Recuperación de Textos

Introducción

Los sistemas de recuperación de texto (*Full Text Retrieval Systems*), involucran técnicas de organización eficiente de índices a documentos de texto y de resolución de distintos tipos de consultas sobre esos índices.

En lugar de trabajar con referencias a registros, como en organización de archivos, se trabaja con referencias a documentos (referencias documentales), donde la colección de documentos sería el equivalente de un archivo, con un documento por registro.

Los documentos que se indexan pueden ser libros de una biblioteca, revistas de una hemeroteca, artículos científicos digitalizados, páginas web de Internet, secciones de una enciclopedia digital en CD-ROM, etc.

Las claves de la indexación de textos se denominan “términos”, y su conformación puede variar según el sistema. Por ejemplo, si el sistema es de indexación de libros que no están digitalizados, los términos deben definirse como temas canónicos para la catalogación de los libros, y para cada libro se debe digitalizar en su representación el conjunto de temas que comprende (en bibliotecología este conjunto se denomina *tesauro*); en cambio, si los documentos estuvieran

digitalizados, los términos podrían ser las palabras que contenga.

En cualquier caso, a cada clave o término del índice de un sistema le corresponde una lista de varias referencias a documentos que responden a esa clave: según se vio en organización de archivos, el índice es de clasificación (la lista representa al subconjunto de documentos que contienen el término). Estos índices se denominan "Invertidos" o "De Inversión de Textos", y las listas de referencias "listas invertidas" (esta denominación se debe a que siendo lo normal tener documentos de texto a partir de los cuales se encuentran los términos, los índices nos proporcionan la relación inversa: para cada término distinto nos permiten conocer qué documentos lo contienen).

La indexación de documentos digitalizados usando sus palabras como términos, en sistemas con gran cantidad de documentos, tiene como grandes inconvenientes que la cantidad de términos suele ser tan grande como un diccionario, y que las listas de referencias a documentos correspondientes a cada término también pueden ser muy extensas.

Términos

Cuando se indexan documentos digitalizados el problema de la cantidad de términos se ataca por varios frentes.

En primer lugar, si se elimina la sensibilidad a mayúsculas (*case folding*), se elimina la redundancia de términos por diferencias de grafía. Por ejemplo el término informática podría aparecer hasta cuatro veces en un índice: informática, Informática, INFORMATICA, INFORMÁTICA.

También se puede reducir sensiblemente la cantidad de términos si se reemplazan palabras de una misma familia o de tronco común por un término raíz (*stemming*). Por ejemplo, si tomamos el siguiente refrán de Chesterton como documento:

> "A algunos hombres los disfraces no los disfrazan, sino los revelan. Cada uno se disfraza de aquello que es por dentro."

Vemos que aparecen tres palabras de una misma familia, que podrían sustituirse por un término representante o sustituto de todas ellas, que podría ser "disfrazar". En general, cualquier forma personal de un verbo, e incluso las formas no personales como participios y gerundios, se pueden sustituir por el infinitivo, lo mismo que los sustantivos que tengan un verbo relacionado. También sería factible usar como término sustituto directamente la raíz "disfra".

Y desde luego, hay varias técnicas de enfrentar el problema de la cantidad de términos que tienen que ver con el almacenamiento de los mismos, y que se ven en detalle más adelante.

Referencias

En cuanto a la extensión de las listas de referencias, también se impone buscar artilugios y técnicas que permitan manipular las listas eficientemente.

En primer lugar, hay palabras, como artículos, pronombres, preposiciones, abreviaturas de uso común, etc. *(stop words),* que con seguridad aparecen en todo documento y no son útiles o significativas para realizar consultas; entonces no tiene sentido indexarlas, y el no hacerlo no sólo evita listas que seguramente referirían a todos los documentos, sino también reduce la cantidad de términos o registros del índice.

Las referencias pueden variar según los documentos que se indexen: si se trata de libros de una biblioteca las referencias podrían ser claves primarias de identificación de registros de libros, como un número de ejemplar; si se trata de artículos científicos digitalizados en PDF (*Portable Document Format*), las referencias podrían ser los *paths* de acceso a los documentos; si se trata de páginas web, las referencias serían URL. Para reducir los tamaños de los índices y por consiguiente los

tiempos de acceso, en lugar de almacenar las referencias directamente en el índice, se sustituyen por un número entero que permite su localización en un diccionario de rápido acceso, como podría ser un archivo directo.

Y desde luego, existen técnicas específicas de compresión de números que permiten reducir el tamaño de las listas, y que se ven a continuación.

Compresión de Números de Documento

Supóngase que la organización de un congreso científico decide catalogar los artículos que se presenten según una lista de temas excluyentes, y que a cada artículo se le asigna un número que permite localizarlo en un diccionario (los artículos no se indexan totalmente sino que se catalogan).

Sea por caso el término canónico o maestro

```
Algoritmos y Estructuras de Datos
```

Al cual corresponden los documentos

```
3, 51, 82, 97, 159, 164, 228, 415
```

La búsqueda de artículos en el sistema sólo se puede efectuar combinando lógicamente uno o más términos que se escogen de la lista de catalogación.

¿Cómo se puede almacenar los números de documento en forma compacta?

Ya sea que se diseñe la solución conociendo cantidad de documentos a catalogar o estimando dicha cantidad, en, por ejemplo, quinientos documentos, una primera aproximación podría ser representar los números con la cantidad mínima de bits: la representación de 500 en binario es 111110100, o sea que la cantidad mínima de bits para representar números entre 1 y 500 es 9. La fórmula para calcular la cantidad de bits con la que puede representarse un número decimal entero N en binario es:

```
techo(log2(N)) = techo(log10(N)/log10(2))
```

Para N = 500:

```
techo(log2(500)) = techo(log10(500)/log10(2)) =
= techo(2,698 / 0,301) = techo(8,965) = 9
```

Con esta solución se representan todos los números con la misma cantidad de bits, y la ganancia que se obtiene al achicar la representación podría decirse que se pierde con el procesamiento extra necesario para desempaquetar los números de su formato de representación.

Una mejora al método anterior, para ver si le logra ganar algo, podría ser la siguiente: asumiendo que los números de documento siempre se disponen ordenados de menor a mayor, se puede reemplazar cada número, excepto el primero de la lista, por su distancia del número anterior. Así la lista: `3, 51, 82, 97, 159, 164, 228, 415` se transformaría a:

```
3, 48, 31, 15, 62, 5, 64, 187
```

Con este artilugio se puede reducir más la cantidad de bits para representar los números de documentos, pero ¿si hubiera un término que sólo aparece en el documento número 500? Entonces la cantidad de bits para representar las distancias tendría que seguir siendo la misma que para representar números de documento...

Pero la idea de las distancias puede ser provechosa si se hace el siguiente análisis: al representar números de documentos con distancias relativas, se pude prever que términos que aparezcan en muchos documentos (muy frecuentes) tendrán listas largas de distancias de valores pequeños, y que términos que aparezcan en pocos documentos (poco frecuentes) tendrán listas cortas de distancias

de valores altos; entonces si se codifican las distancias con cantidades variables de bits, usando pocos bits para valores pequeños pero muy frecuentes, y muchos para valores altos pero poco frecuentes, se puede mejorar la relación entre tiempo de procesamiento y tamaño de representación. Estos esquemas de codificación se denominan *entrópicos*, porque la longitud de la codificación de un elemento es proporcional al opuesto del logaritmo de la probabilidad del elemento (los elementos más comunes tienen códigos más cortos).

A continuación se presenta varias técnicas de *compresión de distancias.* Las técnicas pueden ser globales, cuando utilizan el mismo método de compresión para todas las listas, y locales, cuando utilizan un método específico para cada lista.

Codificación Unaria

Es una codificación que representa a cualquier número natural n mediante n-1 ceros seguidos de un 1. Por ejemplo, el número 3 seguido del 10 se representa 0010000000001.

Los dígitos de la codificación se pueden invertir sin pérdida de generalidad; así el 3 seguido del 10 se puede codificar también 1101111111110.

Esta solución todavía sigue siendo inaceptable para el caso de ejemplo, 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

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

- 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:

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

Otra codificación equivalente se obtiene invirtiendo los prefijos, en cuyo caso el significado de las partes queda:

- Piso(log2(n)) bits en 0, tantos cuantos ocupe la representación binaria de n, menos 1.
- Piso(log2(n))+1 bits con la representación binaria de n.

n	**Piso(log2(n))**	**γ(n)**
1	0	1
2	1	0 10
3	1	0 11
4	2	00 100
5	2	00 101
6	2	00 110
7	2	00 111
8	3	000 1000
9	3	000 1001
10	3	000 1010
...	...	...
48	5	00000 110000

Para decodificar un código:

1. Se cuentan los b bits consecutivos en 0 que prefijan el código hasta encontrar un 1

2. Se lee el número representado en binario en b+1 bits partiendo del primer 1.

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

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	γ(Piso(log2(n))+1)	n-2^Piso(log2(n))	δ(n)
1	1	1	~~1~~	1
2	2	0 10	~~1~~0	0 10 0
3	2	0 10	~~1~~1	0 10 1
4	3	0 11	~~1~~00	0 11 00
5	3	0 11	~~1~~01	0 11 01
6	3	0 11	~~1~~10	0 11 10
7	3	0 11	~~1~~11	0 11 11
8	4	00 100	~~1~~000	00 100 000
9	4	00 100	~~1~~001	00 100 001
10	4	00 100	~~1~~010	00 100 010
...	...	...		
48	6	00 110	~~1~~10000	00 110 10000

Para decodificar un código:

1. Se cuentan los b bits consecutivos en 0 que prefijan el código hasta encontrar un 1
2. Se obtiene el número x, representado en binario en b+1 bits partiendo del primer 1.
3. Se obtiene el número y, representado en binario en los x-1 bits siguientes.
4. 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.

Compresión Aritmética Global

Se basa en las probabilidades de las distancias, calculadas en función de la probabilidad de que un término se encuentre en un documento: si la probabilidad de que un término se encuentre en un documento número cualquiera n, es p, entonces la probabilidad de una distancia x se puede calcular como la probabilidad de que el término no se encuentre en los próximos x-1 documentos (n+1 a

n+d) pero sí en el siguiente (el documento número n+d+1):

P(x) = (1-p)^(x-1)*p (distribución de Bernoulli)

La probabilidad p puede calcularse en función de la cantidad total R de referencias del índice (representadas como distancias), la cantidad total D de documentos y la cantidad total T de términos:

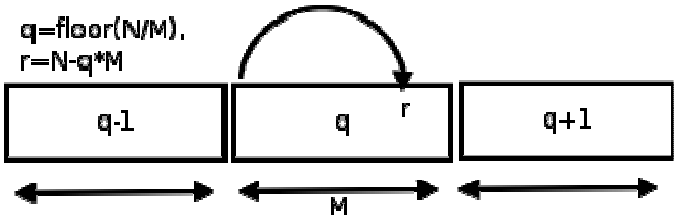
p = R/(D*T)

Compresión Aritmética Local

La compresión aritmética también se aplicar para comprimir listas, es decir, como técnica *local*: 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.

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 en b-1 bits

Si r ≥ 2^b-M se representa 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))

La codificación de Golomb se puede usar como técnica de compresión global o local de distancias, igual que la compresión aritmética.

Para codificar la lista de distancias del caso de ejemplo: 3, 48, 31, 15, 62, 5, 64, 187 con compresión local, 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 011111
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 010110
187	4	19	00001	5	00001 10011
				42 bits	

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

Organización de Índices de Inversión de Textos

En los índices de inversión de textos se distinguen claramente dos tipos de componentes: los términos indexados, que conforman lo que se llama el *vocabulario* del índice, y para cada término la lista de referencias a documentos o *lista invertida.*

Las consultas posibles sobre estos índices son las consultas booleanas, que permiten conjunciones, disyunciones y diferencias de términos:

- Consultas conjuntivas o AND: los documentos deben tener todos los términos.
- Consultas disyuntivas u OR: los documentos deben contener algún término.
- Consultas de diferencia o BUTNOT: los documentos deben contener uno o más términos y no contener a otros.

Por ejemplo, si se quisiera recuperar documentos relativos a los atentados terroristas del 11 de septiembre (en EEUU) y del 11 de marzo (en España), se podría consultar:

```
(11 AND (Septiembre OR Marzo)) BUTNOT 1973
```

La exclusión de 1973 se debe a que el 11 de septiembre de ese año es famoso en Chile por el golpe militar de Pinochet, y también el 11 de marzo de ese año es famoso en Argentina por haberse efectuado las primeras elecciones generales después de diez años de dictadura.

Las consultas se evalúan buscando las listas invertidas para los términos involucrados, y luego haciendo intersecciones de listas para consultas conjuntivas, uniones para consultas disyuntivas, y diferencias para las consultas de ese tipo. Una vez procesadas las listas y obtenida la lista definitiva de identificadores de documentos candidatos, se debe mapear los identificadores con las direcciones físicas de los documentos en una *tabla de direcciones* que puede estar en memoria o en disco.

Una estructura efectiva para almacenar vocabularios es la de los árboles B+. El alto factor de bifurcación típico de estos árboles implica que los nodos internos ocupen un porcentaje pequeño del tamaño del vocabulario. Por ejemplo, suponiendo que las hojas del índice apuntan a las listas invertidas, codificadas con alguna de las técnicas vistas anteriormente y organizadas a su vez en bloques del mismo archivo índice o de un archivo auxiliar con bloques más pequeños, que el tamaño de los nodos es de 8 Kb y que la raíz del índice está en memoria, se podría asumir que bastarían dos o tres accesos a disco para obtener la lista de documentos de un término (Zobel, J.; Moffat, A.; Ramamohanarao, K.:*"Inverted Files Versus Signature Files for Text Indexing"* , ACM 1998).

