Algoritmos e Estruturas de Dados II

Ordenação Externa II

Profa. Debora Medeiros

Slides originais do Prof. Ricardo J. G. B. Campello

Ordenação Externa

- As análises dos métodos de ordenação tradicionais se preocupam basicamente com o tempo de execução dos algoritmos
  - Ordem computacional é estimada em função da quantidade de operações (comparações, trocas, etc) feitas utilizando a memória principal (primária) da máquina
  - modelo de **ordenação interna**
- Quando é preciso ordenar uma base de dados muito grande, que não cabe na memória principal, um outro modelo faz-se necessário
- No modelo de **ordenação externa** assume-se que os dados devem ser recuperados a partir de dispositivos externos
  - ordenação em memória secundária

Ordenação Externa

- Como o acesso à memória secundária é muito mais lento, a maior preocupação passa a ser minimizar a quantidade de leituras e escritas nos respectivos dispositivos
- Uma dificuldade é que o projeto e análise dos métodos de ordenação externa dependem fortemente do estado da tecnologia
  - Por exemplo, o acesso a dados em fitas magnéticas é seqüencial, mais lento, enquanto em discos tem-se o acesso direto
  - Nesses últimos, no entanto, tem-se o tempo de localização de trilha (*seek time*) e de setor/cluster (*latency time*), que por sua vez dependem da velocidade de rotação do disco, da estrutura de dados utilizada para armazenamento, etc

Ordenação Externa

- Por estas razões, ao analisar o problema de ordenação externa usualmente utiliza-se um modelo simplificado, que abstrai ao máximo os detalhes tecnológicos
- Basicamente, preocupa-se com a quantidade de operações envolvendo a transferência de blocos de registros entre as memórias primária e secundária
  - Operações de leitura e escrita (L/E) ou de **acesso**

Ordenação Externa

- O modelo de ordenação externa assume que o hardware e o S.O. são dados (e.g. tamanho dos blocos), isto é, se direcionam ao programador e não aos projetistas
- Assume-se usualmente que os blocos contêm múltiplos registros
  - Pares chave-informação
- Assume-se usualmente que os registros possuem tamanho fixo
  - Caso contrário as análises ganham um caráter de "estimativa média"
- Por simplicidade e sem perda de generalidade, as análises subseqüentes assumem que um registro é simplesmente um número inteiro, que também é a sua chave

Ordenação Externa

- Sabemos que é possível ordenar um arquivo grande em disco separando-o em **k** arquivos ordenados em RAM e fazendo a fusão intercalada (merging) desses arquivos
  - Ordenação externa via multi-way merging
- Essa abordagem, porém, possui uma limitação:
  - A quantidade k e o tamanho dos arquivos são determinados pela memória primária disponível e pelo tamanho do arquivo a ser ordenado
    - Fusão pode ter que lidar com diversos arquivos de tamanho reduzido e necessariamente ordenados
    - Isso pode inviabilizar a paralelização de L/E em múltiplos dispositivos (tópico a ser discutido posteriormente)

Ordenação Externa

- Seria possível fazer a fusão ordenada de um no. arbitrário de arquivos de tamanho arbitrário, não ordenados e possivelmente armazenados em diferentes dispositivos externos (p. ex. fitas) ?

Ordenação Externa

- Seria possível fazer a fusão ordenada de um no. arbitrário de arquivos de tamanho arbitrário, não ordenados e possivelmente armazenados em diferentes dispositivos externos (p. ex. fitas) ?
  - A resposta é **SIM**.
  - Porém, existe um preço...
    - São necessárias múltiplas passagens coseqüenciais pelos arqs.
    - Algoritmo é denominado **Merge-Sort Externo**

Merge-Sort Externo

- A versão básica do algoritmo opera com 4 arquivos:
  - Para discutir a idéia do algoritmo, vamos inicialmente assumir que todos os 4 arquivos são armazenados em um único disco
- Os registros são lidos de 2 arqs. de origem e reescritos de forma parcialmente ordenada em 2 arqs. de destino
- Os arquivos de origem e destino se alternam nas sucessivas iterações do algoritmo.
- Utiliza-se o conceito de **rodada** (*run*):
  - Subconjuntos ordenados de registros

