Máster Universitario en Ingeniería Informática

Inteligencia Computacional

GESTIÓN DE INFORMACIÓN EN LA WEB

Práctica 1:

Desarrollo de un Sistema de Recuperación de Información con Lucene

UNIVERSIDAD DE GRANADA

Carlos Morales Aguilera
75925767-F
carlos7ma@correo.ugr.es

Curso Académico 2020-2021

Índice

1. Descripción de la práctica

La práctica a resolver consiste en la elaboración y desarrollo de un Sistema de Recuperación de Información (SRI) utilizando la librería de Java *Lucene*, la cual consiste en una API de recuperación de información de Apache. Se utilizará la versión más reciente, siendo esta *Lucene 8.8.1*.

El lenguaje de programación escogido es el de Java, al ser el nativo de la librería, y al encontrar un mayor soporte y documentación del mismo. La práctica será desarrollada apoyándose del IDE de programación en Java *NetBeans 12.0*.

La idea de la práctica consistirá en realizar una aplicación de indexador que permita indexar los documentos indicados en una carpeta, un motor de búsqueda que permita realizar búsquedas por términos. Además se realizará una interfaz gráfica en NetBeans que permita visualizar todo el proyecto.

1.1. Estructura del proyecto

El proyecto se compone de 3 ficheros principales:

- **Indexer.java**: Se implementa la lógica del indexador, donde lee el fichero, se define un analizador inglés, la configuración del indexador y lectura de carpetas y ficheros para la indexación.
- **SearchEngine.java**: Se implementa la lógica del motor de búsqueda, donde se define el buscador, responde a la consulta y se valora la puntuación.
- **GUI.java**: Se implementa una interfaz diseñada en NetBeans donde se muestra una barra para indicar la carpeta a indexar, la consulta a realizar por término.A la derecha se muestra una columna con los ficheros que se corresponden a la consulta y el contenido de uno de ellos si clickamos sobre él.

1.2. Desarrollo

La práctica se ha compuesto de tres partes como se indica anteriormente, para ello primero se ha realizado la implementación del indexador viendo ejemplos oficiales y de páginas web interesantes que han resultado de apoyo.

1.2.1. Indexador

El clase indexador contiene un método principal *Index* que se encarga de recibir la carpeta destino donde indexar (por defecto *indexes*), la carpeta que contiene los documentos (por defecto *documents*) y el fichero de palabras vacías. Este método comprueba que se puede leer la carpeta de documentos y crea la carpeta de índices.

Por último se encarga de configurar el indexador de Lucene como en los ejemplos oficiales y llama a otro método que se encarga de indexar la carpeta que contiene los documentos:

```
/**
    * Method for indexing documents in a SRI.
    * @param indexFolder route of the directory used to store indexes.
    * @param documentsFolder route of the directory that contains documents.
    * @param stopWordsFile route of the stop words file provided.
    */
   public void Index(String indexFolder, String documentsFolder, String
       stopWordsFile) throws IOException, Exception{

   // Get path to documents Folder
   final Path docFolder = Paths.get(documentsFolder);

   // Check if the documents folder is readable
   if (!Files.isReadable(docFolder)) {
     System.out.println(docFolder.toAbsolutePath() + " ´doesnt exist.");
     throw new Exception(docFolder.toAbsolutePath() + " ´doesnt exist.");
   }

   // Get start time
   Date start = new Date();

   try {
     // Open directory where indexes are going to be stored
     Directory indFolder = FSDirectory.open(Paths.get(indexFolder));

     // Read stop words file
     String stopWords = FileUtils.readFileToString(new File(stopWordsFile),
         "UTF-8");
     // Parse stop words to array to separate them
     String[] stopWordsSplitted = stopWords.split("\\s+");
     // Define analyzer
     EnglishAnalyzer analyzer = new EnglishAnalyzer(new
         CharArraySet(Arrays.asList(stopWordsSplitted),true));

     // Set index writer configuration from index analyzer
     IndexWriterConfig indexWriterConf = new IndexWriterConfig(analyzer);
     indexWriterConf.setOpenMode(IndexWriterConfig.OpenMode.CREATE);

     // Define index writer with folder specified to store indexes
     IndexWriter indWriter = new IndexWriter(indFolder, indexWriterConf);
     // Write indexes of folder
     indexFolder(indWriter, docFolder);
     // Close index writer
     indWriter.close();

     // Get end time
     Date end = new Date();
     System.out.println("Indexer finished in " + (end.getTime() -
         start.getTime()) + " ms.");

   } catch (IOException e) {
   }
 }
```

