

Indexação

Prof. Dr. Leandro Balby Marinho

<http://www.dsc.ufcg.edu.br/~lbmarinho>



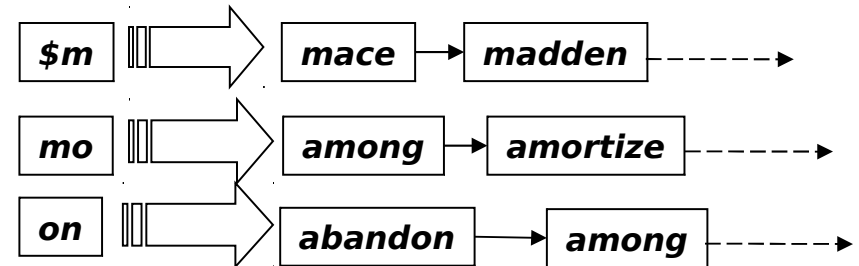
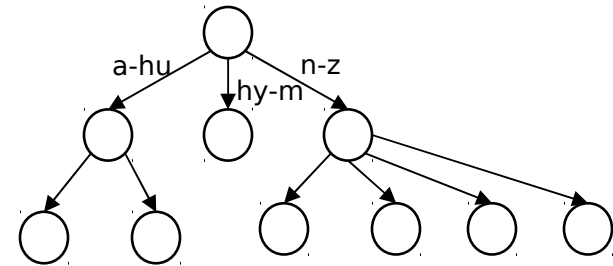
UFCG CEEI Departamento de
*Sistemas e
Computação*

Sistemas de Recuperação da Informação

(Slides Adaptados de Cristopher D. Manning)

Roteiro

- Última aula:
 - Dicionários
 - Recuperação tolerante
 - Curingas
 - Correção ortográfica
 - Soundex
- Hoje:
 - Construção do índice



Construção de Índice

- Como se constrói um índice?
- Que estratégias usar com memória limitada?

Básico sobre Hardware

- Acesso aos dados na memória é ***muito*** mais rápido que acesso aos dados no disco.
- Leitura do disco: nenhum dado é transferido do disco enquanto a cabeça de leitura estiver sendo posicionada.
- Portanto: transferir um grande pedaço de dados do disco para a memória é mais rápido que transferir muitos pedaços pequenos.
- I/O do disco é baseada em blocos: leitura e escrita de blocos inteiros (ao contrário de pequenos pedaços).
- Tamanho de blocos: 8KB a 256 KB.

Básico sobre Hardware

- Servidores usados em SRIs contemporâneos possuem tipicamente vários GB de memória principal, algumas vezes dezenas de GB.
- Espaço de disco disponível é várias (2-3) ordens de magnitude maior.
- Tolerância a falhas é muito cara: mais barato utilizar várias máquinas ordinárias do que uma tolerante a falhas.

Processando consultas na árvore de permutações

▪ símbolo	estatística	valor
▪ s	tempo médio de acesso	5 ms = 5×10^{-3} s
▪ b	tempo de transferência por byte	$0.02 \mu\text{s} = 2 \times 10^{-8}$ s
▪	frequência de relógio	10^9 s^{-1}
▪ p	operações baixo nível (e.g., compara & troca uma palavra)	$0.01 \mu\text{s} = 10^{-8}$ s
▪	tamanho de memória	vários GB
▪	espaço em disco	1 TB ou mais
▪	acessar 1 byte na mem.	5×10^{-9} s

RCV1: Nosso corpus para aula de hoje

- Como exemplo para algoritmos escaláveis de construção de índices, usaremos a coleção de documentos Reuters RCV1.
- Essa coleção corresponde a um ano de notícias do Reuters (entre Agosto de 1995 e Agosto de 1996).

Um Documento do Reuters



You are here: [Home](#) > [News](#) > [Science](#) > [Article](#)

Go to a Section: [U.S.](#) [International](#) [Business](#) [Markets](#) [Politics](#) [Entertainment](#) [Technology](#) [Sports](#) [Oddly Enough](#)

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

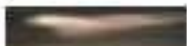
[Email This Article](#) | [Print This Article](#) | [Reprints](#)

[\[-\]](#) [Text](#) [\[+\]](#)



SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.



Estatísticas da coleção RCV1

■ símbolo	estatística	valor
■ N	documentos	800,000
■ L	tokens por doc (méd.)	200
■ M	termos	400,000
■	bytes por token (méd.) (incl. espaço/pont.)	6
■	bytes por token (méd.) (sem espaços/pont.)	4.5
■	bytes por termos (méd.)	7.5
■	postings não-posicionais	100,000,000

Relembre indexação da aula 2

- Documentos são analisados para a extração de termos, os quais são salvos com os resp. doclds.

Doc 1

I did enact Julius
Caesar I was killed
i' the Capitol;
Brutus killed me.

Doc 2

So let it be with
Caesar. The noble
Brutus hath told you
Caesar was ambitious



Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Etapa Chave

- Depois que todos os docs. foram analisados, o índice é ordenado por termo e depois por docID.

Foco na ordenação.
100M itens para ordenar.

Term	Doc #	Term	Doc #
I	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
I	1	caesar	1
was	1	caesar	2
killed	1	caesar	2
i'	1	did	1
the	1	enact	1
capitol	1	hath	1
brutus	1	I	1
killed	1	I	1
me	1	i'	1
so	2	it	2
let	2	julius	1
it	2	killed	1
be	2	killed	1
with	2	let	2
caesar	2	me	1
the	2	noble	2
noble	2	so	2
brutus	2	the	1
hath	2	the	2
told	2	told	2
you	2	you	2
caesar	2	was	1
was	2	was	2
ambitious	2	with	2



Indexação Escalável


- Indexação na memória não escala.
- Como podemos construir índices para grandes coleções?
- Levando em consideração as restrições de hardware . . .
- Memória, disco, velocidade, etc.

Construção de Índice

- Na construção do índice, analisamos um documento de cada vez.
- Os postings para cada termo são incompletos até o final.
- Assumindo 12 bytes por entrada de posting (*termo, doc, freq*), grandes coleções demandam muito espaço.
- 100,000,000 postings no RCV1
 - Mas há coleções bem maiores. E.g. o *New York Times* possui um índice de >150 anos de notícias.
- Portanto: Precisamos guardar resultados intermediários no disco.

Indexação no disco

- Podemos usar o mesmo algoritmo de indexação da aula 2 para grandes coleções, mas usando o disco em vez de memória?
- Fazendo isso com buscas de disco aleatória seria muito lento – temos $T=100M$ registros.

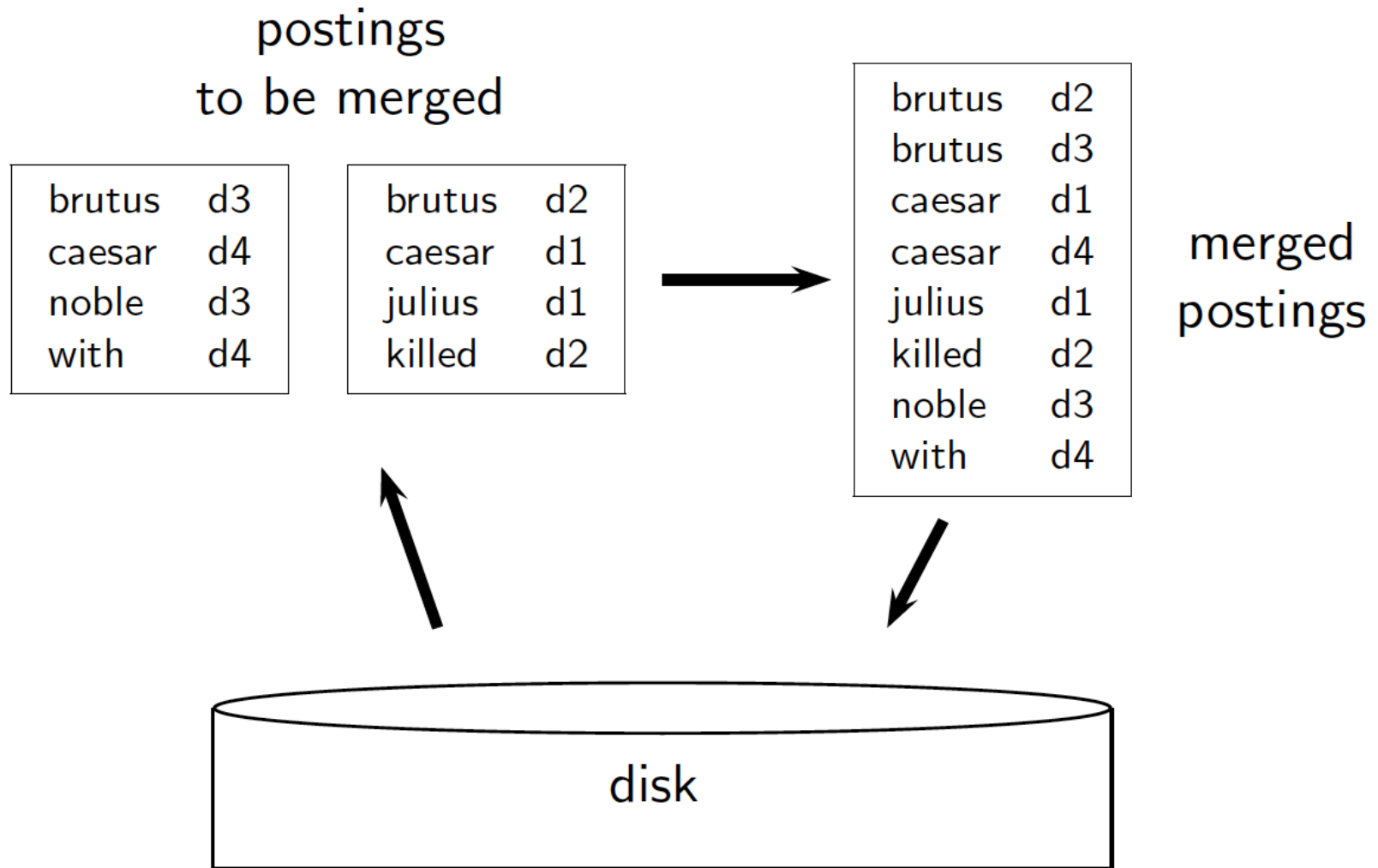


Se cada comparação precisa 2 buscas de disco, e N itens podem ser ordenadas em $N \log_2 N$ comparações, quanto tempo demoraria isso?

Indexação com poucas buscas no disco

- Registros de 12-bytes (4+4+4) (*termo*, *doc*, *freq*) são gerados quando analisamos os docs.
- Precisamos ordenar 100M desses registros de 12-bytes para cada *termo*.
- Defina Blocos ~ 10M
 - Tal que caibam confortavelmente na memória.
 - Teríamos 10 blocos para começar.
- Idéias básica do algoritmo:
 - Acumula postings para cada bloco, ordena, escreve para o disco.
 - Combina os blocos no final.

Indexação com poucas buscas no disco



Ordenando 10 blocos de 10M

- Primeiro, leia cada bloco e ordene:
 - Quicksort precisa de $N \ln N$ passos.
 - No nosso caso $10M \ln 10M$ passos.

Exercício: estime o tempo total para ler cada bloco do disco e ordenar com o quicksort.

Indexação baseada em blocos

BSBIINDEXCONSTRUCTION()

```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4       $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5      BSBI-INVERT( $block$ )
6      WRITEBLOCKTODISK( $block, f_n$ )
7  MERGEBLOCKS( $f_1, \dots, f_n; f_{\text{merged}}$ )
```

Problemas Restantes

- Nossa hipótese: podemos guardar o dicionário na memória.
- Precisamos do dicionário (que cresce dinamicamente) de forma a implementar o mapeamento termo/termID.
- Na verdade, poderíamos ter postings termo,docID em vez de postings termID,docID . . .
- . . . mas então arquivos intermediários se tornam muito grandes.

Indexação de Passada Única na Memória

- Ideia-chave 1: Dicionários separados para cada bloco – não é necessário manter mapas termo-termold entre blocos.
- Ideia-chave 2: Não ordena. Acumula postings em listas de postings durante o parsing.
- Com essas duas ideias podemos gerar um índice invertido completo para cada bloco.
- Esses índices separados podem então ser combinados em um índice maior no final.

Algoritmo SPIMI

```
SPIMI-INVERT(token_stream)
1  output_file = NEWFILE()
2  dictionary = NEWHASH()
3  while (free memory available)
4  do token  $\leftarrow$  next(token_stream)
5      if term(token)  $\notin$  dictionary
6          then postings_list = ADDTODICTIONARY(dictionary, term(token))
7          else postings_list = GETPOSTINGSLIST(dictionary, term(token))
8          if full(postings_list)
9              then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10         ADDTOPOSTINGSLIST(postings_list, docID(token))
11 sorted_terms  $\leftarrow$  SORTTERMS(dictionary)
12 WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13 return output_file
```

- Fusão de blocos análoga ao BSBI.

Indexação distribuída

- Para indexação na escala da Web:
 - Precisamos usar um cluster distribuído de computadores.
- Máquinas individuais sujeitas a falhas
 - Podem falhar ou ficar lentas de forma imprevisível.

Centro de Dados do Google

- Centros de dados do Google estão distribuídos pelo mundo.
- Estimativa: total de 1 milhão de servidores, 3 milhões de processadores/cores (Gartner 2007).
- Estimativa: Google instala 100,000 servidores por semestre.
 - Com base na despesa de 200-250 milhões de dólares por ano
- Isso seria 10% de toda a capacidade computacional do mundo!?!

Indexação Distribuída

- Mantém uma máquina *mestra* direcionando os *jobs* de indexação.
- Quebra a indexação em conjuntos de tarefas (paralelas).
- Máquinas mestras atribuem cada tarefa a uma máquina ociosa do “repositório” de máquinas.

Tarefas Paralelas

- Usaremos dois tipos de tarefas paralelas
 - Parsers
 - Inversores
- Quebre a coleção de documentos em *partições*
- Cada partição é um subconjunto de documentos (análogo aos blocos em BSBI/SPIMI)

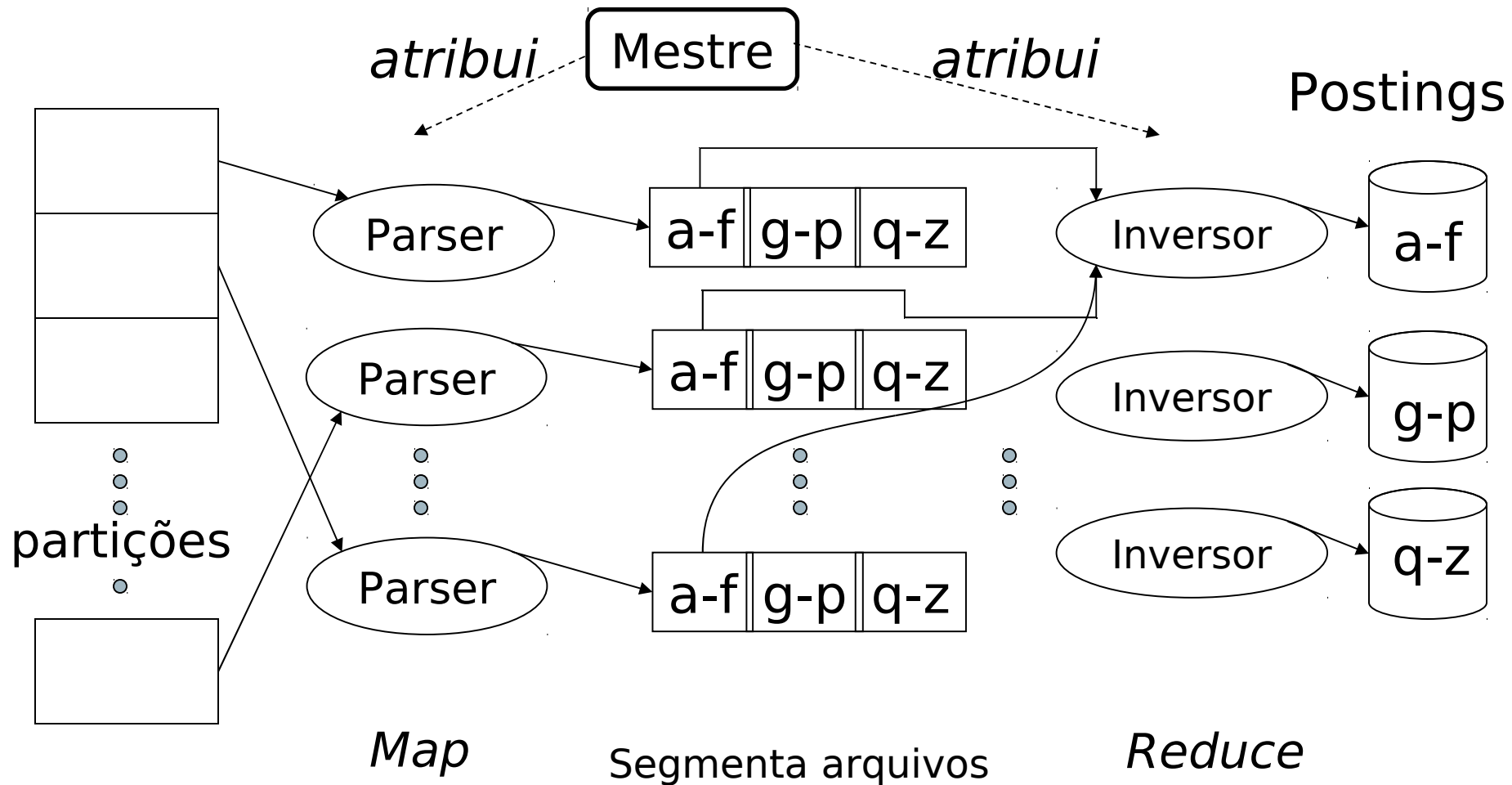
Parsers

- O *mestre* atribui partições a máquinas (parsers) ociosas.
- O *parser* analisa um documento por vez e gera pares (termo, doc).
- O parser escreve pares em j arquivos de acordo com a segmentação dos termos (ou docs.).
- Cada partição corresponde a um intervalo de letras referente as primeiras letras dos termos
 - (e.g., **a-f**, **g-p**, **q-z**) - aqui $j = 3$.

Inversores

- Um *inversor* coleta todos os pares (termo,doc) (= postings) para um segmento de termos.
- Ordena e escreve como listas de postings.

Fluxo de Dados



MapReduce

- O algoritmo de construção de índice recém descrito é uma instância do MapReduce.
- MapReduce (Dean and Ghemawat 2004) é um framework conceitual simples e robusto para computação distribuída.
- A indexação do Google (ca. 2002) é realizada em várias fases, cada uma implementada com MapReduce.

Esquema para indexação com MapReduce

- **Esquema das funções map e reduce**
- map: $\text{input} \rightarrow \text{list}(k, v)$ reduce: $(k, \text{list}(v)) \rightarrow \text{output}$
- **Instanciação do esquema para indexação**
- map: coleção da web $\rightarrow \text{list}(\text{termID}, \text{docID})$
- reduce: $(\langle \text{termID1}, \text{list}(\text{docID}) \rangle, \langle \text{termID2}, \text{list}(\text{docID}) \rangle, \dots) \rightarrow (\text{postings list1}, \text{postings list2}, \dots)$
- **Exemplo de construção de índice**
- map: $d2 : C \text{ died. } d1 : C \text{ came, } C \text{ c'ed. } \rightarrow (\langle C, d2 \rangle, \langle \text{died}, d2 \rangle, \langle C, d1 \rangle, \langle \text{came}, d1 \rangle, \langle C, d1 \rangle, \langle \text{c'ed}, d1 \rangle)$
- reduce: $(\langle C, (d2, d1, d1) \rangle, \langle \text{died}, (d2) \rangle, \langle \text{came}, (d1) \rangle, \langle \text{c'ed}, (d1) \rangle) \rightarrow (\langle C, (d1:2, d2:1) \rangle, \langle \text{died}, (d2:1) \rangle, \langle \text{came}, (d1:1) \rangle, \langle \text{c'ed}, (d1:1) \rangle)$

Indexação dinâmica

- Até agora assumimos que as coleções são estáticas.
- Isso raramente é o caso:
 - Novos documentos aparecem com o tempo e devem ser inseridos.
 - Documentos também são deletados e modificados.
- Portanto o dicionário e listas invertidas devem ser modificados:
 - Atualização de postings para termos existentes no dicionário
 - Adição de novos termos ao dicionário

Abordagem Simples

- Manter um índice principal.
- Novos docs vão para um índice auxiliar menor.
- Use os dois para busca e combine os resultados.
- Deleções
 - Vetor de bits de invalidação para documentos deletados
 - Filtra docs do resultado da busca através desse vetor de bits
- Periodicamente realiza a fusão do índice auxiliar com o índice principal.

Índices Dinâmicos em Motores de Busca

- A maioria dos motores de busca utiliza índices dinâmicos.
- Seus índices sofrem incrementos frequentemente
 - Novos itens, blogs, novas páginas, ...
- Mas (tipicamente) eles reconstroem o índice inteiro
 - O processamento da consulta é então direcionado ao novo índice e o antigo é deletado.

Recursos da Aula

- Livro Texto Cap. 4