Merge-Sort Externo

- Inicialmente divide-se o arquivo original em dois arquivos f_1 e f_2, ditos **de origem**, com as seguintes propriedades:
  - O número de rodadas de f_1 e f_2 (incluindo eventual **cauda**) difere em no máximo 1
  - No máximo um dentre f_1 e f_2 possui uma cauda
  - Aquele com cauda possui pelo menos tantas rodadas quanto o outro
- Uma **cauda** é uma rodada com número incompleto de registros

f_1: 7 15 29 | 8 11 13 | 16 22 31 | 5 12

f_2: 8 19 54 | 4 20 33 | 00 10 62 |

rodada de tamanho 3

cauda

Merge-Sort Externo

- Em princípio vamos tratar os arquivos como estando organizados em rodadas de tamanho unitário

Início:

f_1:	28	03	93	10	54	65	30	90	10	69	08	22
f_2:	31	05	96	40	85	09	39	13	08	77	10	

- Inicia-se então a leitura de blocos de registros dos arquivos
- A sistemática de leitura dos blocos será discutida posteriormente
- Por hora considera-se que conjuntos de registros são lidos seqüencialmente de ambos os arquivos e intercalados
  - algoritmo *merging* **por rodadas**
- Isso significa que cada rodada de um arquivo é fundida com a rodada correspondente do outro arquivo, formando uma rodada com o dobro do tamanho

Merge-Sort Externo

- Ao final de cada passagem pelos arquivos, tem-se os arquivos ditos **de destino**, g_1 e g_2, organizados em rodadas com o dobro do tamanho dos arquivos de origem:

1a Passagem:

g_1:	28	31	93	96	54	85	30	39	08	10	08	10
g_2:	03	05	10	40	09	65	13	90	69	77	22	

- Esses arquivos tornam-se então os arquivos de origem e o processo se repete:

2a Passagem:

	03	05	28	31	09	54	65	85	08	10	69	77
	10	40	93	96	13	30	39	90	08	10	22	

Merge-Sort Externo

3a Passagem

03 05 10 28 31 40 93 96 | 08 08 10 10 22 69 77
09 13 30 39 54 65 85 90 |

4a Passagem

03 05 09 10 13 28 30 31 39 40 54 65 85 90 93 96
08 08 10 10 22 69 77

- Número de passagens?

5a Passagem

03 05 08 08 09 10 10 10 13 22 28 30 31 39 40 54 65 69 77 85 90 93 96
Ø

Merge-Sort Externo

3a Passagem

03 05 10 28 31 40 93 96 | 08 08 10 10 22 69 77
09 13 30 39 54 65 85 90 |

4a Passagem

03 05 09 10 13 28 30 31 39 40 54 65 85 90 93 96
08 08 10 10 22 69 77

- Como o tamanho das rodadas dobram a cada passagem, tem-se que após i passagens o tamanho da rodada é $k = 2^i$, e que quando $k \geq n$ (onde n é a quantidade total de registros a serem ordenados) tem-se:

5a Passagem

03 05 08 08 09 10 10 10 13 22 28 30 31 39 40 54 65 69 77 85 90 93 96
Ø

Desempenho (Interno)

- O número de passagens necessárias é portanto tal que $2^i \geq n$
- Logo, qualquer $i \geq \log n$ número de passagens são suficientes:
  - Ou seja, não mais que Teto[$\log n$] passagens bastam
- Como são n registros e a fusão se dá pela comparação de pares de chaves em tempo constante, a complexidade do algoritmo em termos de números de comparações é $O(n \log n)$
  - A mesma que o Merge-Sort recursivo tradicional (ordenação interna)
- Mas e o número de acessos ???

