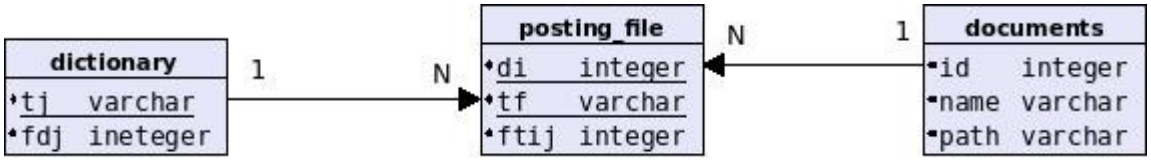
Práctica de almacenamiento y recuperación de la información. Curso 2011-2012.

Jesús Manuel Muñoz Mazuecos.

Introducción.
Se ha desarrollado un buscador basado en el modelo vectorial para la práctica de almacenamiento y recuperación de la información. El lenguaje de implementación elegido es python, la razón viene porque python ofrece un soporte completo para manejo de expresiones regulares y tablas hash (Diccionarios en python) de una manera bastante amigable. El sistema dispone de un *indexador* que se encarga de la persistencia (*MySQL),* por otra parte la aplicación dispone de una interfaz gráfica que hace funcionar el sistema.

Persistencia.
Para gestionar la persistencia el sistema se apoya en una base de datos *MySQL* para recuperar información relacionada con las estadísticas de cada documento.



Estructura del sistema.
El sistema está compuesto por una clase python y el programa principal.

- Clase *indexer.py*: esta clase proporciona métodos para gestionar la persistencia. Entre estos métodos están:
    - Añadir un nuevo documento al sistema.
    - Restaurar el contenido de la base de datos con los datos de inicio.
    - Devolver la lista completa de documentos.

- Clase, *Buscardor.py:* esta clase se encarga de construir el modelo vectorial ante una pregunta sobre el sistema. Se construye un vector para cada documento, y para cada uno hace la función coseno respecto del vector pregunta. Posteriormente se genera un fichero xml en donde podemos navegar a los documentos recuperados. Esta clase también dispone de funcionalidad para buscar documentos similares a uno dado.

- Clase *XMLDocument.py*: esta clase se encarga de construir un documento xml a partir de una lista de documentos recuperados.

- Programa principal, *GUI.py*: este programa es el encargado de levantar la interfaz gráfica, y de hacer funcionar el sistema. Para ello crea instancias de las clases mencionadas anteriormente, apoyándose en los métodos que ofrecen las mismas.

Fundamentos principales de diseño.
En esta sección se van describir los fundamentos de diseño de las partes más importantes de nuestro sistema. Fundamentalmente el diseño se basa en diccionarios de python (*Tablas hash)*

1. **Función para añadir un nuevo documento.**
   Esta funcionalidad está implementada dentro de la clase *indexer.py,* puede ser utilizada por el usuario para añadir nuevo contenido, o por el sistema cuando la base de datos está vacía. A partir del siguiente pseudocódigo entenderemos la manera operar del sistema para añadir contenido nuevo.

```
Entrada:
   NombreDoc : cadena
   Path : cadena

Variables:
   listaPalabras : lista
   idDoc : integer

Inicio:

Si(esValido(documento, path)):

   insertarEnDocumentos(documento, path)
   idDoc = getIDUltimoDocuemto()
   doc = abrirFichero(path + documento)

   Para cada linea en doc:

      s = minusculas(linea)
      listaPalabrasLinea = SepararPor((Blancos | símbolos)*)
      listaPalabras = listaPalabras + listaPalabrasLinea

   conjuntoPalabras = listaToConjunto(listaPalabras)
   conjuntoPalabras = ConjuntoPalabras – (stopList U cadVacía)

   Para cada palabra en ConjuntoPalabras:

      n = contarOcurrencias(palabra, listaPalabras)
      ActualizarDiccionario(palabra)
      InsertarEnPosting_file(idDoc, palabra, n)
Sino:
      LanzarNuevaExcepción(error.mensaje())
Fin.
```

2. **Función de búsqueda.**
   Esta funcionalidad está implementada dentro de la clase buscador.py, se encarga de devolver una lista de documentos ante una pregunta introducida por el usuario. A partir del siguiente pseudocódigo entenderemos cómo se computa una búsqueda.

```
Entrada:
      Pregunta : cadena
Variables:
      d : integer
      vectores : diccionarioPython //(de diccionarios)
      dicFdjs : diccionarioPython
      dicAngulos : diccionarioPython
      vectorPregunta : diccionariosPython
      CjtoPalabras : conjunto
      listaDocs : lista
Inicio:
d = getD()
CjtoPalabras, vectorPregunta, dicFdjs = DoPregunta(pregunta, d)

// CjtoPalabras → Palabras contenidas en el vector pregunta
// vectorPregunta → vector de pesos (diccionario).
// dicFdjs → Tabla hash con el numero de documentos por término

listaDocs = getDocs()
Para cada doc in listaDocs:

      // dicfrecuencias: palabra → frecuencias para documento doc
      dicFrecuencias = Frecuencias(doc)

      //diccionario vacío con todos los términos de la colección
      //inicializando todos los pesos a cero.
      vectorPesos = vectorVacio()

      Para cada clave en dicFrecuencias.claves()
            peso = getPeso(dicFrecuencias[palabra], d,
                           dicFdjs[palabra])
            vectorPesos[palabra] = peso
      vectores[doc] = vectorPesos

Para cada clave en vectores.claves(): //clave → IdDoc

      escalar = producto(vectores[clave], vectorPregunta)
      modPreg = modulo(vectorPregunta)
      modVector = modulo(vectores[claves]
      dicAngulos[clave] = arcos(escalar / (modPreg * modVector)

listaRecuperados = dicToSortedList(diccnarioAngulos, orderBy(alpha))
// listaRecuperados(elem) : tupla(idDoc, Nombre, path, angulo)
XML = getXML(listaRecuperados)
f = abrir('resultados.xml', 'escritura')
escribir(f, XML)
cerrar(f)
```

Pruebas y estadísticas.
En esta sección adjuntaremos información sobre el tiempo de ejecución y estadísticas del sistema.

- **Pregunta: "**operating 3, system 2, http".
- **Tiempo de ejecución:** 78.156 segundos.
- **Documentos Recuperados:** 236**.**
- **Similares de: "**4mb-Laptops.txt".
- **Tiempo de ejecución**: 88.920 segundos.
- **Documentos Recuperados:** 237.

Ejecución.
El sistema dispone de una interfaz gráfica que se ejecuta desde la línea de comandos. Para ejecutar la aplicación nos situamos bajo el directorio raíz de la aplicación mediante la terminal de comandos y escribimos:

- **`$ python GUI.py`**

Una vez lanzado ese comando se levantará la interfaz gráfica de la aplicación que resulta bastante intuitiva. Sólo resaltar que si queremos introducir un documento y no especificamos su path, el sistema lo buscará bajo el directorio por defecto***`./Doc`***.
Para formular una pregunta al sistema se especificar cada término seguido de su peso según el siguiente formato:

- **Pregunta:** `<término> [<peso>], <término> [<peso>], <término> [<peso>]`

Para un correcto funcionamiento del sistema debemos tener instalados los siguientes programas instalados:

- El visor de documentos ***gedit***.
- El explorador web ***firefox***.