Tópicos em Recuperação de Informação[1]

Nivio Ziviani

[1]Conjunto de transparências elaborado por Nivio Ziviani e Patrícia Correia

Indexação

Índice

Mecanismo utilizado para localizar um dado termo em um texto (custo $<$ linear).

Tipos mais comuns de índices:

- Arquivos Invertidos (*inverted files* ou *postings files*).
- Arquivos de Assinaturas (*signature files*).
- Mapas de Bits (*bitmaps*).

Arquivos Invertidos

Tipicamente composto de:

1. arranjo contendo todas as palavras distintas do texto (chamado de vocabulário).
2. para cada palavra do vocabulário, uma lista de todos os documentos (identificados por números de documentos) nos quais aquela palavra ocorre.

A parte 2 pode ser mais geral: para cada palavra do vocabulário, uma lista de apontadores para todas as ocorrências daquela palavra no texto.

Exemplo de Arquivo Invertido

Documento	Texto
1	Pease porridge hot, pease porridge cold,
2	Pease porridge in the pot,
3	Nine days old.
4	Some like it hot, some like it cold,
5	Some like it in the pot,
6	Nine days old.

Exemplo de Arquivo Invertido

Número	Termo	Documento
1	cold	(2;1,4)
2	days	(2;3,6)
3	hot	(2;1,4)
4	in	(2;2,5)
5	it	(2;4,5)
6	like	(2;4,5)
7	nine	(2;3,6)
8	old	(2;3,6)
9	pease	(2;1,2)
10	porridge	(2;1,2)
11	pot	(2;2,5)
12	some	(2;4,5)
13	the	(2;2,5)

Consultas em Arquivos Invertidos

Termo único

Busca no vocabulário e recupera a lista de ocorrências (em documentos).

Conjunção: “palavra AND ... AND palavra”

Intersecção das listas de ocorrências.

Disjunção: “palavra OR ... OR palavra”

União das listas de ocorrências.

Negação: “NOT ...”

Complemento do resultado.

Exemplo: “some AND hot”

$< 4, 5 >$ AND $< 1, 4 > \ \Rightarrow \ < 4 >$

Granularidade do Índice

Corresponde a precisão com que uma palavra é localizada pelo índice:

1. Grossa (“*coarse-grained*”) : identifica apenas um bloco de texto.

2. Moderada (“*moderate-grained*”) : identifica apenas um documento.

3. Fina (“*fine-grained*”) : identifica cada palavra (outras vezes, cada byte).

Granularidade do Índice

Grossa:

- Índice requer menos memória.
- Consulta exige mais processamento em cada bloco.
- Consulta de frases leva a maior número de *"false matches"* (cada palavra aparece em algum lugar do bloco mas não em posições contíguas).

Fina (Indexação por palavra):

- Busca por frases é eficiente.
- Proximidade pode ser tratada sem ir ao texto.
- Busca aproximada é eficiente.
- Índice requer mais memória (uma ordem de grandeza).

Indexação por Palavra

Número	Termo	(Documento;Palavras)
1	cold	[2;(1;6),(4;8)]
2	days	[2;(3;2),(6;2)]
3	hot	[2;(1;3),(4;4)]
4	in	[2;(2;3),(5;4)]
5	it	[2;(4;3,7),(5;3)]
6	like	[2;(4;2,6),(5;2)]
7	nine	[2;(3;1),(6;1)]
8	old	[2;(3;3),(6;3)]
9	pease	[2;(1;1,4),(2;1)]
10	porridge	[2;(1;2,5),(2;2)]
11	pot	[2;(2;5),(5;6)]
12	some	[2;(4;1,5),(5;1)]
13	the	[2;(2;4),(5;5)]

Índice cresce:

- Cada apontador: número do documento e a posição da palavra no documento.

- Uma palavra pode ocorrer várias vezes em um mesmo documento.