Desempenho (Externo)

- Tem-se que cada passagem requer a leitura e escrita de 2 arquivos, cada um com aproximadamente $n/2$ registros
  - No. de acessos em cada passagem $\sim= 4(n/2) \sim= 2n$
- Sabemos que as leituras e escritas podem ser feitas em blocos de registros através do uso de buffers em memória RAM
-

Desempenho (Externo)

- Tem-se que cada passagem requer a leitura e escrita de 2 arquivos, cada um com aproximadamente $n/2$ registros
  - No. de acessos em cada passagem $\sim= 4(n/2) \sim= 2n$
- Sabemos que as leituras e escritas podem ser feitas em blocos de registros através do uso de buffers em memória RAM
- Nesse caso, o número de leituras e escritas de blocos em cada passagem é em torno de $2n/b$, onde b é o tamanho (capacidade de registros) do bloco
- Logo, o número total de leituras e escritas de blocos em todo o processo de ordenação é em torno de $(2n \log n)/b$, ou seja, é de excelente ordem $O(n \log n)$

Otimização

- É possível minimizar o tempo de espera decorrido das operações de leitura/escrita, denominado ***elapsed time***
- Note que se for possível iniciar os arquivos f_1 e f_2 já organizados em rodadas de tamanho maior, um número menor de passagens pelos arquivos será necessário
- Isso pode ser feito com uma passagem inicial pelos dados lendo, ordenando internamente na memória principal, e re-escrevendo no arquivo, grupos com o máximo número cabível de registros
- Assim, esgota-se completamente o potencial de ordenação em memória interna e aplica-se ordenação externa apenas em arquivos cujas rodadas superam a capacidade interna de memória

Otimização

- Por exemplo, supondo que temos um arquivo com 1 milhão de registros e que podemos ordenar em memória interna um número máximo de 10.000 registros
- Podemos ler, ordenar internamente e re-escrever o arquivo em dois arquivos f_1 e f_2 iniciais ordenados em rodadas de 10.000 registros
- ...

Otimização

- Por exemplo, supondo que temos um arquivo com 1 milhão de registros e que podemos ordenar em memória interna um número máximo de 10.000 registros
- Podemos ler, ordenar internamente e re-escrever o arquivo em dois arquivos f_1 e f_2 iniciais ordenados em rodadas de 10.000 registros
  - Cada arquivo de origem contendo 50 rodadas
- Nesse caso, apenas **7 passagens** adicionais pelos dados são suficientes, uma vez que $10.000 \times 2^7 = 1.280.000 > 1$ milhão
- Com rodadas iniciais unitárias, **20 passagens** seriam necessárias

Sistemática de Leitura

- A leitura de um novo bloco deve ser realizada sempre que um buffer de entrada se esgota, a partir do arquivo de origem correspondente
  - Para saber de antemão qual será esse arquivo, basta comparar a maior chave do último bloco lido de cada um dos arquivos
  - Dado que os blocos são subconjuntos de rodadas, e portanto são ordenados, a maior chave do bloco é aquela do seu último registro
  - O bloco com a **menor maior chave** será sempre o primeiro a se esgotar, e o próximo bloco deve ser lido do arquivo correspondente

Sistemática de Leitura

- Deve-se apenas tomar o cuidado para não intercalar registros de rodadas diferentes lidos em um mesmo bloco
- Para isso, basta controlar o tamanho da rodada corrente e o no. de registros processados de cada um dos arquivos de origem

Comparação de Desempenho

- **Exemplo:** Ordenação de um arquivo de 40Gb em disco, contendo 40.000.000 de regs. de 1Kb, sendo 1Gb de RAM disponível para trabalho
  - RAM comporta 1.000.000 de registros (1Kb cada)
- Esgotando a capacidade de ordenação em memória RAM, conseguimos gerar 2 arquivos com 20.000.000 registros cada, ordenados em rodadas de 1.000.000
  - Cada arq. de origem com 20 rodadas de 1.000.000 registros
  - Número de passagens?

Comparação de Desempenho

- O número de passagens necessárias é tal que:
  - $1.000.000 \times 2^i > 40.000.000 \Rightarrow i > \log_2 40 \Rightarrow \boxed{i = 6}$
- Assumindo que as leituras e escritas se dão em blocos de 1/40 Gb, isto é, $b = 25.000$ registros
  - Número de acessos?