Es factible tener el vocabulario completo en memoria. Una ley empírica en Recuperación de Información, la ley de Heaps, establece que el vocabulario de un texto de largo n mide

```
k = C * n^x

Donde C y 0 < x < 1 dependen del tipo de texto. En la práctica,

5 < C < 50 y 0,4 ≤ x ≤ 0,6, y k es menos del 1% de n
```

Por ejemplo, 1 Gb de texto tendría un vocabulario de 5 Mb. Entonces se puede mantener el vocabulario ordenado en un arreglo en memoria y localizar términos mediante búsqueda binaria. El principal inconveniente de este esquema es su rigidez ante la posibilidad de que, al agregar nuevos textos a la base, aparezcan términos nuevos: ello implicaría el corrimiento en memoria de porciones de varios Mb de registros de índice; y si se usara una estructura de árbol para mantener los registros de índice en memoria, se complicaría la persistencia al tener que convertir estructuras de almacenamiento en memoria a estructuras de almacenamiento en disco al momento de persistir el índice, o viceversa al momento de cargarlo en memoria.

Otra posibilidad para almacenar el vocabulario en disco son los archivos directos, pero con la dificultad de encontrar funciones de dispersión efectivas y el riesgo de desaprovechar mucho espacio de almacenamiento.

Continuando con la idea de organizar un índice en un árbol B+, los registros de índice tendrían entonces que tener los siguientes campos:

```
término, frecuencia (cantidad de documentos en los que aparece),
referencia a lista de documentos
```

Compresión de Términos

Como los términos son de longitud variable, se puede ahorrar más espacio que el que ya se ahorra codificando las listas, almacenándolos abreviados: como en cada nodo del índice los términos están ordenados lexicográficamente y por tanto dos términos consecutivos pueden comenzar con varios caracteres en común, de cada término sólo se almacena los caracteres no compartidos con el anterior (*front coding*).

Por ejemplo, sea la secuencia de registros de índice:

```
(ábaco, 3, 9), (abad, 2, 5), (abadía, 3, 5), (abajo, 508, 1),
(abalanz, 10, 9), (abandon, 7, 6), (abanico, 2, 8), (abarat, 1,
8), (abarc, 4, 6), (abarrot, 3, 6)
```

Los términos abalanz, abandon, abarat, abarc y abarrot son raíces.

La referencia a la lista de documentos en cada registro es un número de bloque donde se empaquetan varias listas. Este esquema permite modificar las listas cuando se agreguen documentos a la base de textos. Como un bloque de almacenamiento de listas podría contener más de una lista de referencias, las listas deben almacenarse con una referencia al término que corresponden (por ejemplo, repitiendo el término) o bien con un número de control que debe acompañar al número de bloque en el índice. Las listas se organizan como registros de longitud variable en un archivo indexado por el índice; este archivo requiere a su vez estructuras de control para organizar el espacio libre.

Para almacenar los registros en un nodo hoja se requieren dos campos de control de un byte cada uno para distinguir los términos: uno que indique la cantidad de caracteres en común, y otro que indique la cantidad de caracteres almacenados:

```
(0, 5, abaco, 3, 9), (3, 1, d, 2, 5), (4, 2, ía, 3, 5), (3, 2,
jo, 508, 1), (3, 4, lanz, 10, 9), (3, 4, ndon, 7, 6), (4, 3,
ico, 2, 8), (3, 3, rat, 1, 8), (4, 4, c, 4, 6), (4, 3, rot, 3,
6)
```

En el ejemplo se asume que los caracteres con signos ortográficos (acentos, diéresis) se sustituyen por el correspondiente sin signo.

Si se considera que los nodos del índice son de 8 Kb, la cantidad de registros que puede tener cada nodo resulta considerable a la hora de buscar un término, ya que las búsquedas en cada nodo son secuenciales y desabreviando los términos.

Para acelerar las búsquedas evitando desabreviar todos los términos, conviene agrupar los registros en secciones tales que el primer término de cada sección se encuentre sin abreviar, y cada sección tenga la distancia en bytes al primer término de la próxima, para permitir saltar al primer registro de la próxima. La cantidad de registros de cada sección conviene que ronde la raíz cuadrada de la cantidad de registros en el nodo; así, en el ejemplo anterior, asumiendo que las frecuencias y referencias a listas ocupan dos bytes cada campo, los registros quedarían:

```
30[(0, 5, abaco, 3, 9), (3, 1, d, 2, 5), (4, 2, ía, 3, 5)],
35[(0, 5, abajo, 508, 1), (3, 4, lanz, 10, 9), (3, 4, ndon, 7,
6)], 0[(0, 7, abanico, 2, 8), (3, 3, rat, 1, 8), (4, 4, c, 4,
6), (4, 3, rot, 3, 6)]
```

Se agrega un campo por sección indicando la distancia en bytes al primer término de la próxima sección, de manera que al buscar un término, primero se recorran los comienzos de sección desde la segunda (sin desabreviar), y una vez determinada la sección de pertenencia (al encontrar el primer término de inicio de sección mayor al buscado), se busque dentro de ella desabreviando términos. Cada distancia en el ejemplo representa la posición relativa a partir de la posición de la distancia, al campo indicador de longitud del primer término de la sección siguiente. La distancia 0 indica última sección. El indicador de bytes abreviados del término en el primer registro de cada sección se conserva por generalización, pero podría eliminarse.

En caso de que se opte por mantener el vocabulario en memoria, es posible aplicar las mismas técnicas de abreviación, considerando secciones de tamaño fijo, sin necesidad de tener campos de control.

También es factible aplicar otras técnicas de compresión, siempre y cuando la técnica que se aplique permita:

- Obtener una buena tasa de compresión.
- Descomprimir eficientemente.
- Descomprimir el texto a partir de cualquier posición sin tener que descomprimir todo lo anterior.
- Buscar en el texto comprimido sin descomprimirlo.

Los modelos más exitosos para comprimir texto son los basados en palabras: los símbolos de la secuencia son las palabras y no los caracteres. Un simple Human sobre palabras obtiene 25% de compresión. El alfabeto fuente coincide con el vocabulario, lo que facilita integrarlo en un sistema de RI (Recuperación de Información). Como el alfabeto es grande, es posible que el árbol de Huffman tenga aridad 256 y todavía la compresión es razonable (30 %). En este caso la salida es una secuencia de bytes, que se descomprime muy rápidamente. Es posible buscar desde palabras hasta expresiones complejas en el texto comprimido, más rápido que en el texto original.