A continuación se define un método *IndexFolder* que como su nombre indica indexa la carpeta, esto se realiza leyendo la carpeta y recorriendo su contenido fichero a fichero y llamando a una función que indexa los documentos uno por uno:

```
/**
   * Method for indexing documents given an IndexWriter and folder to store them
       individually.
   * @param indWriter index writer.
   * @param indFolder folder where indexes will be stored.
   */
  static void indexFolder(final IndexWriter indWriter, Path indFolder) throws
      IOException {
    // Check if indFolder is a directory
    if (Files.isDirectory(indFolder)) {
      // Walk through the folder to iterate over it
      Files.walkFileTree(indFolder, new SimpleFileVisitor<Path>() {
        @Override
        // Action on a file (visit)
        public FileVisitResult visitFile(Path filePath, BasicFileAttributes
            fileattrbs) throws IOException {
          try {
            // Index a single document
            indexDocument(indWriter, filePath);
          } catch (IOException ignore) {

          }
          // Continue walking through files tree
          return FileVisitResult.CONTINUE;
        }
      });
    }
  }
```

Por último, el elemento principal del indexador consiste en el método *indexDocument* la cual indexa un documento almacenando su *path* y leyendo línea a línea almacenándolas de forma que se permita reconstruir el contenido del mismo con el mismo formato:

```
/**
   * Method for indexing a single document.
   * @param indWriter index writer.
   * @param filePath file to be indexed.
   */
  static void indexDocument(IndexWriter indWriter, Path filePath) throws
      IOException {
        // Try to read file
   try (InputStream stream = Files.newInputStream(filePath)) {
              // Define document
      Document doc = new Document();
              // Define path field
      Field pathField = new StringField("path", filePath.toString(),
          Field.Store.YES);
              // Add path field to document
      doc.add(pathField);
              // Read file content
              String content;
              // Define a file buffered reader
              BufferedReader reader = new BufferedReader(new FileReader
                  (filePath.toString()));
              // Define line
              String line = null;
              StringBuilder stringBuilder = new StringBuilder();
              String lineSeparator = System.getProperty("line.separator");
              // Read line by line
              while((line = reader.readLine()) != null) {
                  stringBuilder.append(line);
                  stringBuilder.append(lineSeparator);
              }
              // Content to String
              content = stringBuilder.toString();
              // Close reader
              reader.close();
              // Add content to document
              doc.add(new TextField("contents", content,Field.Store.YES));

              // Check if index writer is creating or updating document
      if (indWriter.getConfig().getOpenMode() == OpenMode.CREATE) {
                      // Add coument
                      System.out.println("Added " + filePath);
         indWriter.addDocument(doc);
      } else {
                      // Update document
         System.out.println("Updated " + filePath);
         indWriter.updateDocument(new Term("path", filePath.toString()), doc);
      }
   }
}
```

1.2.2. Motor de Búsqueda

El motor de búsqueda consiste en un único método *search* el cual realiza la búsqueda de un término dada una consulta de un término, la carpeta de los índices y el fichero de palabras vacías (por defecto los mismos valores que en el indexador). Lee la carpeta de los índices y define el analizador y el parser de la consulta, para poder realizar la búsqueda.