Comparação de Desempenho

- O número de passagens necessárias é tal que:
  - $1.000.000 \times 2^i > 40.000.000 \Rightarrow i > \log_2 40 \Rightarrow \boxed{i = 6}$
- Assumindo que as leituras e escritas se dão em blocos de 1/40 Gb, isto é, b = 25.000 registros, e lembrando que o número total de **acessos** é $(2\ n\ i)/b$, tem-se:
  - **Total de acessos: 19.200**
- Mas podemos otimizar o uso da RAM disponível em 3 buffers de 330Mb, o que implica b = 330.000 regs.
  - **Total de acessos: 1.454**
  - Mas no. de **seeks** por **acesso** é potencialmente maior...

Comparação de Desempenho

- Por outro lado, sabemos que a ordenação desse mesmo arquivo através de **merging 40-way** de 40 arquivos com 1.000.000 de registros cada (ordenados em RAM) requer 1600 **acessos** para leitura
- Sabemos ainda que cada um desses acessos presume a leitura de 1/40Gb, i.e., um bloco com 25.000 regs.
- Assumindo que a escrita é feita em blocos do mesmo tamanho, tem-se mais 1600 acessos de escrita
- **Total de Acessos: 3.200**

Comparação de Desempenho

- Tem-se então o no. de **acessos** estimado em:
  - **3200** de 25Mb para **Merging Multi-Way**; e
  - **1454** de 330Mb para **Merge-Sort Externo**...
- Merge-Sort Externo poderia ser melhor ?
  - Múltiplos dispositivos de memória secundária
  - ...

Intercalação Multi-Dispositivos

- O modelo de ordenação externa visto anteriormente assume que se dispõe de um único canal para L/E em um único dispositivo
- O processamento interno se dá usualmente de forma quase instantânea quando comparado ao tempo de acesso, que representa praticamente todo o tempo de ordenação
- Uma primeira forma de evitar esse gargalo (*bottleneck*) é utilizar 2 dispositivos independentes, um para leitura e outro para escrita
- Nesse caso, pode-se ler e escrever em paralelo, o que representa um ganho de 2 vezes no tempo

Intercalação Multi-Dispositivos

- De forma mais geral, quando se dispõe de $2m$ dispositivos, pode-se utilizar m dispositivos para leitura e m dispositivos para escrita
- Para tanto, Merge-Sort Externo deve ser modificado
- Ao invés de 2 arquivos de origem (leitura) e 2 arquivos de destino (escrita) se alternando nessas funções, passa-se a trabalhar com m arquivos de origem e m arquivos de destino em $2m$ dispositivos
- Se usarmos múltiplos buffers de L/E independentes em memória interna, pode-se reduzir significativamente os tempos de acesso

Intercalação Multi-Dispositivos

- Segundo (Aho, Hopcroft & Ullman, 1983), a redução do tempo total t de L/E pode ser de até $2m$ vezes, ou seja, t pode ser reduzido para algo da ordem de $t/2m$
  - OBS: essa estimativa simplificada é otimista...
- Número de passagens?

Intercalação Multi-Dispositivos

- Segundo (Aho, Hopcroft & Ullman, 1983), a redução do tempo total t de L/E pode ser de até $2m$ vezes, ou seja, t pode ser reduzido para algo da ordem de $t/2m$
  - OBS: essa estimativa simplificada é otimista...
- Adicionalmente, o no. de passagens requeridas é também reduzido
- De fato, após cada passagem o tamanho da rodada aumenta em m vezes. Com rodadas iniciais unitárias, são necessárias i passagens, onde $m^i \geq n$, sendo n o número de registros
- Logo, $O(\log_m n) = O(\log n / \log m)$ passagens são suficientes
- Isso significa uma redução da ordem de $\log m$ passagens

Intercalação Multi-Dispositivos

- Como exemplo, lembramos que Merge-Sort Externo demanda em torno de 1450 acessos de 330Mb para ordenar um arquivo de 40Gb composto de registros de 1Kb quando se dispõe de 1Gb de RAM
  - 6 passagens pelos arquivos
- Com m = 3...

