Tarea 1 CC52D - Recuperación de Información

Ricardo Baeza Yates y Georges Dupret
Semestre 2005/2

1. Objetivo

El objetivo de esta tarea es construir un buscador sobre una base de datos de documentos XML.

2. Colección

La colección de documentos a usar es la colección CFC (documentos sobre *fibrosis cística*) en formato XML. Esta colección está disponible en`sunsite.dcc.uchile.cl/irbook`. A continuación se presenta un ejemplo del contenido de la colección:

```
<RECORD>
<PAPERNUM>PN79004</PAPERNUM>
<CITATIONS> ...  </CITATIONS>
<RECORDNUM>00984 </RECORDNUM>
<MEDLINENUM>79121155</MEDLINENUM>
<AUTHORS>
<AUTHOR>Berg-N-O</AUTHOR><AUTHOR>Dahlqvist-A</AUTHOR><AUTHOR>Lindberg-T</AUTHOR>
</AUTHORS>
<TITLE>Exocrine pancreatic insufficiency, small intestinal dysfunction and
protein intolerance.  A chance occurrence or a connection?.</TITLE>
<SOURCE>Acta-Paediatr-Scand. 1979 Mar. 68(2). P 275-6.</SOURCE>
<MAJORSUBJ> <TOPIC>CELIAC-DISEASE: co</TOPIC><TOPIC>CYSTIC-FIBROSIS: co</TOPIC></MAJORSUBJ>
<MINORSUBJ><TOPIC>CASE-REPORT</TOPIC><TOPIC>CHILD</TOPIC><TOPIC>DIPEPTIDASES: me</TOPIC>
<TOPIC>DISACCHARIDASES: me</TOPIC><TOPIC>FEMALE</TOPIC><TOPIC>GLUTEN: du</TOPIC>
<TOPIC>HUMAN</TOPIC><TOPIC>INTESTINAL-MUCOSA: en, pa</TOPIC>
<TOPIC>INTESTINE-SMALL: en, pa</TOPIC><TOPIC>MALE</TOPIC></MINORSUBJ>
<ABSTRACT>The coexistence of cystic fibrosis (CF) and coeliac disease (CD) is
reported in eight children, the odds against this occurring in the
same child being 1:2 million to 1:5.9 million. ...</ABSTRACT>
<REFERENCES> ...  </REFERENCES>
</RECORD>
```

La colección está contenida en una serie de archivos llamados `cf*.xml`. El número del archivo no es relevante, pues consideraremos que todos los `cf*.xml` son en realidad fragmentos de un mismo archivo.

Para nosotros, un **documento** será un **RECORD** en alguno de los archivos XML.

En todos los casos, el formato de entrada y salida de cada programa ha sido cuidadosamente especificado para hacer más precisa la corrección.

3. Programa indexador: 'index'

El indexador es usado entregando como parámetro los nombres de los archivos de salida y los de entrada, ejemplo:

```
% index xmldb.lst stopwords.lst words.voc tags.voc index.idx
```

3.1. Archivos

El archivo `xmldb.lst` es un archivo de entrada. Contiene los nombres de los archivos de la colección XML, uno por línea.

El archivo `stopwords.lst` es un archivo de entrada. Contiene todas las palabras que **no** deben ser indexadas por ser demasiado comunes y/o no aportar información relevante. Una primera aproximación la encontrarán en el mismo lugar de la colección en el archivo `frequent_english_words.txt`:

```
the
at
...
for
```

Se les recomienda copiar este archivo a su cuenta y aumentarlo usando palabras de la colección o stopwords del inglés que encuentren en otras fuentes.

El archivo `words.voc` es un archivo de salida. Contendrá todas las palabras que se encuentren en los archivos de la colección y que no sean *stopwords*.

Pueden escoger la definición de palabra que les parezca más apropiada, lo normal es cualquier cadena que contenga caracteres imprimibles que no sean espacios.

El formato de salida para `words.voc` es el siguiente:

- Una línea por palabra.
- En cada línea, `palabra,identificador,frecuencia`.

donde `palabra` es alguna palabra que aparece en la colección, `identificador` es un número único que identifica a esa palabra y `frecuencia` es el número de veces que aparece la palabra en la colección, por ejemplo:

```
analysis,243,3258
amalgamate,452,1567
...
xylometric,1690,924
zygoxis,28,4
```

Significa que "analysis"tiene el identificador 243, y aparece 3.258 veces en la colección completa. Las palabras en este archivo deben estar en orden alfabético.

El archivo `tags.voc` es un archivo de salida. Contendrá todas las marcas que se encuentran en los archivos de la colección.

El formato de salida para `tags.voc` es el siguiente:

- Un tag por línea.
- En cada línea, `tag, identificador,frecuencia`.

.

Los tags en este archivo deben estar en orden alfabético. La frecuencia de aparición de cada marca no la utilizaremos por ahora:

```
author,13,1451
authors,26,354
...
references,28,354
topic,12,497
```

Noten que en los archivos de vocabulario, se convierten las palabras a minúsculas. No consideraremos mayúsculas/minúsculas en esta tarea.

El archivo `index.idx` es un archivo de salida. Es el más importante, y contendrá un **índice invertido**. El formato es:

- Una palabra por línea.
- En cada línea, el identificador de palabra, seguido por tríos de números. Cada trío contiene el identificador del documento, del tag y la frecuencia.

Por ejemplo, la siguiente línea:

```
...
28,1,12,1,2,4,3
...
```

Significa: la palabra número `28` (*zygoxis* en nuestro ejemplo), aparece en el documento 1, dentro del tag 12 (*topic*) 1 vez, y en el documento 2, dentro del tag 4, 3 veces. Noten que la suma $1 + 3 = 4$ es coincidente con `words.voc` que dice que *zygoxis* aparece 4 veces en la colección.

El programa debe asignar los `docid` en orden correlativo, comenzando desde 1, **sin considerar** los `recordnum` de los documentos originales.

3.2. Estructuras de datos

Necesitarán una tabla de hashing que asocie cada palabra con un `wordid` y su frecuencia y otra tabla de hashing que asocie cada tag con un `tagid` y su frecuencia.

Necesitarán un índice que tenga para cada `wordid` un puntero a una lista de `hits`. Un `hit` será un trio: `docid,tagid,frecuencia`.

Pueden usar las estructuras de datos provistas por los lenguajes: en C una biblioteca de estructuras de datos, en C++ pueden usar las tablas de hashing de la STL, en Java las clases base, en Perl o Python los módulos estándar. El lenguaje de programación es libre, pero se considerará en la evaluación la eficiencia de su solución.

3.3. Sugerencias

Utilicen una biblioteca XML (ojalá un SAX parser) para leer los archivos de datos. Esta biblioteca genera un evento cada vez que lee un tag o un texto.

Realicen las optimizaciones de código al final, pero cuiden de que los algoritmos base sean los apropiados para el volumen de documentos a considerar desde el principio.

4. Programa Buscador: 'search'

El programa buscador es invocado de la siguiente manera:

```
% search query.lst stopwords.lst words.voc tags.voc index.idx
```

donde la consultas están en el archivo `query.lst`. Cada consulta está en una línea y es una lista de palabras. La respuesta deben ser todos los documentos que tienen algunas de las palabras (consulta OR). Todos los otros archivos son los mismos anteriores, pero son de entrada.

La salida estándar deben ser dos líneas por consulta, separadas por una línea vacía, conteniendo:

```
query: children diseases
answer: 12 9* 678 46
```

Esto significa que la consulta aparece en los documentos 12, 9, 678 y 46. Las consultas pueden estar compuestas de más de una palabra, en cuyo caso se debe indicar cuáles documentos contienen **todos** los términos buscados marcándolos con un * (9 en el ejemplo). Si la consulta no retorna nada, la lista de la segunda línea queda vacía.

4.1. Archivos

Utilice los mismos archivos que la sección anterior. La idea es cargar los archivos en memoria al iniciar el programa, re-construir las estructuras de datos del programa anterior y luego hacer las consultas con los archivos ya cargados.

5. Nota Importante

En esta tarea no abordaremos el problema de ranking de los documentos encontrados, da lo mismo el orden en que sean entregados; tampoco consideraremos el tag en que ocurre el calce. Sin embargo esto se considerará en la segunda tarea, asi que diseñe sus algoritmos pensando en que después usará el modelo vectorial y dentro de que tag se encuentra la palabra.

6. Como Entregar la Tarea

El código debe estar adecuadamente comentado, no explicando qué hace cada línea de código, pero sí **cómo funciona** y qué algoritmo usan. La estructura de directorios a utilizar es la siguiente:

TAREA1CC52D/README.txt	Su nombre e instrucciones para correr la tarea
TAREA1CC52D/src	Código fuente de los programas que entrega
TAREA1CC52D/bin	Programas `index` y `search`
TAREA1CC52D/doc	Documentos que quiere que leamos sobre su tarea, opcional
TAREA1CC52D/etc	Todo lo demás, este directorio no será revisado, opcional

La tarea debe ser entregada instalada y funcionando antes del día Lunes 10 de Octubre del 2005.

La entrega se realiza en "clotaire.dcc.uchile.cl". La máquina usa Linux. Para tener acceso, por favor enviar un e-mail **DESDE SU CUENTA EN ANAKENA O DICHATO** con el subject "CC52D ACCESO" a: `gdupret@dcc.uchile.cl`.

En clotaire, ejecute:

```
/usr/local/bin/cc52d enunciado 1
```

Esto instalará un directorio `TAREA1CC52D` en su cuenta con los directorios y archivos base para la tarea, puede copiar este directorio a otra máquina y luego copiarlo de vuelta para la entrega.

Para entregar la tarea, utilice:

```
/usr/local/bin/cc52d entrega 1
```

lo que copiará su directorio `TAREA1CC52D` a un directorio de tareas entregadas.