.90 **0.1977**  
at 1.00 **0.1977**

- Esos valores se introducirán en el fichero "graficaPrecisionCobertura.doc" para generar las **cuatro líneas** para los dos modelos de similitud (DFR/BM25) cuando se ha indexado **con y sin** stemming y **todos ellos con** "quitar acentos y mayúsculas" (p.ej. los siguientes resultados que NO son reales a obtener en vuestra práctica):

Recall	DFR	BM25		
0	0.6693	1		
0.1	0.6693	1		
0.2	0.6443	0.83		
0.3	0.604	0.83		
0.4	0.5873	0.83		
0.5	0.5835	0.83		
0.6	0.5116	0.83		
0.7	0.5107	0.5		
0.8	0.4835	0.4		
0.9	0.4197	0.3		
1	0.4197	0.2		

Debería dar una gráfica similar a la siguiente:



Además, en el directorio raíz, deberá aparecer el fichero “nombres.txt”: fichero de texto con los datos del autor. El formato de este fichero es:

1\_DNI: DNI1

1\_NOMBRE: APELLIDO1.1 APELLIDO1.2, NOMBRE1

Cuando llegue el momento de la entrega, toda la estructura de directorios ya explicada (¡ATENCIÓN! excepto el MAKEFILE), debe estar comprimida en un fichero de forma que éste NO supere los 300 K. Ejemplo:

```
user@srv4:~$ ls
```

```
include
lib
nombres.txt
src
```

```
user@srv4:~$ tar cvzf PRACTICA.tgz *
```

## Evaluación

La práctica se corregirá casi en su totalidad de un modo automático, por lo que los nombres de las clases, métodos, ficheros a entregar, ejecutables y formatos de salida descritos en el enunciado de la práctica SE HAN DE RESPETAR EN SU TOTALIDAD.

Uno de los objetivos de la práctica es que el alumno sea capaz de comprender un conjunto de instrucciones y sea capaz de llevarlas a cabo. Por tanto, es esencial ajustarse completamente a las especificaciones de la práctica.

Cuando se corrige la práctica, el corrector automático proporcionará ficheros de corrección llamados “main.cpp”. Este fichero utilizará la sintaxis definida para cada clase y los nombres de los ficheros asignados a cada una de ellas: únicamente contendrá una serie de instrucciones `#include` con los nombres de los ficheros “.h”.

Las prácticas no se pueden modificar una vez corregidas y evaluadas (no hay revisión del código). Por lo tanto, es esencial ajustarse a las condiciones de entrega establecidas en este enunciado.

En especial, se debe llevar cuidado con los nombres de los ficheros y el formato especificado para la salida.