Organización de Datos – Curso Servetto

Recuperación de Textos

Construcción de Índices

Los pasos que se presentan a continuación deben completarse para cada término (o secuencia de hasta cinco términos):

1. **Eliminación de signos ortográficos (acentos y diéresis) y sustitución de mayúsculas por minúsculas *(case folding)***
2. **Eliminación de palabras comunes *(stop words)***
   Hay clases de palabras cuyas funciones sintácticas son las de complementar el significado, relacionar o sustituir a otras y que no tienen valor o significado por sí solas y que nunca serían objeto de búsqueda; también hay palabras que comúnmente aparecen en locuciones (grupos de palabras que forman sentido), como en el caso de algunos conectores de coordinación (p.e. "en *consecuencia*", "de cualquier *manera*", "en *caso* de que"), que en ese contexto no tienen significado relevante aunque individualmente fuera de ese contexto sí, por lo que debe considerarse la detección no sólo de palabras individuales sino de secuencias de palabras, considerando todos los separadores posibles como espacios, saltos de línea y tabulaciones.

   Se debe disponer en un archivo de texto la **lista de palabras a excluir** (si una palabra aparece prefijada con un asterisco representa todas las palabras que terminen con lo que sigue al asterisco, p.e. *mente), y en otro, la **lista de locuciones**, ambas ordenadas alfabéticamente. Para construir un índice ambos archivos deben cargarse en memoria, en estructuras que permitan búsquedas ágiles.

   Las palabras que se reconozcan de un texto y se transformen según el paso 1 deben "entubarse" en una estructura con capacidad máxima para cinco, que permita reconocer locuciones. Cuando se intente agregar una palabra al **"tubo"** y éste se encuentre lleno, se verifica si a partir de la primer palabra no se encuentra formada ninguna locución de la lista de locuciones; de ser así, se extraen todas las palabras de la locución del tubo y se descartan; en caso contrario se saca la primer palabra del tubo y se la busca en la lista de palabras irrelevantes; en caso de ser una palabra irrelevante se la descarta.
3. **Sustitución de palabras por raíces *(steming)***

   Este mecanismo se debe aplicar para generalizar formas conjugadas (tiempo, modo, aspecto, número y persona) de verbos regulares.

   Se debe disponer en un archivo de texto la **lista de terminaciones a podar** de cualquier término que no se descarte al salir del tubo, y que para construir un índice debe cargarse en memoria igual que las listas previas.

   Para todo término que no se descarte del tubo debe determinarse si no termina con alguna de las terminaciones de la lista de steming. Si así fuera se sustituye la terminación por un carácter de control y el término resultante se pasa al índice con los datos que se manejen acerca de su situación en el documento para actualizar la lista correspondiente a ese término en el índice o crear una nuevo registro según corresponda.

Organización del Índice

La organización más general para un índice invertido, y que permite actualizaciones (indexar

nuevos documentos o desindexar existentes) sin necesidad de reconstrucción, es la de **archivo B#** (B+ con restricciones de mantener todos sus nodos excepto el raíz con al menos 2/3 de su capacidad ocupada), con nodos cuyo tamaño sea coincidente con, o múltiplo del tamaño típico de las unidades de asignación o bloques de los sistemas de archivo (4 u 8 Kb). Durante el proceso de construcción del índice, la frecuencia de acceso a las hojas para incorporar o buscar términos garantiza que una gran proporción de ellas, sino todas, permanezcan en RAM gracias al buffering del sistema operativo.

Cuando se llene un nodo, se explora un hermano adyacente para balancear cargas; si el hermano está lleno se agrega un nodo entre medio de ambos y se distribuye las cargas entre los tres.

Compresión de Términos

Para comprimir el índice, los términos de las hojas se organizan con **compresión frontal** *(front coding)*: cada término, excepto el primero de cada hoja, se almacena como una tripla (cantidad de caracteres iniciales coincidentes con el término anterior, cantidad de caracteres almacenados, terminación del término). Por ejemplo la secuencia:

> codazo, codearse, codera, codicia, codiciar, codiciosa, codicioso, codificar, codigo

quedaría comprimida:

> 6|codazo|3|5|earse|4|2|ra|3|4|icia|7|1|r|6|3|osa|8|1|o|4|5|ficar|4|2|go

Entre término y término se puede almacenar la lista invertida o un número de bloque donde encontrar la lista. Las barras verticales están al sólo efecto de permitir una mejor visualización de los datos (no se almacenan).