Por último obtiene todos los documentos que coinciden con la búsqueda (máximo 500) y los ordena según su relevancia basándose en el *score* obtenido:

```
/**
     * Method that searchs in index folder for an specific query.
     * @param indexFolder
     * @param stopWordsFile
     * @param queryInput
     * @return arraylist of documents that satysfies the query.
     */
    public ArrayList<Document> search(String indexFolder, String stopWordsFile,
        String queryInput) throws IOException, ParseException{
    // Define arraylist of documents
    ArrayList<Document> documents = new ArrayList<Document>();

    // Define index reader using index folder path
    IndexReader indReader =
        DirectoryReader.open(FSDirectory.open(Paths.get(indexFolder)));
    // Define index searcher using index reader
    IndexSearcher indSearcher = new IndexSearcher(indReader);

    // Read stop words file
    String stopWords = FileUtils.readFileToString(new File(stopWordsFile),
        "UTF-8");
    // Split stop words in array
    String[] stopWordsSplitted = stopWords.split("\\s+");
    // Define analyzer
    EnglishAnalyzer analyzer = new EnglishAnalyzer(new
        CharArraySet(Arrays.asList(stopWordsSplitted),true));

    // Define query parser using analyzer
    QueryParser parser = new QueryParser("contents" , analyzer);
    // Parse query input
    Query query = parser.parse(queryInput);

    // Search results of the query with a maximum of 500
    TopDocs results = indSearcher.search(query, 500);
    // Get score of documents found
    ScoreDoc[] hits = results.scoreDocs;

    // Save total hits made
    int totalHits = Math.toIntExact(results.totalHits.value);
    System.out.println("Found " + totalHits + " documents.");

    // Check if ´theres a hit or more
    if(totalHits>0){
        // Get hits score
```

```
            hits = indSearcher.search(query, totalHits, Sort.RELEVANCE).scoreDocs;
            System.out.println(totalHits);
            System.out.println(Arrays.toString(hits));
            // Add documents to results
            int i = 0;
            // For each hit add it to results
            for(ScoreDoc hit : hits){
                // Get document
                Document doc = indSearcher.doc(hit.doc);
                // Get path of the document
                String path = doc.get("path");
                // Add to results
                documents.add(doc);

                i++;
            }
        }
        else{
            // If there isnt hits
            System.out.println("There ´arent any documents that fits the query.");
        }
        // Close reader
        indReader.close();
        // Return results
        return documents;
    }
```

1.2.3. GUI

La interfaz se ha definido utilizando un objeto *JFrame* de Java el cual nos permite definir los distintos elementos de la misma, utilizando para ello la interfaz de NetBeans. Los elementos que se han definido son:

- *JLabel*: Para las etiquetas.
- *JTextField*: Para los campos a introducir.
- *JButton*: Para las acciones de indexar y buscar.
- *JScrollPane*: Para los paneles de la lista de documentos y previsualización del contenido del documento.
- *JTextArea*: Para la previsualización del contenido del documento.
- *JList*: Para la lista de documentos.

Además de los elementos mencionados previamente, se han definido las acciones de los botones como *Index*:

```
private void IndexButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // Get text from field for query
        String query = FolderField.getText();
        try {
            // Index documents
            indexer.Index("indexes/", "documents/" +
                query,"documents/english_stopwords.txt");
            // If there are any hit
            // Indicate there are results
            ResultsLabel.setForeground(Color.orange);
            ResultsLabel.setText("Documents indexed.");

        } catch (IOException ex) {
            Logger.getLogger(GUI.class.getName()).log(Level.SEVERE, null, ex);
        } catch (Exception ex){
            ResultsLabel.setForeground(Color.red);
            ResultsLabel.setText(ex.toString().substring(ex.toString().lastIndexOf("\\")
                + 1));
        }
    }
```

Y el botón *Search*:

```
private void SearchButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // Get text from field for query
        String query = TermField.getText();
        // Clear document content
        DocumentTextArea.removeAll();
        // Define list of results
        ArrayList<Document> results = new ArrayList<Document>();
        try {
            // Search with Search Engine
            results = engine.search("indexes/", "documents/english_stopwords.txt"
                , query);
        } catch (IOException ex) {

        } catch (ParseException ex) {

        }

        // Check if theres no hits
        if(results.isEmpty()){
            // Clear list
            lista.removeAllElements();
            // Clear document content
            docContent.removeAll(docContent);
            // Indicate there are no results
            ResultsLabel.setForeground(Color.red);
            ResultsLabel.setText("Found 0 documents.");
        }else{
            // If there are any hit
            // Indicate there are results
            ResultsLabel.setForeground(Color.orange);
            if(results.size() == 1){
```

```
            ResultsLabel.setText("Found 1 document.");
        }else{
            ResultsLabel.setText("Found " + results.size() + " documents.");
        }
        // Clear list and document content
        if(!lista.isEmpty()){
            lista.removeAllElements();
            docContent.removeAll(docContent);
        }
        // Set documents paths that satisfies the query
        for(Document doc:results){
            String nombre = doc.get("path");
            lista.addElement(nombre);
            docContent.add(doc.get("contents"));
        }
    }
}
```

Y un evento *OnClickListener* sobre la lista de documentos encontrados:

```
private void ListResultOnClickAction(javax.swing.event.ListSelectionEvent evt) {
        // Get selected index from list of files
        int selectedIndex = FilesList.getSelectedIndex();
        // Clear previous document content
        DocumentTextArea.removeAll();
        if(selectedIndex>=0 && selectedIndex<lista.size()){
            // Write document content
            DocumentTextArea.setText(docContent.get(selectedIndex));
        }
    }
```

