Tópicos em Recuperação de Informação

Nivio Ziviani

Conjunto de transparências elaborado por Nivio Ziviani, Patrícia Correia e Fabiano C. Botelho.

Arquivos Invertidos

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

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)]

Indexação por Palavra

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

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

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.

Construção do Índice

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

	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

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

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

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

- 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

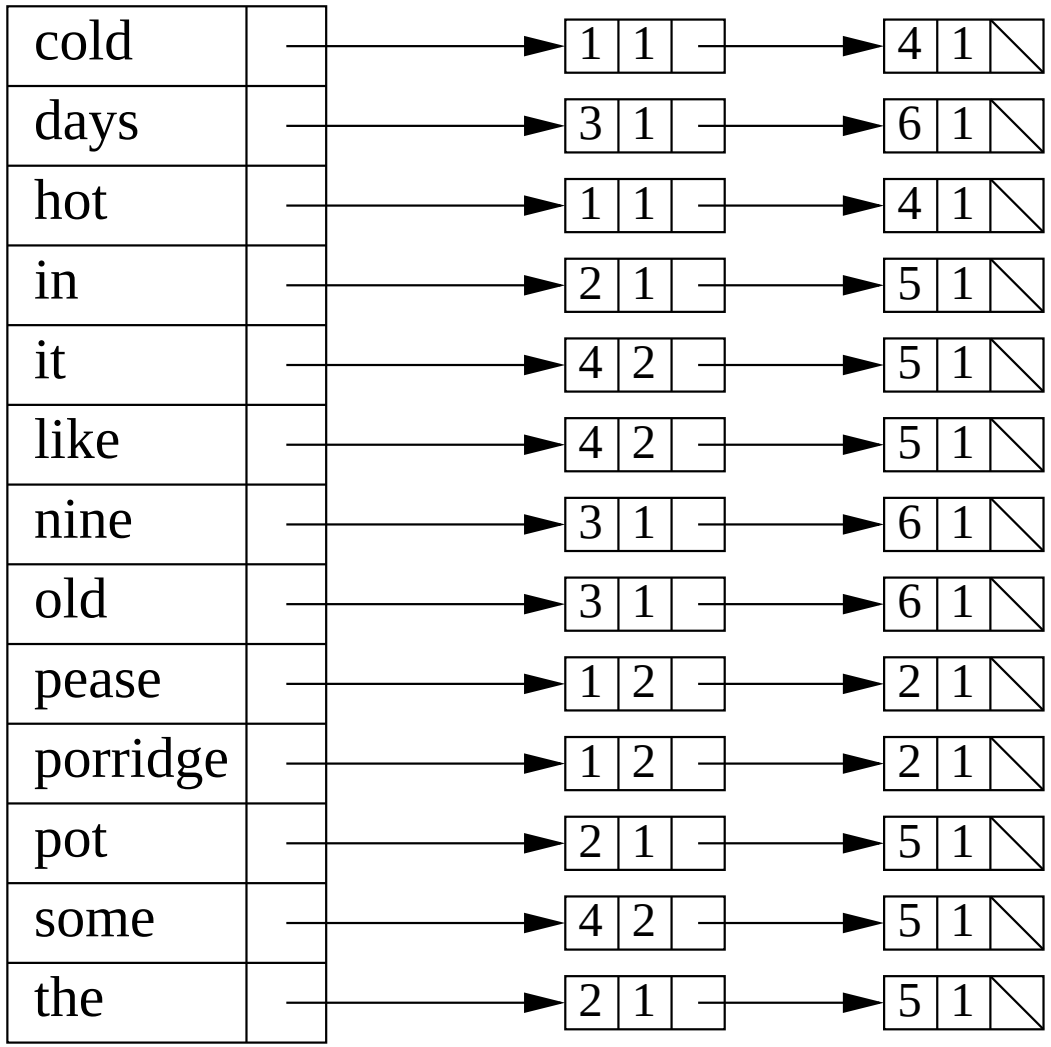
2^o – **Inversão em Memória (*Linked lists*)**

- 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

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



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.