Como el tamaño real de los nodos es bastante grande, en cada uno presumiblemente se tendrá un buen número de términos. La búsqueda de un término en una secuencia comprimida implica el recorrido secuencial con descompresión, y si la secuencia es muy larga el costo de la descompresión hasta encontrar el término que se busca resulta para nada despreciable.

Para optimizar las búsquedas se puede resignar la compresión de un término de tanto en tanto, de manera que el recorrido de las búsquedas se pueda hacer en primera instancia pasando por los términos sin comprimir, hasta encontrar la subsecuencia donde encontrar el término que se busca. El tamaño ideal de las subsecuencias es la raíz cuadrada del tamaño de la secuencia total:

> 6|codazo|3|5|earse|4|2|ra|<p1>|7|codicia|7|1|r|6|3|osa|<p2>|9|codicioso|4|5|ficar|4|2|go|<p3>|9|3

Las búsquedas se comienzan desde la derecha, donde los primeros campos de control, de derecha izquierda, indican 3 la cantidad de secciones o subsecuencias de términos, 9 la cantidad total de términos, y <p3> la posición relativa de <p2>...

Construcción de Listas Invertidas

Dependiendo del modelo de recuperación que se implemente, las listas invertidas pueden almacenarse junto con los términos o en bloques aparte (organizadas como registros de longitud variable). Si no se proveen formas de consulta que requieran exploraciones exhaustivas de términos, puede considerarse almacenar las listas junto con los términos. En cualquier caso, la actualización de una lista durante su construcción cada vez que se encuentra un término relevante en un documento implica

- Agregar el número de documento al final de la lista correspondiente al término, o agregar el término e inaugurar una nueva lista, en el modelo booleano:
  - $t \rightarrow d_1, d_2, \ldots d_n, \mathbf{d_{n+1}}$
  - $\mathbf{t \rightarrow d_1}$

- Hacer lo mismo o incrementar el contador de apariciones (la frecuencia) del término en un documento de la lista y eventualmente reubicar el par formado por el número de documento y el contador de apariciones para mantener el orden decreciente de apariciones, en el modelo vectorial:
  - $t \rightarrow (d_1, a_1), (d_2, a_2), \ldots (d_k, a_k), \ldots (d_n, a_n), \mathbf{(d_{n+1}, 1)}$
  - $\mathbf{t \rightarrow (d_1, 1)}$
  - $t \rightarrow (d_1, a_1), (d_2, a_2), \ldots (d_k, a_k\mathbf{+1}), \ldots (d_n, a_n)$ con $(ai > ai+1)$ o $(a_i = a_{i+1}$ y $d_i < d_{i+1})$
- Y para el modelo de consultas de frases o términos próximos:
  - $t \rightarrow (d_1, (p_{11}, \ldots p_{1a})), (d_2, (p_{21}, \ldots p_{1b})), \ldots (d_n, (p_{n1}, \ldots p_{1x})), \mathbf{(d_{n+1}, (p_{(n+1)1}))}$
  - $\mathbf{t \rightarrow (d_1, (p_{11}))}$
  - $t \rightarrow (d_1, (p_{11}, \ldots p_{1a})), (d_2, (p_{21}, \ldots p_{1b})), \ldots (d_k, (p_{k1}, \ldots p_{kh}, \mathbf{p_{k(h+1)}})), \ldots (d_n, (p_{n1}, \ldots p_{1x}))$

Ya sea que las listas se almacenen junto con los términos en las hojas o que se almacene en bloques aparte, toda actualización en el proceso de indexación implica la escritura de un bloque de tamaño considerable, para consolidar un cambio mínimo, y por cada término relevante que se detecte en un documento. Además, como se verá luego, las listas se almacenan usando técnicas de compresión que hacen el proceso de actualización más costoso en tiempo.