Intercalação Multi-Dispositivos

- Como exemplo, lembramos que Merge-Sort Externo demanda em torno de 1450 acessos de 330Mb para ordenar um arquivo de 40Gb composto de registros de 1Kb quando se dispõe de 1Gb de RAM
  - 6 passagens pelos arquivos
- Com m = 3, $6/\log_2(3) = 4$ passagens são suficientes
  - porém, isso demanda 6 buffers, ou seja, blocos de 165Mb
  - isso implica em torno de 1940 acessos de 165Mb
    - mas acessos em parte paralelizados

Intercalação Multi-Dispositivos

- Exemplo: Ordenar por ordem alfabética das chaves o seguinte arquivo com 22 registros:

  I N T E R C A L A C A O B A L A N C E A D A

- Assumindo que temos $2m$ = 6 fitas magnéticas independentes e que a memória primária só tem espaço para 3 registros, tem-se após a organização inicial nas 3 fitas de origem:

  Fita 1: I N T | A C O | A D E
  Fita 2: C E R | A B L | A
  Fita 3: A A L | A C N

Intercalação Multi-Dispositivos

- Na primeira passagem pelas fitas as rodadas de tamanho 3 são intercaladas em rodadas de tamanho 9 direcionadas de forma alternada para as fitas de saída:

  Fita 4: A A C E I L N R T
  Fita 5: A A A B C C L N O
  Fita 6: A A D E

- Mais uma passagem (rodadas de 27 registros) e o processo termina com todos os registros ordenados em uma única fita. As duas outras ficam vazias já que temos apenas 22 registros

  Fita 1: A A A A A A A B C C C D E E I L L N N O R T

Intercalação Multi-Dispositivos

- O procedimento anterior é chamado **Intercalação Balanceada Multi-Caminhos** (*multi-way balanced merging*) via Múltiplos Dispositivos
- Como em qualquer intercalação multi-caminhos, existe um preço computacional a pagar: a busca interna pela menor dentre m chaves
- Em outras palavras, cada passo do procedimento de fusão não demanda mais apenas a comparação de apenas 2 chaves
- É possível encontrar a menor chave via:
  - $m - 1$ comparações $\Rightarrow$ tempo $O(m)$;
  - Implementação de uma fila de prioridade (heap) $\Rightarrow$ tempo $O(\log m)$

Intercalação Multi-Dispositivos

- Nivio Ziviani sugere que a abordagem de heap passa a compensar a partir de $m \geq 8$ (Ziviani, 2004), pág. 131.
- De qualquer forma, os ganhos computacionais decorrentes do aumento no número m de dispositivos de armazenamento externo tenderiam a inexistir a partir de um dado limite
- A partir desse limite o gargalo passaria a ser o tempo de fusão realizado em memória interna, que cresce na ordem $O(m)$ ou $O(\log m)$, dependendo da abordagem adotada para busca da menor chave

Intercalação Multi-Dispositivos

- Uma pergunta que surge é porque precisamos de $2m$ dispositivos externos para uma intercalação de m caminhos
- A razão essencial é que os dispositivos de leitura e escrita se alternam nessas funções a cada passagem
- Uma alternativa seria utilizar apenas $m + 1$ dispositivos, m para leitura e 1 para escrita.
-

Intercalação Multi-Dispositivos

- Uma pergunta que surge é porque precisamos de $2m$ dispositivos externos para uma intercalação de m caminhos
- A razão essencial é que os dispositivos de leitura e escrita se alternam nessas funções a cada passagem
- Uma alternativa seria utilizar apenas $m + 1$ dispositivos, m para leitura e 1 para escrita.
  - Bastaria fazer uma passada adicional pelo arq. de escrita para redistribuir as rodadas pelos m arqs. de leitura
- No entanto, existe um método que elimina a necessidade da passagem adicional para redistribuição de rodadas quando se utiliza $m + 1$ dispositivos: **Intercalação Polifásica**

