Memoria del Indexador

```
numDocs: 2  numTotalPal: 11 numTotalPalSinParada: 7
numTotalPalDiferentes: 4    tamBytes: 53
```

```
corpus_cortofichero2.txt    idDoc: 2    numPal: 5   numPalSinParada: 3
  numPalDiferentes: 2 tamBytes: 23
corpus_corto/fichero1.txt   idDoc: 1    numPal: 6   numPalSinParada: 4
  numPalDiferentes: 3 tamBytes: 30
```

```
pal3    Frecuencia total: 1 fd: 1   Id.Doc: 1   ft: 1   4
pal2    Frecuencia total: 3 fd: 2   Id.Doc: 2   ft: 2   0   2   Id.Doc: 1   ft: 1   2
pal4    Frecuencia total: 1 fd: 1   Id.Doc: 2   ft: 1   3
pal1    Frecuencia total: 2 fd: 1   Id.Doc: 1   ft: 2   0   3
```

Asignatura: Explotación de la Información(EI)
Número de la Práctica: 2
Alumno: Saul Verdu Aparicio
Curso: 2020-2021

Índice

Introducción

En esta práctica se nos pide implementar un indexador que con ayuda del tokenizador creado en la práctica anterior sea capaz de indexar todas las palabras de un conjunto de documentos lo más eficientemente posible. Esta práctica se utilizará como base para la siguiente de las prácticas, por ello debe de estar construida de la mejor forma posible.

Análisis de la indexación en memoria

Tras varias pruebas, se ha optado por dividir la función principal (la función que se encarga de la indexación de un grupo de ficheros) en dos partes. La primera de ellas será la encargada de abrir el documento que tiene los nombre de los documentos a indexar para comprobar cuáles de ellos tienen que ser indexado y la segunda se encarga de indexar un documento solo (llamada IndexarDoc(const string &nom)).

```
bool IndexadorHash::Indexar(const string& ficheroDocumentos){

    ifstream file;                                          //Declaramos los elementros que vamos a usar
    string nomFichero;
    struct stat buffer;
    int id;
    InfDoc inf;
    bool indexar;

    file.open(ficheroDocumentos.c_str(),ios::binary);       // Intentamos abrir el ficheroStopWords de forma binaria

    if(file){
        stringstream strStream;                             // Creamos un stringstream para almacenar el fichero
        strStream << file.rdbuf();                          // Volcamos el fichore en la variable
        while (getline(strStream, nomFichero, '\n')){       // Recorremos por lineal el fichero

            if(nomFichero.length() != 0 && stat(nomFichero.c_str(), &buffer) != -1 && !S_ISDIR(buffer.st_mode)){ ... }
            else{ cerr << "ERROR!!!: No se ha podido abrir el archivo:\t" << nomFichero << endl; }

        }
    }
    else{ cerr << "ERROR!!!: No se ha podido abrir el archivo :\t" << ficheroDocumentos << endl; }

    file.close();                                   // Cerramos el fichero

    return true;
}
```

Dentro de Indexar (const string& ficheroDocumentos) lo primero que hacemos es abrir el fichero que contiene los nombres de los ficheros a tokenizar y vamos comprobando que el nombre del fichero no sea una cadena vacía, que el nombre exista en el sistema y que este no sea un directorio.

Una vez hecho esto se pasa a comprobar si el documento ya ha sido indexado previamente, si este no ha sido indexado se crea un nuevo objeto InfDoc y se le da un nuevo id. Por otro lado, si el documento a indexar ya ha sido previamente indexado se comprobará la última fecha de indexación de ese documento, para así saber si este tiene que volver a ser indexado.

```
bool IndexadorHash::Indexar(const string& ficheroDocumentos){

  ...

      if(nomFichero.length( ) != 0 && stat(nomFichero.c_str(), &buffer) != -1 && !S_ISDIR(buffer.st_mode)){
          if(indiceDocs.find(nomFichero) == indiceDocs.end()){    //Si el documento no esta indexado
              id = indiceDocs.size() + 1;
              indexar = true;
          }
          else{                                                   //Si el documento estaba ya indexado
              if(indiceDocs.find(nomFichero)->second.Posterior()){        // Si ya se ha indexado anteriormente
                  id = indiceDocs.find(nomFichero)->second.Get_IdDoc();   // Obtenemos el id
                  BorraDoc(nomFichero);                                   // Borramos los datos que teniamos de antes del documento
                  indexar = true;                                         // Indicamos que se tiene que reindexar el documento
              }else{
                  indexar = false;                                        // Indicamos que NO se tiene que reindexar
              }
              cerr << "WARNING!!!: Este documento ya ha sido indexado:\t" << nomFichero << endl;
          }

          if(indexar){                                            // Si es necesario indexamos el documento
              inf = InfDoc();                                     // Añadimos el documento
              inf.Set_IdDoc(id);
              indiceDocs.insert({nomFichero,inf});
              IndexarDoc(nomFichero);                             // Indexamos el documento
          }

      }
      else{ cerr << "ERROR!!!: No se ha podido abrir el archivo:\t" << nomFichero << endl; }

  ...
}
```