Para evitar este requerimiento constante de escrituras en posiciones aleatorias del archivo, se puede diferir la construcción de las listas de la siguiente manera:

- Cada vez que aparece un término nuevo **t** en el vocabulario del índice, se lo incorpora junto con un número relativo **n** que denote su orden de aparición: **(t, n)**
- Todo registro de aparición de un término en un documento se efectúa en un archivo secuencial de trabajo:
  - Modelos booleano y vectorial: **(n, d)** con d número de documento
  - Consultas de frases o términos próximos: **(n, d, p)** con p posición del término número n en el documento número d

De esta manera el archivo del índice sólo se actualiza cada vez que aparezca un término nuevo.

Al finalizar el proceso de indexación se realizan los siguientes pasos:

1. Se exportan los registros del índice (t, n), que quedan ordenados por t, a un archivo secuencial de trabajo
2. Se ordena con el método de ordenación externa el archivo de términos por el número de término n.
3. Se ordena con el método de ordenación externa el archivo de aparición de términos por número de término n y número de documento d (y además por número de posición p, para el índice de consulta de frases o términos próximos).
4. Se recorre secuencialmente el archivo ordenado de aparición de términos en coordinación con el de términos del paso 2 para construir las listas invertidas por cortes de control, y almacenarlas en el índice en lugar del número de término o en un bloque aparte sustituyendo el número de término por el número de bloque donde quedó almacenada la lista.

Los pasos 1 y 2 pueden evitarse si al tiempo que se graban los registros de aparición de términos en documentos, también se graba en otro archivo secuencial los términos que se van agregando al índice; de esta forma se obtiene el mismo resultado del paso 2.

Caso de Ejemplo

Nro. Documento	Texto del Documento
1	Las Cosas de la Vida
2	La Vida es Bella
3	Las Cosas del Querer
4	La Vida después de la Vida

Las palabras en gris son irrelevantes *(stopwords)*.

En los desarrollos sucesivos, las listas resultantes se muestran sin campos de control y asociadas a sus respectivos términos para una mejor comprensión. Debe tenerse en cuenta que para su implementación se requiere campos de control iniciales que permitan conocer cantidades de elementos que se repiten y que las listas pueden almacenarse junto con los términos o en bloques aparte.

Modelo Booleano

Archivo de aparición de términos: (1, 1), (2, 1), (2, 2), (3, 2), (1, 3), (4, 3), (2, 4), (2, 4)

1. cosas, vida, bella, querer
2. (1, 1), (1, 3), (2, 1), (2, 2), (2, 4), (2, 4), (3, 2), (4, 3)
3. 1(1, 3)|2(1, 2, 4)|3(2)|4(3) ↓

   (bella, 3(2))(cosas, 1(1, 3))(querer, 4(3))(vida, 2(1, 2, 4))

Modelo Vectorial

Archivo de aparición de términos: (1, 1), (2, 1), (2, 2), (3, 2), (1, 3), (4, 3), (2, 4), (2, 4)

1. cosas, vida, bella, querer
2. (1, 1), (1, 3), (2, 1), (2, 2), (2, 4), (2, 4), (3, 2), (4, 3)
3. 1((1, 1), (3, 1))|2((1, 1), (2, 1), (4, **2**))|3((2, 1))|4((3, 1)) ↓

   (bella(0,6), 3((2, 1)))(cosas(0,3), ((1, 1), (3, 1)))(querer(0,6), 4((3, 1)))
   (vida(0,125), 2((**4, 2**), (1, 1), (2, 1)))

Consultas de Frases o Términos Próximos

Archivo de aparición de términos: (1, 1, 1), (2, 1, 2), (2, 2, 1), (3, 2, 2), (1, 3, 1), (4, 3, 2), (2, 4, 1), (2, 4, 2)

1. cosas, vida, bella, querer
2. (1, 1, 1), (1, 3, 1), (2, 1, 2), (2, 2, 1), (2, 4, 1), (2, 4, 2), (3, 2, 2), (4, 3, 2)
3. 1(1(1), 3(1))|2(1(2), 2(1), 4(**1, 2**))|3(2(2))|4(3(2)) ↓

   (bella, 3(2(2)))(cosas, (1(1), 3(1)))(querer, 4(3(2)))(vida, 2(1(2), 2(1), 4(1, 2)))

Compresión de Listas Invertidas

Para colecciones de documentos muy grandes (miles de documentos), las listas invertidas pueden ser muy extensas, y las búsquedas proporcionalmente costosas por los tiempos de recuperación. Para optimizar tiempos de recuperación y al mismo tiempo reducir tamaños y aumentar chances de

que las listas quepan completas en un solo bloque de organización (haciéndolas más manipulables), se utilizan técnicas de compresión de números.