Espaço Ocupado pelo Índice

Texto: N documentos

Índice: f apontadores

Espaço: $f.\lceil \log N \rceil$ bits

Métodos para Construção de Índice

Existem vários métodos para a construção de índices invertidos:

1^o - Matriz de freqüência (*frequency matrix*)

- Cada linha corresponde a um documento e cada coluna corresponde a um termo do vocabulário.

- Para criar o índice, deve-se gerar a transposta da matriz de freqüência.

Construção do Índice

Matriz de freqüência

Documento	Texto
1	Pease porridge hot, pease porridge cold,
2	Pease porridge in the pot,
3	Nine days old.
4	Some like it hot, some like it cold,
5	Some like it in the pot,
6	Nine days old.

	Termo												
	cold	days	hot	in	it	like	nine	old	pease	porridge	pot	some	the
1	1	–	1	–	–	–	–	–	2	2	–	–	–
2	–	–	–	1	–	–	–	–	1	1	1	–	1
3	–	1	–	–	–	–	1	1	–	–	–	–	–
4	1	–	1	–	2	2	–	–	–	–	–	2	–
5	–	–	–	1	1	1	–	–	–	–	1	1	1
6	–	1	–	–	–	–	1	1	–	–	–	–	–

Construção do Índice

Matriz de freqüência

Número	Termo	Documento					
		1	2	3	4	5	6
1	cold	1	–	–	1	–	–
2	days	–	–	1	–	–	1
3	hot	1	–	–	1	–	–
4	in	–	1	–	–	1	–
5	it	–	–	–	2	1	–
6	like	–	–	–	2	1	–
7	nine	–	–	1	–	–	1
8	old	–	–	1	–	–	1
9	pease	2	1	–	–	–	–
10	porridge	2	1	–	–	–	–
11	pot	–	1	–	–	1	–
12	some	–	–	–	2	1	–
13	the	–	1	–	–	1	–

Construção do Índice

Matriz de freqüência

- Para construir o índice: ler cada documento da coleção, ao final escrever a matriz, linha por linha no disco.

- Construção é bastante simples.

- Entretanto solução é bastante cara!

  Ex.: Bíblia contém 8.965 termos e 31.101 documentos.

  Tamanho da matriz: 8.965 x 31.101 x 4 bytes $\simeq$ 1 Gigabyte de memória principal.

- Soluções mais econômicas devem ser consideradas.

Construção do Índice

Coleção de 5 Gigabytes e 5 milhões de documentos.

Parâmetro	Símb.	Valores assumidos
Tamanho total do texto	B	5×10^9 bytes
Número de documentos	N	5×10^6
Número de palavras distintas	n	1×10^6
Número total de palavras	F	800×10^6
Número de ponteiros de índice	f	400×10^6
Tamanho final do arquivo invertido comprimido	I	400×10^6 bytes
Tamanho da estrutura de vocabulário dinâmico	L	30×10^6 bytes
Tempo de *seek* em disco	t_s	10×10^{-3} seg
Tempo de transferência em disco, por byte	t_r	$0,5 \times 10^{-6}$ seg
Tempo de codificação do arquivo invertido, por byte	t_d	5×10^{-6} seg
Tempo de comparação e troca de um registro de 10 b	t_c	10^{-6} seg
Tempo de *parsing*, *stemming* e procura por termo	t_p	20×10^{-6} seg
Total de memória principal disponível	M	40×10^6 bytes

Construção do Índice

Custo:

$$T = B.t_r + F.t_p + I.(t_d + t_r)$$

$B.t_r$: tempo de leitura
$F.t_p$: *parsing*
$I.(t_d + t_r)$: escreve arquivo invertido comprimido

- $B = 5 \times 10^9$ bytes (texto)
- $t_r = 0,5 \times 10^{-6}$ seg (tempo de transferência em disco)
- $F = 800 \times 10^6$ (número de palavras ou termos)
- $t_p = 20 \times 10^{-6}$ seg (*parsing* + *stemming* + *look up*)
- $I = 400 \times 10^6$ bytes (tamanho índice comprimido)
- $t_d = 5 \times 10^{-6}$ seg (tempo codificação)
- $T \cong 6$ horas (Bíblia < 30 segs)

Construção do Índice

Se fôssemos usar disco:

- 1^a fase:
  ler todos os documentos e gravar sequencialmente no disco:
  30 minutos a mais

- 2^a fase:
  agora cada $(d, f_{d,t})$ correspondente a um termo t está em um lugar diferente no disco!
  4 milhões de segundos $\cong 6$ semanas!