El resultado es una situación en que se gana por todos lados:

- Texto más índice ocupan menos espacio que el texto original: 40%-75% según el tipo de índice.
- El texto se busca secuencialmente más rápido que el texto original: 3 a 8 veces más rápido (sin contar con que se tiene un índice).
- El texto siempre esta comprimido y se descomprime sólo para mostrarlo.

Construcción de Índices

La construcción de índices depende de la organización que se escoja. Si se escoge una organización de árbol B+, puede ser razonable construirlo recorriendo secuencialmente cada documento, y por cada término que se lea, buscarlo en el índice (si no estuviera, agregar el término con una lista de documentos vacía), recuperar la lista de documentos asociada, y, si el número de documento que se está leyendo no figura al final de la lista, agregarlo. Este procedimiento es muy costoso en tiempo y genera índices con la densidad usual de los árboles B+. Para construir árboles más densos se puede utilizar un método que mantenga el vocabulario ordenado en memoria de manera que los términos se busquen por pruebas binarias o dicotómicas, y una vez construido se cree el árbol en disco con un procedimiento de altas ordenadas y carga de nodos con un porcentaje determinado. La inserción

y búsqueda de términos en el arreglo de memoria se pueden realizar mediante una función de dispersión con conservación de orden (se puede construir la función a partir de un muestreo de términos indexables, para determinar la probabilidad de letras iniciales de términos, con la inversa de la función de distribución acumulativa de probabilidad), de manera que cuando haya que insertar nuevos términos no se tenga que desplazar todos los términos mayores sino tan sólo una porción.

Si se tiene el vocabulario en memoria, las listas de documentos se pueden construir también en listas enlazadas en memoria, con el riesgo de que la memoria no alcance... Si se opta por armar las listas en disco durante el proceso, el costo en accesos puede resultar muy alto; entonces una forma práctica para construir las listas es registrar en un archivo secuencial de tipo bitácora (registro de eventos), el número de aparición del término más el número de documento actual. El número de aparición del término se debe guardar en memoria junto con el término. Una vez terminados de recorrer todos los documentos, se ordena el archivo por número de término eliminando duplicados, se guarda una copia del vocabulario en disco, se ordena el vocabulario en memoria por número de término, y así se puede construir las listas por procesamiento coordinado de ambas estructuras.

Por ejemplo, sean los siguientes documentos:

Número de Documento	Contenido
1	Vendo autos y camionetas
2	Autos usados
3	Excelente oferta de camionetas
4	Autos de segunda mano
5	Autos y camionetas de ocación
6	Permuto auto por caminoeta
7	Autos y más autos

Se aclara que las palabras ocación, en el documento 5, y caminoeta, en el documento 6, están mal escritas porque se supone así están en los documentos...

Luego del proceso de construcción, el vocabulario quedaría ordenado en memoria:

Término	Orden de aparición
auto	2
caminoeta	11
camioneta	3
excelente	5
mano	8
ocación	9
oferta	6
permut	10
segund	7
usad	4
vend	1

Y el archivo secuencial:

Nro. de aparición	Nro. de documento
1	1
2	1
3	1
2	2
4	2
5	3
6	3
3	3
2	4
7	4
8	4
2	5
3	5
9	5
10	6
2	6
11	6
2	7
2	7

Luego, ambas estructuras reordenadas, una en memoria y otra por ordenación externa, quedan:

Término	Orden de aparición
vend	1
auto	2
camioneta	3
usad	4
excelente	5
oferta	6
segund	7
mano	8
ocación	9
permut	10
caminoeta	11

Nro. de aparición	Nro. de documento
1	1
2	1
2	2
2	4
2	5
2	6
2	7
~~2~~	~~7~~
3	1
3	3
3	5
4	2
5	3
6	3
7	4
8	4
9	5
10	6
11	6

De allí es posible armar por recorrido coordinado de ambas estructuras:

Término	Lista de documentos
vend	1
auto	1, 2, 4, 5, 6, 7
camioneta	1, 3, 5
usad	2
excelente	3
oferta	3
segund	4
mano	4
ocación	5
permut	6
caminoeta	6

Estas listas se codifican comprimidas y se graban en un archivo en bloques, referidas desde el vocabulario, ya sea nuevamente en memoria ordenado por términos u organizado como árbol B+ en un archivo.

Índices por Firmas

Una alternativa a los índices de inversión de textos son los archivos de firmas (s*ignature files*).

Una firma digital de un archivo es cualquier secuencia de bytes de tamaño fijo, es decir sin relación con el tamaño del archivo, que se obtenga mediante una función que procese su contenido (en archivos de texto, sus términos). La razón de que se llame firma digital o sólo firma, es que su propósito principal es el de validación de integridad; de este modo, un CRC (control de redundancia cíclica) de un archivo podría considerarse una firma: permite validar la integridad del archivo luego de su transmisión o recuperación de un dispositivo de almacenamiento, que pudo verse afectada ya sea por problemas técnicos o por un virus.

Para recuperación de textos, la idea es que la firma de un documento sirva para verificar si un término cualquiera puede estar contenido en el mismo. Para esto se determina el tamaño en bits B que tendrá la firma, y se escoge una serie de funciones de dispersión (*hash*) que, aplicadas a cualquier término, devuelvan un entero entre 0 y B-1, determinando los bits que se pondrán en uno en la firma. Es inevitable que más de un término de un documento establezca el mismo bit de su firma en 1, por lo que así como las firmas no son infalibles para validar integridad, tampoco lo son para asegurar si un término está contenido en un documento: todo documento que tenga bits en 1 en las mismas posiciones que en la firma de un término de consulta se debe recuperar e inspeccionar en busca del término antes de agregarlo como resultado de la consulta...

Archivos de Firmas (Bitstring Signature Files)

Un índice de firmas es un archivo secuencial con un registro con la firma correspondiente a cada documento.

Por ejemplo, para los documentos de avisos clasificados de autos y camionetas, se obtiene firmas de 24 bits usando tres funciones de dispersión para cada término:

Término	h1	h2	h3
vend	21	10	16
auto	3	9	18
camioneta	9	5	3
usad	9	10	20
excelente	9	20	0
oferta	17	7	23
segund	22	5	18
mano	1	3	23
ocación	15	14	5
permut	3	12	22
caminoeta	3	11	7

Número de Documento	Contenido	Firma
1	Vendo autos y camionetas	`00010100 01100000 10100100`
2	Autos usados	`00010000 01100000 00101000`
3	Excelente oferta de camionetas	`10010101 01000000 01001001`

Número de Documento	Contenido	Firma
4	Autos de segunda mano	01010100 01000000 00100011
5	Autos y camionetas de ocación	00010100 01000011 00100000
6	Permuto auto por caminoeta	00010001 01010000 00100010
7	Autos y más autos	00001000 01000000 00100000

Para consultar, por ejemplo, documentos que contengan los términos de “camionetas usadas”, se construye la firma de la consulta fc=00010100 01100000 0001000 y luego se recupera la firma de cada documento fd, y si (fd and fc)=fc se considera el documento como candidato para la respuesta, aunque hay que validar si contiene o no todos los términos de la consulta.