Una primera aproximación para reducir el tamaño en bytes para la representación de los números es reducir magnitudes: si los números de documento (y de posiciones en la variante para consultas de frases o términos próximos) se ordenan de mayor menor, cada número puede expresarse como la **distancia** al precedente. Esto no obstante no garantiza que todas las magnitudes se reduzcan, porque podrían aparecer términos que sólo se encuentran en los últimos documentos, y el número del primer documento en el que se encuentren no se puede reducir. Se logra tener muchos números de documento expresados con magnitudes pequeñas, pero algunos pueden quedar con magnitudes grandes.

Una segunda aproximación, sobre la base de la primera, es pensar en algún sistema de codificación que utilice cantidades variables de bits para representar números, proporcionales a sus magnitudes.

Codificación Unaria

Es una codificación que representa a cualquier número natural n mediante n-1 unos seguidos de un 0. Por ejemplo, el número 3 seguido del 10 se representa `1101111111110`.

Los dígitos de la codificación se pueden invertir sin pérdida de generalidad; así el 3 seguido del 10 se puede codificar también `0010000000001`.

Esta solución todavía sigue siendo inaceptable, ya que para distancias grandes se necesita muchos bytes: para la lista de distancias `3, 48, 31, 15, 62, 5, 64, 187` se emplearían:

```
3 + 48 + 31 + 15 + 62 + 5 + 64 + 187 = 415 bits
```

O sea 52 bytes, siendo que si se representase cada distancia en complemento a 2 de 2 bytes se necesitarían tan sólo 16 bytes.

Codificación Alineada a Bytes

Es una codificación que representa números naturales con cantidades enteras de bytes, usando los dos bits más significativos del código para indicar la cantidad de bytes empleados:

```
0..2^6-1        →    00xxxxxx
64..2^14-1      →    01xxxxxx xxxxxxxx
2^14..2^22-1    →    10xxxxxx xxxxxxxx xxxxxxxx
2^22..2^30-1    →    11xxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

Por ejemplo, el 3 seguido del 10 se representa `0000001100001010`.
Para el caso de ejemplo sería ya una solución razonable, ya que la lista de distancias `3, 48, 31, 15, 62, 5, 64, 187` quedaría representada en 10 bytes (menos bytes que representando cada distancia en complemento a 2 de dos bytes).

Codificación Gamma de Elias

Es una codificación que representa a un número natural n como:

- Piso(log2(n)) bits en 1, que representan la máxima potencia de 2 que no exceda a n.
- Un 0 como marca de separación.
- Piso(log2(n)) bits para representar n-2^Piso(log2(n)), que es equivalente a la representación binaria de n sin el bit más significativo.

Usa 2*Piso(log2(n))+1 bits para representar el valor n.

n	**Piso(log2(n))**	**γ(n)**

1	0	0
2	1	10 0
3	1	10 1
4	2	110 00
5	2	110 01
6	2	110 10
7	2	110 11
8	3	1110 000
9	3	1110 001
10	3	1110 010
...	...	...
48	5	111110 10000

Para decodificar un código:

- Se cuentan los b bits consecutivos en 1 que prefijan el código hasta encontrar el primer 0
- Se descarta el primer 0
- Se lee el número representado en binario en los próximos b bytes y se le suma 2^b.

Para codificar la lista de distancias del caso de ejemplo: `3, 48, 31, 15, 62, 5, 64, 187`, se requerirían:

```
3 + 11 + 9 + 7 + 11 + 5 + 13 + 15 = 74 bits
```

O sea 10 bytes: igual resultado que usando la codificación anterior.

Codificación Delta de Elias

La codificación de un número natural n se compone de:

- El código gamma de Piso(log2(n))+1 (2*Piso(log2(Piso(log2(n)+1)))+1 bits iniciales).
- Piso(log2(n)) bits finales con la representación binaria de n-2^ Piso(log2(n)) (la representación binaria de n sin el bit más significativo).

Usa 2*Piso(log2(Piso(log2(n)+1)))+1+Piso(log2(n)) bits para representar el valor de n.

n	**Piso(log2(n))+1**	**n-2^Piso(log2(n))**	**δ(n)**
1	1	~~+~~	0
2	2	~~+~~0	100 0
3	2	~~+~~1	100 1
4	3	~~+~~00	101 00
5	3	~~+~~01	101 01
6	3	~~+~~10	101 10
7	3	~~+~~11	101 11
8	4	~~+~~000	11000 000
9	4	~~+~~001	11000 001
10	4	~~+~~010	11000 010

...	...		
48	6	1+10000	11000 10000

Para decodificar un código:

- Se decodifica el número x en Gamma.
- Se obtiene el número y, representado en binario en los x-1 bits siguientes.
- Se calcula 2^(x-1)+y.

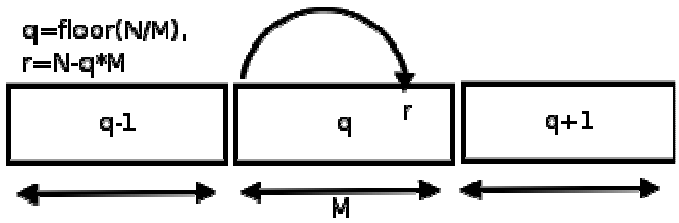
Para codificar la lista de distancias del caso de ejemplo: `3, 48, 31, 15, 62, 5, 64, 187`, se requerirían:

```
4 + 10 + 9 + 8 + 10 + 5 + 11 + 14 = 71
```

O sea 9 bytes: un byte menos que con código gamma.

Codificación de Golomb

Es un esquema de compresión inventado por Solomon W. Golom en los años 60. Usa un parámetro ajustable M para dividir el número N a comprimir en dos partes: el cociente q y el residuo r.



El código final se construye concatenando la codificación de q+1 en unario con la codificación de r en binario, calculando la cantidad de bits para representar r según las siguientes condiciones:

b = Techo(log2(M))

Si r < 2^b-M se representa r en b-1 bits

Si r ≥ 2^b-M se representa 2^b-M+r en b bits

Se puede demostrar que la compresión es óptima cuando M se relaciona con la probabilidad p de que un término se encuentre en un documento de la siguiente manera:

(1-p)^M+(1-p)^(M+1) ≤ 1 < (1-p)^(M-1)+(1-p)^M

M se obtiene redondeando al entero más próximo el valor:

log2(2-p)/(-log2(1-p))

Se calcula p para cada lista de distancias como el cociente entre la cantidad de distancias de la lista (es decir, la cantidad de documentos en los que aparece el término de la lista) y la cantidad total de documentos.

El cálculo de p supone un problema para la actualización del índice: al indexar un nuevo documento habría que recodificar las listas afectadas.
Para codificar la lista de distancias del caso de ejemplo: 3, 48, 31, 15, 62, 5, 64, 187 suponiendo una cantidad total de 500 documentos, se calculan

```
p = 8/500 = 0.016

M = Redondear(log2(2-0.016) / (-log2(1-0.016))) =
  = Redondear(log2(1.984) / (-log2(0.984))) =
  = Redondear(42.47) = 42
```

b = Techo(log2(42)) = Techo(5.39) = **6**

2^b-M = 2^6-42 = **22**

x	**q=Piso(x/42)**	**r=x-q*42**	**unario(q+1)**	**Si(r < 22; 5; 6) bits**	**golomb(x)**
3	0	3	1	5	1 00011
48	1	6	01	5	01 00110
31	0	31	1	6	1 110101
15	0	15	1	5	1 01111
62	1	20	01	5	01 10100
5	0	5	1	5	1 00101
64	1	22	01	6	01 101100
187	4	19	00001	5	00001 10011
				42 bits	

Se requieren 6 bytes (3 bytes menos que con codificación delta).