1.3. Innovaciones

En cuanto a las mejoras o innovaciones respecto al código de ejemplo propuesto, he realizado las siguientes en el Indexador:

- Para la lectura de las palabras vacías, he necesitado utilizar la librería *FileUtils* de Apache, para poder transformar el fichero en un único objeto String, posteriormente dicho objeto ha sido procesado para separar las diferentes palabras e incluirlas en un Array de Strings y poder convertir dicho Array en una Lista que exige el analizador de Lucene.
- También he utilizado la librería *Files* propia de Java para poder no solo para comprobar y abrir directorios, sino también para poder recorrerlos utilizando la función *walkFileTree* y *visitFile* que permiten recorrer de forma iterativa los documentos, facilitando su indexación con su posterior procesamiento de forma individual.
- Para la lectura del fichero he encontrado problemas a la hora de conseguir leer el documento, cosa que he podido resolver utilizando un objeto *BufferedReader* y un objeto *StringBuilder*, que permitía separar el contenido por líneas y conservar el formato original.

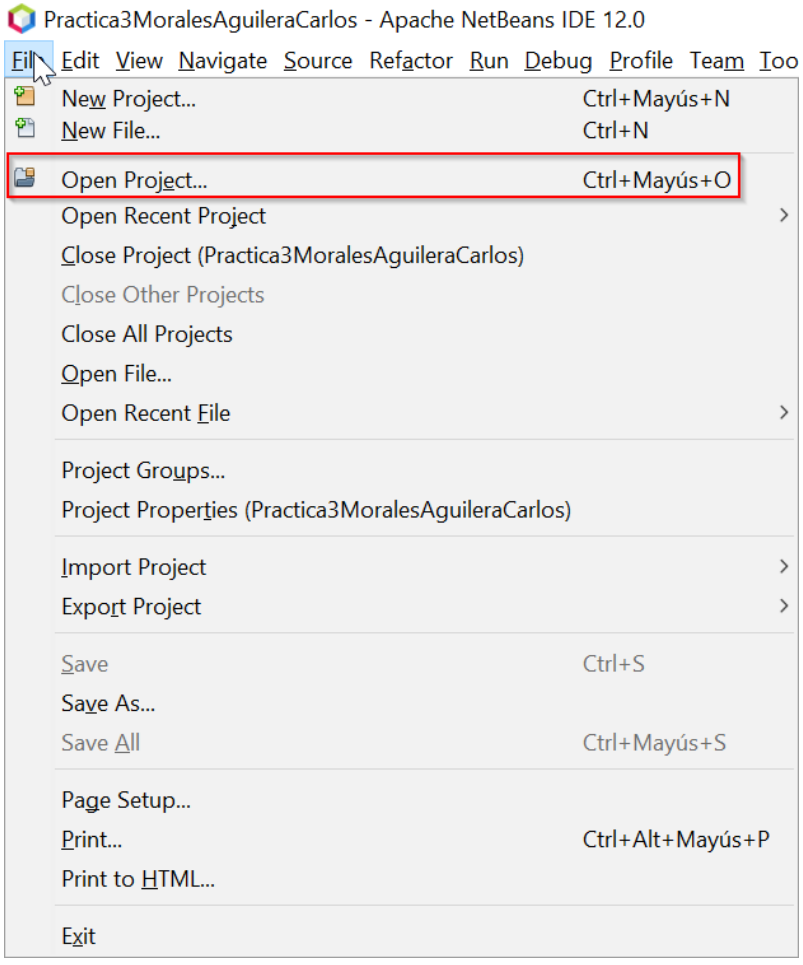
Por otro lado, en cuanto a la parte del Motor de Búsqueda:

- La mayoría de referencias que he encontrado sobre el número total de aciertos *totalHits* eran antiguas, y hasta que he conseguido obtener dicho elemento *totalHits.value*.
- He ordenado los resultados encontrados por la relevancia obtenida por el elemento *score* con *Sort.RELEVANCE*.