Número de Documento	Contenido	Firma
1	Vendo autos y camionetas	00010100 01100000 10100100
2	Autos usados	00010000 01100000 00101000
3	Excelente oferta de camionetas	10010101 01000000 01001001
4	Autos de segunda mano	01010100 01000000 00100011
5	Autos y camionetas de ocación	00010100 01000011 00100000
6	Permuto auto por caminoeta	00010001 01010000 00100010
7	Autos y más autos	00001000 01000000 00100000

Como no hay ningún documento candidato se debe intentar otra consulta. Por ejemplo si sólo se buscara documentos para el término camioneta, los documentos candidatos son 1, 3, 4, 5, debiendo descartarse el 4.

Archivos de Porciones de Firmas (Bitslice Signature Files)

Para resolver una consulta con un archivo de firmas, es necesario recorrer todo el archivo. Para reducir el costo en accesos a disco se puede almacenar las firmas transpuestas: cada registro i del archivo de transposición representa los bits en la i-ésima posición de la firma de todos los documentos; de manera que para determinar los documentos que puedan contener un término t, sólo sería necesario recuperar tantos registros cuantos unos tenga la firma de t (a lo sumo la cantidad de funciones de dispersión que se empleen para construir las firmas).

Los registros de porciones de firma del cado de ejemplo serían:

Número de Porción	Porción
0	0010000
1	0001000
2	0000000
3	1111110
4	0000001

Número de Porción	Porción
5	1011100
6	0000000
7	0010010
8	0000000
9	1111111
10	1100000
11	0000010
12	0000000
13	0000000
14	0000100
15	0000100
16	1000000
17	0010000
18	1101111
19	0000000
20	0110000
21	1000000
22	0001010
23	0011000

Entonces, para determinar qué documentos son candidatos a contener el término “camioneta”, como la firma del término tiene unos en las posiciones 3, 5 y 9, se recupera las porciones de firma correspondientes a estas posiciones `1111110, 1011100 y 1111111`, y la conjunción de las porciones resulta con unos en los documentos candidatos: `1011100` (los documentos cuyos números corresponden a las posiciones de los bits en 1: 1, 3, 4 y 5).

Archivos de Porciones de Firmas de Grupos de Documentos (Blocked Signature Files)

Un problema particular de los archivos de porciones de firma es el tamaño de las porciones, que tienen un bit por documento; entonces si la cantidad de documentos indexados es muy grande, los registros del archivo de porciones resultan gigantes. Este problema se puede atenuar agrupando documentos en bloques, de manera que cada bit de cada porción de firma corresponda a B documentos, siendo B el factor de agrupación. La longitud de las porciones se divide entonces por B, pero para mantener baja la densidad de las porciones (pocos unos) se debe aumentar el tamaño de las firmas. Para reducir la probabilidad de que un bloque contenga todos los términos de una

consulta pero ningún documento del bloque quede en el resultado, los números de documento deben mapearse distinto en cada porción; esto es, los documentos deben asignarse a distintos bloques en cada porción. El número n de mapeos distintos de documentos debe ser submúltiplo del tamaño en bits tf de la firma de los documentos; cada mapeo se aplica a tf/n porciones.

Si bien el caso de ejemplo no es adecuado para graficar esta técnica dada la poca cantidad de documentos, supóngase que se decide agruparlos de a dos: las porciones se reducirían a 4 bits, y habría que mapear los documentos en grupos distintos en cada porción: por ejemplo, si el documento d estuviera en la posición i de una porción completa, podría asignarse la posición (i+1) % d en la porción siguiente. La tabla del ejemplo anterior quedaría con cada porción rotada un bit a derecha y luego los pares de bits 1-2, 3-4 y 5-6 reemplazados por la disyunción entre ambos. Pero para obtener resultados similares que con los documentos sin agrupar, habría que duplicar el tamaño de las firmas.

Consultas de Búsqueda de Texto

Modelo Booleano

La forma general de las consultas booleanas es:

$$(t_{11} \vee t_{12} \vee ... \vee t_{1x}) \wedge (t_{21} \vee t_{22} \vee ... \vee t_{2y}) \wedge ... \wedge (t_{n1} \vee t_{n2} \vee ... \vee t_{nz})$$

Donde t_{ij} son términos.

Esta forma es útil porque las consultas se usan generalmente para identificar documentos referidos a cada uno de los conjuntos de conceptos representados por las disyunciones, y las disyunciones representan términos equivalentes para representar conceptos.

Por ejemplo, continuando con el caso previo, una consulta sobre ventas de camionetas usadas podría plantearse como:

camionetas $\wedge$ (vendo $\vee$ venta $\vee$ oferta $\vee$ usadas $\vee$ segunda)

- Para resolverla con índices de inversión se debe recuperar las listas de documentos que contengan cada término, y hacer la intersección de la lista de los que contengan camioneta con el resultado de la unión de listas de los que contengan el resto de los términos (los de la disyunción múltiple).
- Para resolverla con archivos de firma, se debe chequear cada documento con firma fd tal que:

```
((fd and f(camioneta))=f(camioneta))
and
(((fd and f(vend))=f(vend)) or ((fd and f(venta))=f(venta)) or ((fd
and f(oferta))=f(oferta)) or ((fd and f(usad))=f(usad)) or ((fd and
f(segund))=f(segund)))
```

  Donde f(t) es la firma del término t

El modelo booleano es bastante limitado, ya que:

- Hay sólo dos posibilidades de relevancia de un documento en una consulta, es decir no tiene grados intermedios de relevancia.
- Da lo mismo que un término de una consulta aparezca pocas o muchas veces en un documento resultado.
- No importa cuántas cláusulas de una disyunción se cumplan.
- No considera resultados aproximados o parciales, como por ejemplo que se cumplan casi todas las cláusulas de una conjunción.
- No se puede ordenar los documentos resultantes de una consulta.
- Los usuarios inexpertos cometen muchos errores de lógica, por ejemplo, para buscar documentos sobre los atentados del 11/S y del 11/M, es probable que se pida “11/S AND 11/M”; es un error, ya que se perderían documentos que traten de uno sólo de los atentados en profundidad: debe pedirse “11/S OR 11/M”.

Modelo Vectorial

El modelo vectorial permite obtener documentos que respondan a una consulta ordenados por la importancia de los términos de la consulta en cada documento. Para esto, todo documento d_i de la

base se modela como un vector

$$(p(t_1, d_i), p(t_2, d_i), \ldots p(t_k, d_i))$$

Donde $\{t_1, t_2, \ldots t_k\}$ es el conjunto de todos los términos (el vocabulario) de la base de documentos, y $p(t_h, d_i)$ es el peso del término h en el documento i.