Construção do Índice

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.

Construção do Índice

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

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

- 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

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

- 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).

Exemplo de Construção do Índice

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

Exemplo de Construção do Índice

Initial
< 1, 1, 2 >
< 2, 1, 2 >
< 3, 1, 1 >
< 4, 1, 1 >
< 1, 2, 1 >
< 2, 2, 1 >
< 5, 2, 1 >
< 6, 2, 1 >
< 7, 2, 1 >
< 8, 3, 1 >
< 9, 3, 1 >
< 10, 3, 1 >
< 11, 4, 2 >
< 12, 4, 2 >
< 13, 4, 2 >
< 3, 4, 1 >
< 4, 4, 1 >
< 11, 5, 1 >
< 12, 5, 1 >
< 13, 5, 1 >
< 5, 5, 1 >
< 6, 5, 1 >
< 7, 5, 1 >
< 8, 6, 1 >
< 9, 6, 1 >
< 10, 6, 1 >

Sorted runs
< 1, 1, 2 >
< 1, 2, 1 >
< 2, 1, 2 >
< 2, 2, 1 >
< 3, 1, 1 >
< 4, 1, 1 >
< 5, 2, 1 >
< 6, 2, 1 >
< 7, 2, 1 >
< 8, 3, 1 >
< 9, 3, 1 >
< 10, 3, 1 >
< 11, 4, 2 >
< 12, 4, 2 >
< 3, 4, 1 >
< 4, 4, 1 >
< 5, 5, 1 >
< 11, 5, 1 >
< 12, 5, 1 >
< 13, 4, 2 >
< 13, 5, 1 >
< 6, 5, 1 >
< 7, 5, 1 >
< 8, 6, 1 >
< 9, 6, 1 >
< 10, 6, 1 >

Merged runs
< 1, 1, 2 >
< 1, 2, 1 >
< 2, 1, 2 >
< 2, 2, 1 >
< 3, 1, 1 >
< 3, 4, 1 >
< 4, 1, 1 >
< 4, 4, 1 >
< 5, 2, 1 >
< 5, 5, 1 >
< 6, 2, 1 >
< 6, 5, 1 >
< 7, 2, 1 >
< 7, 5, 1 >
< 8, 3, 1 >
< 8, 6, 1 >
< 9, 3, 1 >
< 9, 6, 1 >
< 10, 3, 1 >
< 10, 6, 1 >
< 11, 4, 2 >
< 11, 5, 1 >
< 12, 4, 2 >
< 12, 5, 1 >
< 13, 4, 2 >
< 13, 5, 1 >

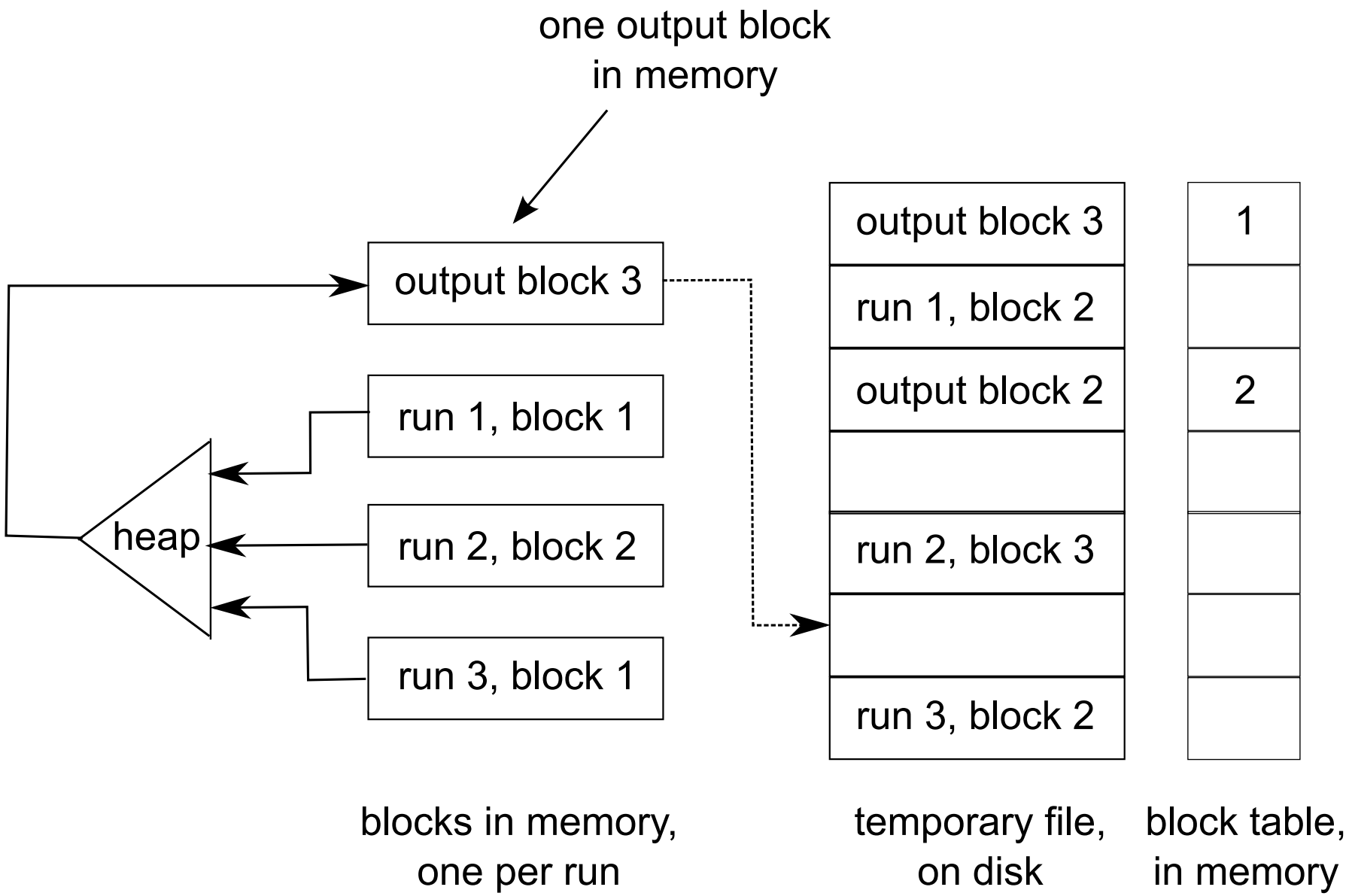
Reduzindo Custos na Construção do Índice

- 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*.

Intercalação de Múltiplos Caminhos



Algoritmo de Permutação In-Place

To permute blocks so that page i holds block i, assuming that, initially, the ith page holds block $block\_table[i]$

1. For $i \leftarrow 1$ to $nblocks$, if $i \neq block\_table[i]$ then
   (a) Read page i into memory.
   (b) Set $holding \leftarrow block\_table[i]$.
   (c) Set $vacant \leftarrow i$
   (d) While $holding \neq vacant$ do
      i. Find j such that $block\_table[j] = vacant$,
      ii. Copy page j to page $vacant$,
      iii. Set $block\_table[vacant] \leftarrow vacant$,
      iv. Set $vacant \leftarrow j$.
   (e) Write block in memory to page $vacant$.
   (f) Set $block\_table[vacant] \leftarrow holding$.

Algoritmo de de Inversão Melhorado

1. /* Initialization */
   – Create an empty dictionary structure S.
   – Create an empty temporary file on disk.
   – Set $L \leftarrow \mid S \mid$.
   – Set $k \leftarrow (M - L)/w$, where w is the number of bytes required to store one $< t, d, f_{d,t} >$ record.
   – $b \leftarrow 50$ Kbytes.
   – $R \leftarrow 0$.
2. /* Process text and write temporary file */
   – For each document D_d in the collection, $1 \leq d \leq N$,
      (a) Read D_d, parsing it into index terms.
      (b) For each index term $t \in D_d$,
         [i.] Let $f_{d,t}$ be the frequency in D_d of term t.
         [ii.] Search S for t.
         [iii.] If t is not in S,
            insert it,
            set $L \leftarrow \mid S \mid$,
            set $k \leftarrow (M - L)/w$.
         [iv.] Add a record $< t, d, f_{d,t} >$ to the array of triples.

Algoritmo de de Inversão Melhorado

(c) If, at any stage, the array of triples contains k items,
[i.] Sort the array of triples using Quicksort.
[ii.] Write the array of triples, coding t-gaps in unary; d-gaps with δ; and $f_{d,t}$ values with γ.
[iii.] Add padding to complete a block of b bytes.
[iv.] Set $R \leftarrow R + 1$.
[v.] $/*$ Reduce the block size if memory looks threatened $*/$
If $b * (R + 1) > M$,
Set $b \leftarrow b/2$

3. /* Merging */
(a) Read the first block from each run, add each block number to the free list.
(b) Build heap with R candidates, one from each run.
(c) While the heap is no empty,
i. Remove the root of the heap.
ii. Add it to the output block, recoding the d-gap and $f_{d,t}$ values.
iii. Replace it by the next candidate from the same run.

Algoritmo de de Inversão Melhorado

(d) Each time the output block becomes full,
  i. Use the free list to find a vacant disk block. If there is no vacant disk block available, create one by appending an empty block to the file.
  ii. Write the output block.
  iii. Update the free list and the block table.

(e) Each time an input block becomes empty,
  i. Read the next block from this run.
  ii. Update the free list.

4. /* In-place permutation */
   Reorder the blocks so that physical order corresponds to the logical order using algorithm of Figure 5.6.
5. Truncate the index file to its final length.

Manutenção do Índice

Inserção de 1 documento

- Adição de poucos bytes ao final da cada lista invertida correspondente a um termo do documento
- Para documentos grandes: 10-20 segundos Na indexação usando ordenação externa: 1000 documentos por segundo, cerca de 10.000 vezes mais lento
- Solução: amortizar o custo de atualização sobre uma sequência de inserções (evitar acesso a listas invertidas no disco)

Manutenção do Índice

Retirada de 1 documento

- Documento é marcado como retirado e mais tarde fisicamente retirado do índice
- Entre a invalidação e a retirada, documentos são removidos das respostas
- Amortizar o custo de atualização sobre uma sequência de retiradas (evitar acesso a listas invertidas no disco)

Manutenção do Índice

Estratégias de Atualização

- Reconstruir
- Intercalação intermitente
- Atualização incremental

Reconstruir

- Situações em que o índice não pode ser atualizado *online* (ou não vale a pena)
  Exemplo: Web site de uma universidade, onde novos documentos são descobertos apenas na coleta

Intercalação Intermitente

- Novos documentos são inseridos em um índice na memória principal
- Os dois índices compartilham um vocabulário comum
- Quando a memória enche (ou outro critério é atingido) o índice em memória é intercalado com o do disco
  O índice antigo pode ser usado até que a intercalação termine, a um custo de manter duas cópias completas do índice
- Durante a intercalação
  - novo índice em memória pode ser criado ou
  - inserções novas são bloqueadas até terminar a intercalação (e nesse caso novos documentos são acessíveis sequencialmente)

Atualização Incremental

- Atualiza o índice termo por termo (quando surgir uma oportunidade)
- Atualização é assíncrona
  - uma lista é recuperada do disco
  - a nova informação é adicionada ao final da lista
  - a lista é escrita de volta no disco
  - usando uma gerência de espaços disponíveis a lista pode até voltar para o mesmo lugar de origem ou ser escrita em outro lugar (se não houver espaço suficiente)

Atualização Incremental

- Atualização pode ocorrer
  - quando for responder a uma consulta
  - empregar um processo no *background* que lentamente fica percorrendo as listas
- Não é eficiente como a intercalação intermitente, a qual processa os dados sequencialmente no disco