Construção do Índice

Método	Seção	Figura	Memória (Mbytes)	Disco (Mbytes)	Tempo (Horas)
Linked lists (memory)	5.1	5.1	4000	0	6
Linked lists (disk)	5.1	5.1	30	4000	1100
Sort-based	5.2	5.3	40	8000	20
Sort-based compressed	5.3	–	40	680	26
Sort-based multiway merge	5.3	–	40	540	11
Sort-based multiway in-place	5.3	5.8; 5.9	40	150	11
In-memory compressed	5.4	5.12	420	1	12
Lexicon-based, no extra disk	5.4	–	40	0	79
Lexicon-based, extra disk	5.4	–	40	4000	12
Text-based partition	5.4	–	40	35	15

Construção do Índice

2^o - Inversão em Memória (*linked lists*)

- O índice é todo construído em memória principal.
- Para armazenar o vocabulário: tabela *hash* é geralmente a escolha mais econômica em termos de espaço e velocidade.
- Listas encadeadas em memória para armazenar as listas invertidas dos termos.
- O método leva cerca de 6 horas para indexar uma coleção de 5GB e consome 4GB de memória principal e nenhum espaço extra em disco.

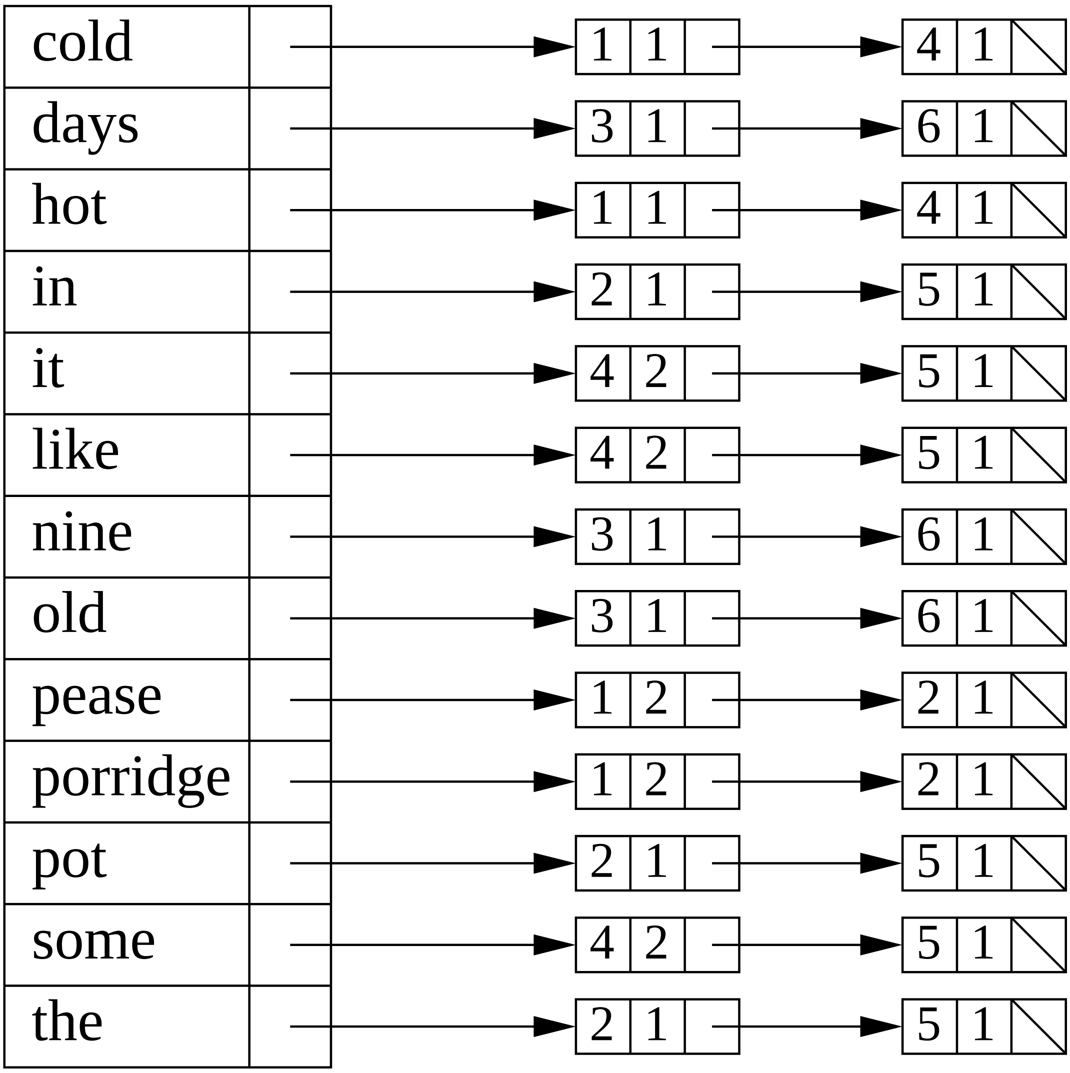
Construção do Índice

Inversão em Memória

1. /* Inicialização */
   Crie uma estrutura de dicionário vazia S.
2. /* Fase um: coleta das características dos termos */
   Para cada documento D_d na coleção, $1 \leq d \leq N$,
   (a) Leia D_d, realizando o *parser* para obter termos indexáveis.
   (b) **Para cada** termo indexável $t \in D_d$,
   i. **Faça** $f_{d,t}$ receber a frequência do termo t em D_d.
   ii. Busque por t em S.
   iii. **Se** t não estiver em S, insira-o.
   iv. Adicione um nó armazenando $(d, f_{d,t})$ na lista correspondente ao termo t.
3. /* Fase dois: saída do arquivo invertido */
   Para cada termo $1 \leq t \leq n$,
   (a) Inicialize uma nova entrada do arquivo invertido.
   (b) **Para cada** $(d, f_{d,t})$ na lista correspondente a t,
   Adicione $(d, f_{d,t})$ a essa entrada do arquivo invertido.
   (c) **Se** requerido, comprima a entrada do arquivo invertido.
   (d) Adicione essa entrada do arquivo invertido ao arquivo invertido.

Construção do Índice

Inversão em Memória