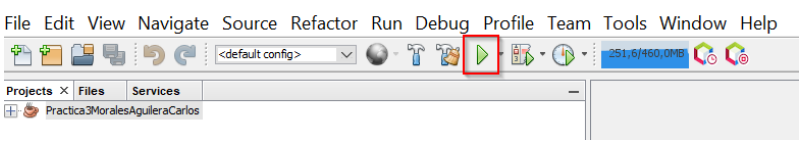
1.4. Manual de uso

1.4.1. Importación del proyecto

Para ejecutar la práctica, simplemente se lanza la interfaz gráfica, ejecutándose desde NetBeans. Para ello es necesario importar el proyecto, lo cual se puede realizar fácilmente descargando NetBeans y seleccionando abrir proyecto:

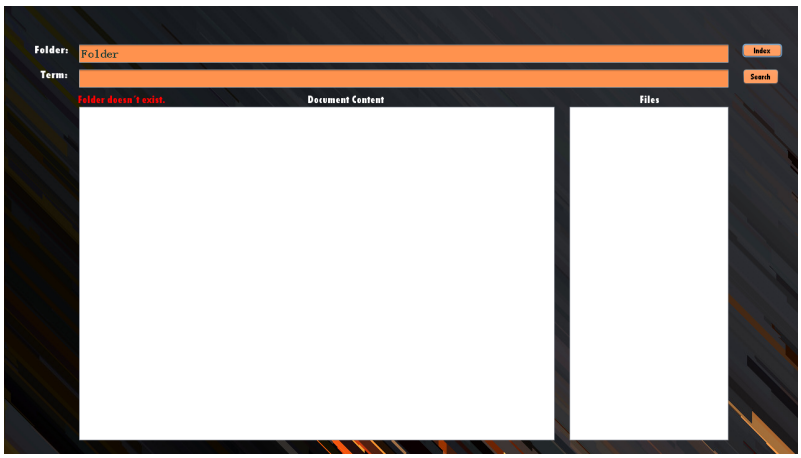


A continuación indicamos la carpeta del proyecto y se realizará la importación, para ejecutar basta con indicar el botón de ejecución:

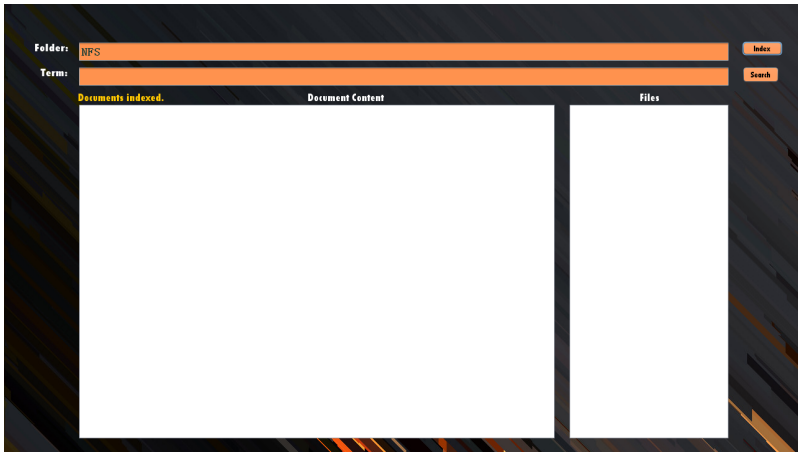


1.4.2. Indexación

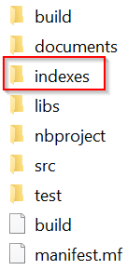
Para la indexación, habrá que indicar el nombre de la carpeta de documentos que se pretende indexar (previamente situada en la carpeta *documents* del proyecto). En caso de no existir dicha carpeta se nos indica:



En el caso de que efectivamente todo esté correcto nos notifica que se ha realizado la indexación:

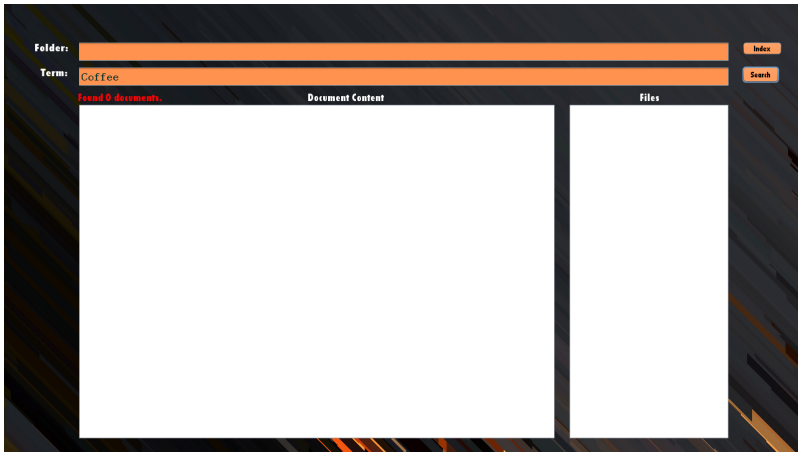


Además se puede comprobar que se ha generado la carpeta *indexes* que posee los indices:

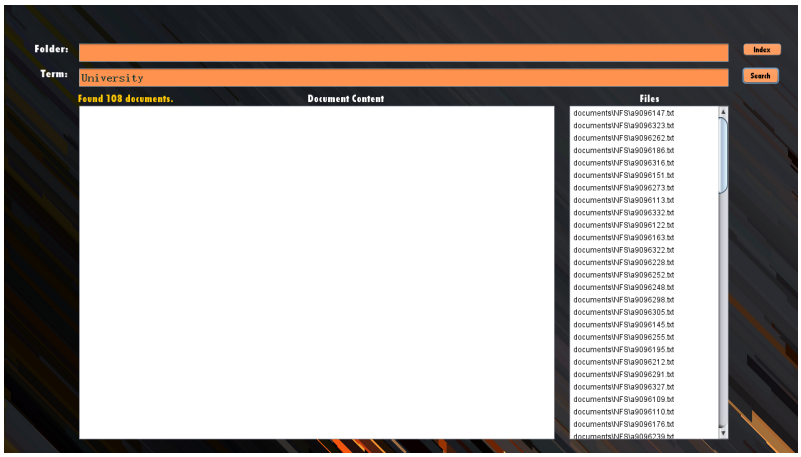


1.4.3. Consulta

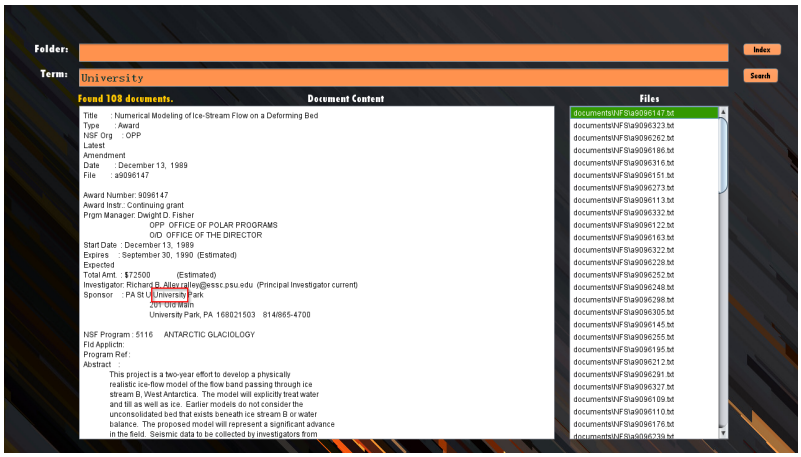
Para la realización de una consulta tenemos la barra de términos donde se indica un término, en caso de no encontrar dicho término:



En caso de encontrar un término, se puede observar en la columna de la derecha en la que aparecen los documentos que incluyen dicho término:



A continuación si pulsamos sobre uno de los documentos, podemos ver una previsualización del mismo en la parte izquierda de la pantalla, y podemos ver que efectivamente contiene el término:



2. Referencias Bibliográficas

[1] Lucene. Documentación oficial de la versión 8.8.1. Link Lucene.

[2] HowToDoInJava. Lucene Tutorial - Index and Search. Link HowToDoInJava.

[3] TutorialsPoint. Lucene - Sorting. Link TutorialsPoint.

[4] Java Code Geeks. Lucene Indexing Example. Link Java Code Geeks.

[5] Lucene. Ejemplo oficial Indexación versión 8.8.1. Link Lucene.

[6] Lucene. Ejemplo oficial Búsqueda versión 8.8.1. Link Lucene.

3. Colección de documentos

La colección de documentos escogida pertenece a la colección ***NSF Research Awards Abstracts 1990-2003*** (Link), de la cual se han extraído únicamente los premios obtenidos en 1990 para reducir el tamaño a un total de 219 documentos de texto.

Como no ocupa mucho espacio, he decidido finalmente incluirlo dentro de la entrega, aunque también se puede encontrar en este directorio de *Drive* (Link) si se desea descargar nuevamente.