Una vez comprobado si se tiene que hacer la indexación del documento se pasa el nombre del fichero a la función IndexarDoc (const string& nom) el cual se va a encargar de abrir el fichero y leer todo su contenido. Una vez se tiene el documento en memoria principal solo lo tenemos que tokenizarlo, aplicar el stemmer, guardar la información de los términos en el índice (o crear el término dentro del índice si este no existe) y por último guardar la información del documento dentro del índice de documentos.

```
void IndexadorHash::IndexarDoc(const string& nom) {

    ifstream documento;
    InfDoc *infDocumento;
    int idDoc, pal, palParada;
    struct stat doc_buffer;
    InformacionTermino infoTerm;
    InfTermDoc infTermDoc;
    list<string> tokens;
    string delimitadores;
    stemmerPorter stm;
    unordered_set<string>dif;

    //Iniciamos las variables
    infDocumento = &indiceDocs.find(nom)->second;
    idDoc = (*infDocumento).Get_IdDoc();
    pal = 0;
    palParada = 0;
    delimitadores = tok.DelimitadoresPalabra();
    tok.DelimitadoresPalabra(delimitadores + "\n"); // Añadimos '\n' para poder tokenizar todo el fichero

    documento.open(nom.c_str(),ios::binary);        // Abrimos el documento

    if(documento){
        stringstream strStream;                     // Creamos un stringstream para almacenar el fichero
        strStream << documento.rdbuf();             // Volcamos el fichore en la variable
        tok.Tokenizar(strStream.str(), tokens);     // Sacamos los tokenes que contiene el documento

        for(string token : tokens){                 // Recorremos el documento token a token
            stm.stemmer(token, tipoStemmer);                // Aplicamos la Stematizacion al token
            if(stopWords.find(token) == stopWords.end()){   // Si no es una stopWord
                if(Existe(token)){                                                      // Si existe la palabra
                    indice.find(token)->second.nuevaReferencia(idDoc,pal,almacenarPosTerm);     //Añadimos la referencia
                }
                else{                                                   // Si NO existe la palabra
                    infoTerm = InformacionTermino();                    // Creamos la informacion del termino
                    infoTerm.nuevaReferencia(idDoc,pal,almacenarPosTerm);       // Agregamos la referencia al nuevo termino
                    indice.insert({token,infoTerm});                    // Metemos el termino en el indice
                }
                if(dif.find(token) == dif.end()){ dif.insert(token); }          // Si no hemos visto la palabra la añadimos
            }
            else{                                           // Si es una stopWord
                palParada++;
            }
            pal++;
        }

        stat(nom.c_str(), &doc_buffer);             // Colocamod el tmaño del documento en el buffer
        tok.DelimitadoresPalabra(delimitadores);    // Dejamos los delimitadores como estaban

        //Colocamos los datos del documento
        (*infDocumento).Set_numPal(pal);
        (*infDocumento).Set_numPalSinParada(pal - palParada);
        (*infDocumento).Set_numPalDiferentes(dif.size());
        (*infDocumento).Set_tamBytes(doc_buffer.st_size);

        // Colocamos los datos del documento en la coleccion
        informacionColeccionDocs.NuevaInfDoc(pal, palParada, indice.size(), doc_buffer.st_size);

    }else{ cerr << "ERROR!!!: No se ha podido abrir el archivo:\t" << nom << endl; }

    documento.close();

}
```

Justificación indexación en memoria

He decidido optar por esta alternativa a la hora de indexar porque creo que es una buena forma de diferenciar las acciones de comprobación de los nombres de ficheros, de la indexación de los ficheros. Esto permite mejorar su mantenimiento, depuración y optimización.

Mejoras de la práctica

Dentro del proceso de indexación lo más destacable en cuanto a la optimización que se ha hecho ha sido la apertura de ficheros en forma binaria, lo cual es mucho más rápido de leer por el procesador; hacer un volcado de fichero a una variable (de tipo stringstream), lo cual hace que solo hagamos un acceso a memoria; y el acceder al contenido del documento por medio de punteros, que ahorra un poco de espacio a la hora de asignar los datos del documento.

```
InfDoc *infDocumento;

infDocumento = &indiceDocs.find(nom)->second;
idDoc = (*infDocumento).Get_IdDoc();

(*infDocumento).Set_numPal(pal);
(*infDocumento).Set_numPalSinParada(pal - palParada);
(*infDocumento).Set_numPalDiferentes(dif.size());
(*infDocumento).Set_tamBytes(doc_buffer.st_size);
```

```
ifstream documento;

documento.open(nom.c_str(),ios::binary);

stringstream strStream;
strStream << documento.rdbuf();
```

Eficiencia computacional

Coste temporal:

```
saul@saul-VirtualBox:~/Escritorio/EI/P2$ ./indexador
ERROR!!!: No se ha podido abrir el archivo:     fichPrueba.txt
ERROR!!!: No se ha podido abrir el archivo:     fichPrueba2.txt
Ha tardado 3.19601 segundos
saul@saul-VirtualBox:~/Escritorio/EI/P2$ ./indexador
ERROR!!!: No se ha podido abrir el archivo:     fichPrueba.txt
ERROR!!!: No se ha podido abrir el archivo:     fichPrueba2.txt
Ha tardado 2.30686 segundos
saul@saul-VirtualBox:~/Escritorio/EI/P2$ ./indexador
ERROR!!!: No se ha podido abrir el archivo:     fichPrueba.txt
ERROR!!!: No se ha podido abrir el archivo:     fichPrueba2.txt
Ha tardado 2.39631 segundos
saul@saul-VirtualBox:~/Escritorio/EI/P2$
```

Coste espacial: