Algoritmos e Estruturas de Dados II

Ordenação Externa II

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

2

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

3

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**

4

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

5

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)

6

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**

7

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

8

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

9

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

10

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					

11

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																						

12

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 ???

13

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 ~= $4(n/2)$ ~= $2n$
- 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 $O(n \log n)$ mesmo assumindo que $n >> b$

14

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

15

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

16

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

17

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

18

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

19

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 330Kb, o que implica b = 330.000 regs.
  - **Total de acessos: 1.454**
  - Mas no. de **seeks** por **acesso** é potencialmente maior...

20

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**

21

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
  - Próxima aula...

22

Exercícios

- Escolha uma seqüência de 31 números não ordenados e ilustre passo-a-passo, em detalhes, a ordenação dessa seqüência de números através de Merge-Sort Externo, iniciando com rodadas de tamanho unitário
- Para exercitar o algoritmo, repita o exercício anterior para outras seqüências de diferentes tamanhos e valores
- Seja um arquivo de 100Gb composto de registros de 500 bytes cada. Supondo que se dispõe de 500Kb de RAM disponível, qual o máximo no. de registros em cada rodada inicial que se pode constituir para minimizar o no. de passagens requeridas pelo algoritmo Merge-Sort Externo? Qual é esse no. de passagens? Quantos acessos de L/E são realizados pelo algoritmo?

23

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.**

24