Hay varias fórmulas para asignar pesos a términos en un documento. La idea es medir *cuánto ayuda un término a distinguir un documento del resto*, para lo cual lo más natural es equilibrar de algún modo la importancia relativa del término en el documento con la importancia relativa del término en el conjunto de documentos, ya que un término puede ser importante en un documento porque se repite muchas veces, pero si ese término aparece en muchísimos documentos (por ejemplo una palabra común) entonces no sería relevante para ninguna consulta.

Para denotar la importancia o peso global de un término se utiliza la fórmula de George Zipf:

$$pg(t_h) = \log_{10}(n/ft_h)$$

Donde n es la cantidad de documentos en la base, y ft_h frecuencia global del término h, es decir la cantidad de documentos en los que aparece. Esta fórmula refleja que la importancia global de un término es grande cuando el término aparece en pocos documentos, y escasa cuando aparece en muchos.

Entonces, una fórmula apta para el peso de un término h en un documento i es:

$$p(t_h, d_i) = (ft_{h,i} * pg(t_h))/norma(d_i)$$

Donde $ft_{h,i}$ es la frecuencia (cantidad de veces que aparece) del término h en el documento i.

La normalización de los pesos es importante para independizar el peso del término del tamaño del documento, ya que en un documento largo, el término puede aparecer muchas veces y eso aumentaría la relevancia del documento en una consulta, en desmedro de otros documentos donde el término puede ser objetivamente más importante pero en los que aparece menos veces por ser más cortos. Entonces la división por la norma relativiza la frecuencia del término en relación a la de todos los términos.

Para el caso de ejemplo:

Número de Documento	Contenido
1	Vendo autos y camionetas
2	Autos usados
3	Excelente oferta de camionetas
4	Autos de segunda mano
5	Autos y camionetas de ocación
6	Permuto auto por caminoeta
7	Autos y más autos

Considerando los once términos del vocabulario ordenados alfabéticamente:

Término	Frecuencia (ft)	Peso global (log(7/ft))
auto	6	0,067
caminoeta	1	0,845
camioneta	3	0,368

Término	Frecuencia (ft)	Peso global (log(7/ft))
excelente	1	0,845
mano	1	0,845
ocación	1	0,845
oferta	1	0,845
permut	1	0,845
segund	1	0,845
usad	1	0,845
vend	1	0,845

Por ejemplo, para el documento 1:

- `La norma es Sqrt(1 * 0,067^2 + 1 * 0,368^2 + 1 * 0,845^2) = 0,924`
- `El modelo del documento: (1 * 0,067 / 0,924; 0; 1 * 0,368 / 0,924; 0; 0; 0; 0; 0; 0; 0; 1 * 0,845 / 0,924) = (0,073; 0; 0,398; 0; 0; 0; 0; 0; 0; 0; 0,914)`

Para el documento 2:

- `Norma: Sqrt(1 * 0,067^2 + 1 * 0,845^2) = 0,848`
- `Vector: (1 * 0,067 / 0,848; 0; 0; 0; 0; 0; 0; 0; 0; 1 * 0,845 / 0,848; 0) = (0,079; 0; 0; 0; 0; 0; 0; 0; 0; 0,996; 0)`

Etc.

Similitud de Documentos

Se puede considerar que cuando los vectores que representan a documentos tienen más o menos la misma dirección, están indicando mayor o menor similitud de contenidos; así, para comparar las direcciones de dos vectores $\overrightarrow{d1}$ y $\overrightarrow{d2}$ se calcula su producto:

$$\overrightarrow{d1} * \overrightarrow{d2} = \left|\overrightarrow{d1}\right| * \left|\overrightarrow{d2}\right| * \text{Cos}\,\alpha$$

Luego se despeja:

$$\text{Cos}\,\alpha = (\overrightarrow{d1} * \overrightarrow{d2}) / (\left|\overrightarrow{d1}\right| * \left|\overrightarrow{d2}\right|)$$

Y el coseno del ángulo entre ambos vectores ya sirve como medida de similitud, ya que cuanto mayor es el coseno menor es el ángulo (más parecida la dirección).

Para resolver consultas se determina el vector de la consulta y se calcula para cada documento el coseno del ángulo entre el vector consulta y el del documento; luego se ordena la lista de documentos en orden descendente del coseno.

Por ejemplo para la consulta c: “Camionetas usadas”, el vector consulta sería:

- `La norma es Sqrt(1 * 0,368^2  +  1 * 0,845^2) = 0,922`
- `El vector: (0; 0; 1 * 0,368 / 0,922; 0; 0; 0; 0; 0; 0; 1 * 0,845 / 0,922; 0) = (0; 0; 0,399; 0; 0; 0; 0; 0; 0; 0,916; 0)`

Y la similitud con d1:

```
Sim(d1, c) = (0,073; 0; 0,398; 0; 0; 0; 0; 0; 0; 0; 0,914) * (0; 0; 0,399; 0; 0; 0; 0; 0; 0; 0,916; 0) = 0,159
```

Y con d2:

```
Sim(c, d2) = (0,079; 0; 0; 0; 0; 0; 0; 0; 0; 0,996; 0) * (0; 0; 0,399; 0; 0; 0; 0; 0; 0; 0,916; 0) = 0,912
```

O sea que el orden de relevancia de los documentos 1 y 2 para la consulta es 2, 1. Si bien la

consulta comparte un término con ambos documentos, el resultado se explica porque el término que comparte con el documento 2 tiene más peso global que el término que comparte con el documento 1.

El modelo vectorial es el más utilizado hoy en día, ya que permite clusterizar documentos similares.

Implementación del Modelo Vectorial

El modelo vectorial sólo puede implementarse con índices de inversión, adaptados para sustentar el modelo: para cada término del vocabulario, no basta con guardar la lista de documentos en los que aparece, sino que además hay que guardar cuántas veces aparece en cada documento; esta lista, que se puede llamar de ocurrencias del término, conviene que esté ordenada por la frecuencia de aparición del término en el documento. También es conveniente guardar junto con el término, además de la frecuencia global o longitud de la lista de ocurrencias, la máxima frecuencia en un documento, para controlar la resolución de consultas sin necesidad de recuperar todas las listas previamente.

En el caso de ejemplo:

Término	Frecuencia Global	Máxima Frecuencia	Lista de documentos
auto	6	2	(7, 2), (1, 1), (2, 1), (4, 1), (5, 1), (6, 1)
caminoeta	1	1	(6, 1)
camioneta	3	1	(1, 1), (3, 1), (5, 1)
excelente	1	1	(3, 1)
mano	1	1	(4, 1)
ocación	1	1	(5, 1)
oferta	1	1	(3, 1)
permut	1	1	(6, 1)
segund	1	1	(4, 1)
usad	1	1	(2, 1)
vend	1	1	(1, 1)

Resolución de Consultas

Las consultas pueden tener muchos términos, y sólo interesa obtener una lista con los R documentos más relevantes, manteniendo el *ranking* de los R documentos di con mayor similitud con la consulta a resolver.