Intercalação Polifásica

- A idéia é revezar um a um os dispositivos como sendo o dispositivo de saída
- Especificamente, faz-se com que a cada passagem um dos dispositivos de leitura se esvazie, tornando-se o próximo dispositivo de escrita
- Nessa abordagem, as rodadas **não** precisam ser distribuídas de forma balanceada entre os dispositivos

Intercalação Polifásica

- Exemplo: ($m = 2$; $n = 34$)

após a passagem	f_1	f_2	f_3
inicial	13 (1)	21 (1)	∅
1	∅	8 (1)	13 (2)
2	8 (3)	∅	5 (2)
3	3 (3)	5 (5)	∅
4	∅	2 (5)	3 (8)
5	2 (13)	∅	1 (8)
6	1 (13)	1 (21)	∅
7	∅	∅	1 (34)

Notação das células: No. de rodadas (No. registros por rodada)

- A cada passagem, as rodadas do dispositivo com menor número de rodadas são intercaladas com um número igual de rodadas dos demais dispositivos
- O dispositivo correspondente se torna vazio (próximo dispositivo de saída)

Intercalação Polifásica

- Exemplo: ($m = 2$; $n = 34$)

após a passagem	f_1	f_2	f_3
inicial	13 (1)	21 (1)	∅
1	∅	8 (1)	13 (2)
2	8 (3)	∅	5 (2)
3	3 (3)	5 (5)	∅
4	∅	2 (5)	3 (8)
5	2 (13)	∅	1 (8)
6	1 (13)	1 (21)	∅
7	∅	∅	1 (34)

Notação das células: No. de rodadas (No. registros por rodada)

- A distribuição inicial das rodadas pode ser obtida de baixo para cima: tome o maior número de rodadas de uma linha, faça-o igual a zero na linha de cima, e adicione-o aos demais números na mesma linha para obter os números correspondentes da linha acima.

Intercalação Polifásica

- Exemplo:
  ($m = 2$; $n = 34$)

após a passagem	f_1	f_2	f_3
inicial	13 (1)	21 (1)	Ø
1	Ø	8 (1)	13 (2)
2	8 (3)	Ø	5 (2)
3	3 (3)	5 (5)	Ø
4	Ø	2 (5)	3 (8)
5	2 (13)	Ø	1 (8)
6	1 (13)	1 (21)	Ø
7	Ø	Ø	1 (34)

Notação das células: No. de rodadas (No. registros por rodada)

- Nesse caso, exceto pela ausência do primeiro valor unitário, a soma total dos números de rodadas em cada passagem segue uma seqüência de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Intercalação Polifásica

- Exemplo:
  ($m = 3$)

f_1	f_2	f_3	f_4
Ø	13	11	7
7	6	4	Ø
3	2	Ø	4
1	Ø	2	2
Ø	1	1	1
1	Ø	Ø	Ø

Células: No. de rodadas

- Conforme ilustrado acima, a idéia pode ser generalizada para qualquer m

Intercalação Polifásica

- Quanto menos iterações "de-baixo-para-cima" melhor, já que cada iteração representa uma passagem completa pelos arqs.
  - Quanto menos iterações "de-baixo-para-cima", menores são os números de rodadas e maiores são os tamanhos dessas rodadas
  - O número de iterações deve ser determinado de forma que as rodadas iniciais sejam do tamanho máximo determinado pela capacidade de pré-ordenação dessas rodadas em memória interna
- No exemplo anterior para $m = 2$, se 5 registros podem ser ordenados em RAM, podemos iniciar já na 3a passagem

Bibliografia

- **A. V. Aho, J. E. Hopcroft & J. Ullman, *Data Structures and Algorithms*, Addison Wesley, 1983.**
- **N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Ed., 2004.**
- **R. Sedgewick, *Algorithms in C: Parts 1-4: Fundamentals, Data Structures, Sorting, Searching*, Addison-Wesley, 3rd Ed., 1997.**