Estrutura de pesquisa *S*

listas armazenando pares $(d,\ f_{d,t})$

Construção do Índice

3^o - Inversão baseada em Ordenação (***sort-based***)

- O principal problema dos métodos descritos anteriormente é o alto consumo de memória principal.

- O uso de disco é inevitável quando grandes quantidades de texto precisam ser indexadas.

- Triplas <termo, doc, freq> são armazenadas em arquivos temporários.

- O índice é obtido ordenando-se estas triplas em ordem ascendente do termo e depois do documento.

1. /* Inicialização */
   Crie uma estrutura de dicionário vazia S.
   Crie um arquivo temporário vazio em disco.
2. /* Processe o texto e escreva arquivo temporário */
   Para cada documento D_d na coleção, $1 \leq d \leq N$,
   (a) Leia D_d, realizando o *parser* para obter termos indexáveis.
   (b) **Para cada** termo indexável $t \in D_d$,
   i. **Faça** $f_{d,t}$ receber a frequência do termo t em D_d.
   ii. Busque por t em S.
   iii. **Se** t não estiver em S, insira-o.
   iv. Escreva um registro $(t, d, f_{d,t})$ no arquivo temporário, onde t é representado por seu número de termo em S.
3. /* Ordenação interna para gerar *runs* */
   Faça k receber o número de registros que podem ser carregados em memória.
   (a) Leia k registros da arquivo temporário.
   (b) Ordene-os em ordem crescente de t e, para valores iguais a t, em ordem crescente de d.
   (c) Escreva a *run* ordenada novamente no arquivo temporário.
   (d) Repita até não haver mais *run* a ser ordenada.
4. /* *Merging* */
   Intercale as *runs* em pares no arquivo temporário até existir somente uma *run* ordenada.
5. /* Geração do arquivo invertido */
   Para cada termo $1 \leq t \leq n$,
   (a) Inicialize uma nova entrada do arquivo invertido.
   (b) Leia todas as triplas $(t, d, f_{d,t})$ do arquivo temporário e gere a emtrada do arquivo invertido para o termo t.
   (c) **Se** requerido, comprima a entrada do arquivo invertido.
   (d) Adicione essa entrada do arquivo invertido ao arquivo invertido.

Construção do Índice

Inversão baseada em Ordenação

- Um método de ordenação, como o *Quicksort*, pode ser usado para esta etapa.
- O resultado da ordenação é gravado em disco e chamado de *run* ordenada.
- A próxima fase é a intercalação de todas *runs* ordenadas para gerar o índice final.
- Se existirem R *runs*, depois de $\lceil \log R \rceil$ passos o processo de intercalação (*external sort*) é finalizado, restando um único arquivo temporário em disco.

Construção do Índice

Inversão baseada em Ordenação

- O passo final na indexação é ler o arquivo temporário ordenado, gerando uma saída comprimida, se necessário.

- A inversão para a coleção de 5GB leva cerca de 20 horas usando 40MB de memória principal e 8GB de espaço extra em disco.

- Devido a grande quantidade de espaço em disco consumida durante a inversão, este método é considerado o melhor para coleções de tamanho moderado (10 a 100 MB).

Reduzindo Custos na Construção do Índice

- Utilizar técnicas de compressão e estratégias mais sofisticadas de intercalação
  - Compressão dos arquivos temporários.
  - Intercalação de múltiplos caminhos (*multiway*).
  - Intercalação *in-place*.

Intercalação de Múltiplos Caminhos

- Uso de intercalação de múltiplos caminhos reduz o tempo de construção do índice e a quantidade de espaço em disco necessária para a geração do índice final.

- Como: fila de prioridade (*heap*).

- Custo: $\lceil \log R \rceil$ comparações.

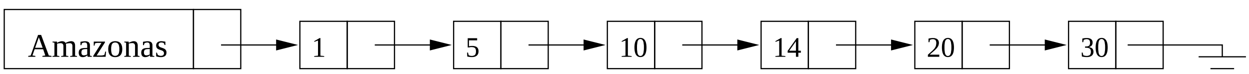
- Esta estratégia reduz o tempo de construção do arquivo invertido para a coleção de 5GB para cerca de 11 horas usando 40MB de memória principal e 540MB de espaço temporário em disco.

- Estes custos podem ainda ser reduzidos utilizando uma intercalação *in-place*.

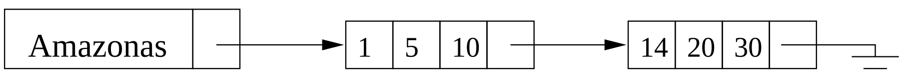
Compressão de Índice em Memória

Comprimir listas na memória equivale a aumentar a memória disponível.

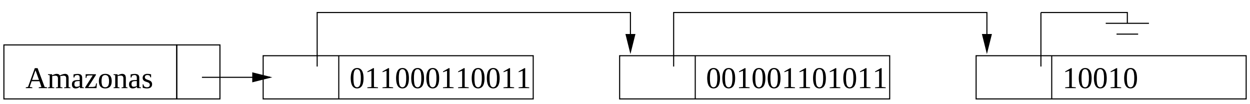
Representação de uma lista de ocorrências:



Utilizando blocos de memória:



Comprimindo dados na memória:



Custo de CPU: $O(u + \frac{u}{\sqrt{M}} \log \frac{u}{M})$

u: Tamanho do texto

M: Tamanho da memória principal