Se empieza por el término de la consulta con mayor peso global (es decir, menor frecuencia global), y se traen los primeros R. Si no hay R documentos en los que aparece el término candidato se sigue con el siguiente término de mayor peso global. Una vez que se tiene R candidatos se sigue recorriendo las listas con el mismo criterio, aunque como las frecuencias de los términos en las listas de documentos están en orden decreciente, se puede determinar un tope de recorrido de cada lista cuando los documentos no puedan entrar al ranking de los R mejores. Si se almacena la máxima cantidad de veces que aparece un término en un documento en con el vocabulario (en el registro índice, como se dijo antes), se puede descartar términos sin recuperar las listas.

La relevancia de un documento dj para una consulta c se calcula $Sim(d_j, c) = p(t_1, d_j) * p(t_1, c) + p(t_2, d_j) * p(t_2, c) + ... + p(t_k, d_j) * p(t_k, c) \approx Suma_{t \in c}(p(t, d_j) * \log(n / ft))$.

Obsérvese que no se normalizan los pesos, ya que es imposible calcular las normas a partir del índice (el índice tiene información por términos, y no se almacena la representación vectorial de los

documentos).

Entonces la resolución de la consulta del ejemplo para un ranking de 3 documentos sería:

- De los términos "camioneta" y "usad", el de mayor peso global (menor frecuencia global) es "usad", entonces se consideran de la lista de documentos correspondientes hasta 3; como sólo hay un documento en el que aparece "usad" se toma ese documento, y se toman los restantes de la lista de "camioneta": 2, 1, 3
- Se calcula la similitud aproximada de esos documentos con la consulta para ordenarlos:

  Sim(2, c) ≈ p(camioneta, 2) * 0,368 + p(usad, 2) * 0,845 = 0 + 0,845^2 = 0,714

  Sim(1, c) ≈ p(camioneta, 1) * 0,368 + p(usad, 1) * 0,845 = 0,368^2 + 0 = 0,135

  Sim(3, c) ≈ p(camioneta, 3) * 0,368 + p(usad, 3) * 0,845 = 0,368^2 + 0 = 0,135

  El orden es 2, 1, 3
- Si se pidieran más documentos, sólo queda el último de la lista de "camioneta", el 5

Consultas Complejas

Hay sistemas de recuperación de textos que ofrecen posibilidades de búsqueda más complejas.

- Frases: los términos de una consulta deben aparecer juntos y en el orden especificado, por ejemplo “Torres Gemelas”. Aquí no alcanza con saber si los términos están en el documento, sino que hay que saber su posición en el mismo para poder verificar que estén a la misma distancia entre sí que en la consulta.
- Términos próximos: los términos de una consulta deben aparecer “cerca” unos de otros, por ejemplo “auto” CERCA DE “usado”, para encontrar documentos con textos como:

  ```
  ... vendo auto usado ...
  ... gran venta de autos de todos los modelos, nuevos y usados ...
  ... permuto por auto o camioneta usados ...
  ```

  Igual que para las consultas de frases, para la consultas de términos próximos es necesario conocer las posiciones de los términos en los documentos para poder calcular distancias. Si no se exige que los términos estén cerca, la consulta podría devolver muchos documentos irrelevantes.
- Patrones: conjuntos de términos que contengan un substring o determinado tipo de caracteres, por ejemplo “doc[0-9]+.html?”.
- Búsqueda aproximada: se busca en los documentos la ocurrencia de un patrón de búsqueda con hasta un número limitado de diferencias. Esto es útil cuando el texto o el usuario puede tener errores de ortografía o tipeo, o cuando se busca apellidos raros.

Para las **consultas de frases y términos próximos**, se debe almacenar para cada término la lista de posiciones de ocurrencia del término en cada documento. Para reducir el espacio que ocupa el índice se puede tener un *índice de bloques*: se divide a los documentos en bloques de texto, y se refieren los bloques donde aparece cada término en lugar de las posiciones.

Término	Frecuencia	Posiciones
auto	6	1-1(2), 2-1(1), 4-1(1), 5-1(1), 6-1(2), 7-2(1, 2)
caminoeta	1	6-1(3)
camioneta	3	1-1(3), 3-1(3), 5-1(2)

excelente	1	3-1(1)
mano	1	4-1(3)
ocación	1	5-1(3)
oferta	1	3-1-(2)
permut	1	6-1(1)
segund	1	4-1(2)
usad	1	2-1(2)
vend	1	1-1(1)

Observar que las posiciones relativas en los documentos se establecen ignorando las palabras que no se indexan. Esto puede traer problemas, porque una consulta por “autos y camionetas” devolvería en el resultado a los documentos 1 y 5, y si el término camioneta estuviera bien escrito en el documento 6, también lo incluiría, siendo que no contiene una frase coincidente. *Cuando se ignoran las palabras no indexadas para establecer posiciones, se debe confirmar la ocurrencia exacta de la frase en los documentos candidatos a incluir en el resultado de las consultas.*

Las **consultas de frases** también se pueden resolver con *índices de pares de términos*. Estos índices son de tres niveles: en el primero se ubica al primer término, en el segundo al segundo término, y en el tercero se encuentra la lista de referencias a documentos con las posiciones del par de términos en cada documento. Las posiciones son necesarias para buscar frases con más de dos palabras. Por ejemplo para buscar “autos de segunda mano”, luego de eliminar “de” por ser un término no indexable, se debe buscar documentos donde aparezcan los pares auto-segund y segund-mano, y luego buscar coincidencias de documentos en ambas listas con posiciones a distancia 1.

El índice para el caso de ejemplo sería:

Término 1	Término 2	Frecuencia	Posiciones
auto	auto caminoeta camioneta segund usad	1 1 2 1 1	7-1(1) 6-1(2) 1-1(2), 5-1(1) 4-1(1) 2-1(1)
camioneta	ocación	1	5-1(2)
excelente	oferta	1	3-1(1)
oferta	camioneta	1	3-1(2)
permut	auto	1	6-1(1)
segund	mano	1	4-1(2)
vend	auto	1	1-1(1)

Aquí también las palabras no indexadas son problemáticas, por lo que se debe verificar la ocurrencia de las frases en los documento candidatos igual que en el caso del índice con posiciones.

