Explotación de la información

Práctica 2 - Indexador

1. Guardar y leer la indexación
2. Borrar un documento de la indexación

1. Guardar y leer la indexación

La indexación se guarda en ficheros de texto usando un fichero para cada variable de la clase IndexadorHash, por tanto la estructura de ficheros tras guardar la indexación quedaría:

_indice => indice.txt => GuardarIndice(); => LeerIndice()
_indiceDocs => indiceDocs.txt => GuardarIndiceDocs(); => LeerIndiceDocs()
_InformacionColeccionDocs => GuardarInformacionColeccionDocs(); => LeerICD()
_tok => tokenizador.txt => GuardarTokenizador(); => LeerTokenizador()
resto de campos => variables.txt => GuardarVariables(); => LeerVariables()

Para guardar los datos redirijo la salida del operador << a los ficheros de esta forma no he tenido que codificar la forma en que guardo los datos. Luego para recuperar la indexación obtengo los datos tokenizando el fichero con el mismo tokenizador general de la práctica 1, en los campos donde la cantidad de elementos varía en función de los documentos a indexar son fáciles de obtener ya que en la salida de cada valor antes de este se escribe de que trata con una o dos palabras.

La estructura que en general se ha usado para leer la indexación es la siguiente:

```
Ifstream file;
file.open("ruta del fichero a leer");

While ( getline(file, linea) ) {

        std::list<std::string> tokens;
        _tokenizador.Tokenizar(linea, tokens);

        for (auto t = tokens.begin(); t != tokens.end(); ++t) {
                /* procesar la linea */
        }

}
```

En cuanto a la complejidad de leer la indexación dependerá de la cantida de lineas a leer por el bucle exterior y a esta se le aplicará el producto del bucle interior que depende de la cantida de tokens extraida de la linea.

2. Borrar un documento de la indexación

Dado que para borrar un documento de la indexación se proporciona su ruta y estos se indexan a partir de un identificador de tipo entero he añadido un mapa que relaciona la ruta de un documento indexado con su identificador de esta forma obtener el id de un documento que se quiere borrar se puede hacer con complejidad lineal. Este mapa "_documentos" se rellena durante la indexación. La implementación quedaría de la siguiente forma:

```
if (_indiceDocs.find(nomDoc) != _indiceDocs.end()) {
// Decrementar los contadores
int id_doc = getDocId(nomDoc);                  // Necesitamos el id del documento
for (auto& i : _indice) {
        // Referencia a la variable privada _l_docs
        auto& lista_doc = i.second.apuntarListaDocs();
        // Comprobar si en _l_docs está el documento
        auto d = lista_doc.find(id_doc);
        if (d != lista_doc.end()) {
                // Borrar d de la _l_docs // Restar la ft
        }
}
```

Para acabar recorro una vez más la indexación para borrar aquellos índices cuya lista de documentos llega a ser cero por tanto ya no están indexados.

```
std::stack<std::string> s;
// Lista en una pila S los indices cuya _l_docs pasa a ser 0
for (auto it=_indice.begin(); it!=_indice.end(); ++it) {
        auto l = it->second.apuntarListaDocs();         // Referencia a _l_docs
        if (l.size() == 0) {s.push(it->first);}    // IF _l_docs.size() == 0 THEN store i key
}

// Borrar de _indice aquellos i que almacena la pila
std::string i = "";
while (!s.empty()) {
        i = s.top();
        s.pop();
        _indice.erase(i);
}
```