Las **consultas de patrones** se pueden resolver mediante *índices secundarios de n-gramas*. Para cada secuencia de n caracteres, se refiere a los términos que los contienen. Los más comunes son los índices de digramas, por ejemplo, el término “camioneta” estaría referido por los digramas |c, ca, am, mi, io, on, ne, et, ta, a|. El símbolo | (o cualquier otro convencional) se usa para denotar principio o fin de término. Si se buscara documentos con el término “*oneta”, para incluir, por ejemplo a camioneta y furgoneta, se hace una consulta conjuntiva en el índice secundarios por los digramas on, ne, et, ta y a|, y luego una disyuntiva en el índice de inversión con todos los términos

resultantes.

Otra alternativa para resolver **consultas de patrones** son los *índices secundarios de léxico rotado*. Se guardan todas las rotaciones de cada término, por ejemplo para camioneta: |camioneta, camioneta|, amioneta|c, mioneta|ca, ioneta|cam, oneta|cami, neta|camio, eta|camion, ta|camione, a|camionet. Para buscar "*oneta", se rota el patrón hasta que el carácter comodín quede al final: *oneta| - |*oneta – a|*onet – ta|*one – eta|*on – neta|*o – oneta|*, y luego se busca todas las entradas de índice secundario que comiencen con oneta|. Estos índices ocupan más espacio que los de n-gramas, pero permiten resolver las consultas más rápido.

Los índices de n-gramas y de léxico rotado insumen mucho espacio, sobre todo los de léxico rotado (250% más que el índice invertido), pero usando cualquiera de ambos, el tiempo de resolución de consultas por patrones se reduce al 0,1% de lo que cuesta resolverlas por fuerza bruta.

Las **consultas de búsqueda aproximada** se pueden resolver optimizando el índice de inversión y transformando consultas con *sustituciones fonéticas* (de igual forma que se realiza las sustituciones por raíces), y de manera similar a la que funciona la autocorrección en los editores de texto con las listas de sugerencias, sustituyendo términos de la consulta por una lista de términos que se extraen de una lista y que se convierten en disyunciones.

Búsquedas en Internet

Internet se puede considerar como un gigantesco texto distribuido en una red de baja calidad, con contenido pobremente escrito, no focalizado, sin una buena organización y consultado por los usuarios más inexpertos:

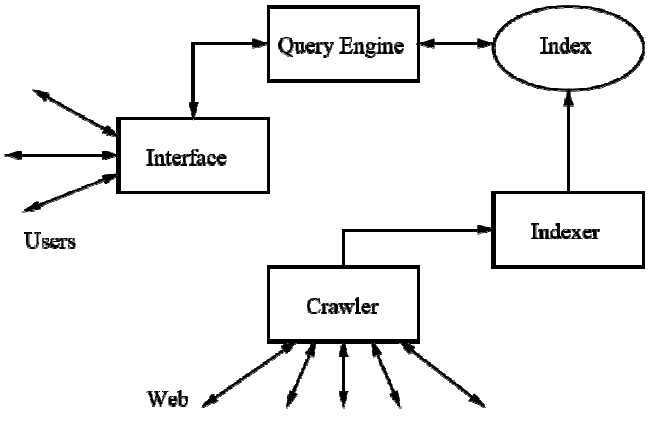
- Volumen gigantesco y creciente a nivel exponencial.
- Información distribuida y conectada por una red de calidad variable.
- Texto altamente volátil: el 40% cambia cada mes, y duplica su tamaño en meses.
- Información mal estructurada y redundante: el 30% de las páginas son prácticamente duplicados.
- Información de calidad variable, sin revisión editorial de forma ni contenido: un error de tipeo cada doscientas palabras comunes y cada tres apellidos extranjeros.
- Información heterogénea: tipos de datos, formatos, idiomas, alfabetos.
- Usuarios generalmente inexpertos, con consultas de 1 a 3 palabras y pobremente formuladas.

Arquitectura de Máquinas de Búsqueda

Componentes básicos:

- *Araña (web crawler):* programa que inspecciona páginas de la web en forma metódica y automatizada creando copias locales para indexarlas. Comienza con una serie de URL que se le deben proporcionar como punto de partida, y luego sigue con enlaces que encuentra en las mismas páginas.
- *Indexador:* indexa las páginas nuevas.
- *Máquina de Consulta:* realiza búsquedas en el índice.
- *Intefaz:* buscador.

El programa araña o robot corre localmente en la máquina de búsqueda descargando páginas, y el indexador, en la misma máquina, mantiene un índice con el texto que extrae de ellas. La máquina de búsqueda también corre localmente realizando búsquedas en el índice y devolviendo URL ordenadas por relevancia (rankeadas). La interfaz corre en los clientes y permite efectuar las consultas y mostrar los resultados:



Ordenación de Consultas (ranking) en la Web

La mayoría de las máquinas de búsqueda en la Web usa variantes del modelo booleano o vectorial. El texto no suele estar accesible al momento de la consulta, por lo que el direccionamiento a bloques es impensable; aún si el texto se almacenase localmente, el proceso sería muy lento.

Una diferencia importante entre indexar páginas web y la recuperación de textos de otro tipo está dada por los enlaces: una página que recibe muchos enlaces se puede considerar muy popular. Páginas muy conectadas entre sí o referenciadas desde la misma página podrían tener contenido similar.

Google valora las páginas por la probabilidad estacionaria de que un proceso de recorrido automático se encuentre con ella. El recorrido salta a una página aleatoria con probabilidad q y se mueve por un link aleatorio con probabilidad 1-q; si a es una página apuntada por las páginas p1, p2,… pn, y C(p) es la cantidad de enlaces que salen de una página p, entonces el rango de a es:

$$PR(a) = q + (1 - q) \sum_{i=1}^{n} PR(pi) / C(pi)$$

Donde q se suele fijar en 0,15.

Índices para la Web

La mayoría de las máquinas de búsqueda usa variantes del índice de inversión. No se puede almacenar el texto completo por el volumen que representa: se almacenan algunas cosas, como el título, los primeros caracteres (unos 400), fecha de creación, tamaño, etc.

Si se considera una media de 500 bytes por página, sólo las descripciones requerirían 50 Gb para 100 millones de páginas. Dados los tamaños de las páginas web, los índices requieren aproximadamente el 30% del tamaño del texto. Para 100 millones de páginas con 5 Kb de texto en promedio de cada una, el índice ocuparía 150 Gb.

Los buscadores en Internet normalmente no permiten búsquedas aproximadas o por patrones arbitrarios, ya que son impensables por la carga de la máquina de consultas. Según Heaps, 1Tb de texto generaría un vocabulario de 300 Mb y requeriría hasta 300 segundos para recorrerlo.

Los resultados de las búsquedas se presentan normalmente de en rangos de entre 10 y 20 documentos; el resto se mantiene en memoria para cuando el usuario los pida (para no tener que recalcularlos). Como el protocolo http no tiene concepto de sesión, las respuestas se mantienen en memoria caché, ya sea en forma de páginas más consultadas, pedazos de listas invertidas más populares, etc.