

Algoritmos e Estruturas de Dados II

Processamento Conseqüencial & Ordenação Externa I

Prof. Debora Medeiros

Baseado nos slides dos Profs. Ricardo Campello, Thiago Pardo, Leandro C. Cintra e M.C.F. de Oliveira

1

Operações Conseqüenciais

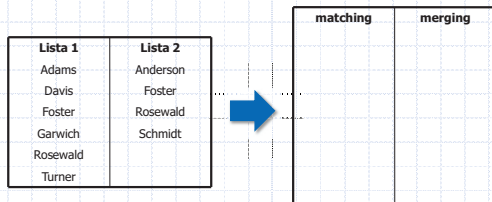
- ◆ Processamento coordenado (simultâneo) de duas ou mais "listas" de entradas seqüenciais, produzindo uma única lista como saída

2

Operações Conseqüenciais

◆ Exemplos Típicos:

- **merging** (união) ou **matching** (intersecção) de dois ou mais conjuntos de registros mantidos em arquivos separados e ordenados por chave
- Exemplo (só chaves dos registros):

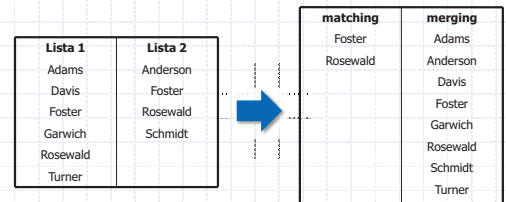


3

Operações Conseqüenciais

◆ Exemplos Típicos:

- **merging** (união) ou **matching** (intersecção) de dois ou mais conjuntos de registros mantidos em arquivos separados e ordenados por chave
- Exemplo (só chaves dos registros):



4

Operações Conseqüenciais

◆ Merging:

- lê uma entrada de cada lista/arquivo e as compara
- se ambas são iguais, copia a entrada na lista/arquivo de saída e avança para a próxima entrada em cada lista
- se uma das entradas é menor:
 - ♦ copia essa entrada na saída e lê a próxima entrada da respectiva lista
- retorna ao segundo passo até que ambas as listas terminem

5

Merging: algoritmo

```
inicializa() //abre arquivos lista1 e lista2, cria arquivo saída e inicializa
variável existem_mais_nomes como verdade
```

```
lê nome1 da lista1
lê nome2 da lista2
```

```
enquanto (existem_mais_nomes) faça
  se (nome1 < nome2)
    escreva nome1 em saída
    lê nome1 de lista1
  senão se (nome1 > nome2)
    escreva nome2 em saída
    lê nome2 de lista2
  senão
    escreva nome1 em saída
    lê nome1 de lista1
    lê nome2 de lista2
```

```
fim-enquanto
```

```
finaliza() //fecha arquivos
```

6

Merging

Exemplo:

```
#include <stdio.h>
#define VALOR_ALTO 100000000
bool MAIS_CHAVES_EXISTEM = true;

void main(void) {
    int chave1 = 0, chave2 = 0;
    FILE *pt_arq_in1, *pt_arq_in2, *pt_arq_out;
    pt_arq_in1 = fopen("arq_in1.txt", "r");
    pt_arq_in2 = fopen("arq_in2.txt", "r");
    pt_arq_out = fopen("arq_out.txt", "w");
    chave1 = input(pt_arq_in1, chave2);
    chave2 = input(pt_arq_in2, chave1);
    while (MAIS_CHAVES_EXISTEM) {
        if (chave1 < chave2) {
            fprintf(pt_arq_out, "%d\n", chave1);
            chave1 = input(pt_arq_in1, chave2);
        } else if (chave2 < chave1) {
            fprintf(pt_arq_out, "%d\n", chave2);
            chave2 = input(pt_arq_in2, chave1);
        } else {
            fprintf(pt_arq_out, "%d\n", chave1);
            chave1 = input(pt_arq_in1, chave2);
            chave2 = input(pt_arq_in2, chave1);
        }
    }
    // continua...
}
```

Nota: Implementação acima assume que listas são simplesmente de números inteiros (arquivos texto com um número por linha)

7

Merging

Exemplo:

(continuação)

```
int input(FILE *pt_arq, int CHAVE_OUTRA_LISTA) {
    int flag;
    int chave;
    flag = fscanf(pt_arq, "%d", &chave);
    if (flag == EOF) {
        chave = VALOR_ALTO; // lista atual finalizada
        if (CHAVE_OUTRA_LISTA == VALOR_ALTO)
            MAIS_CHAVES_EXISTEM = false; // duas listas finalizadas
    }
    return chave;
}
```

8

Cuidados com o merging

Garantir que os chaves estão ordenadas

Quando uma lista acabar, a outra lista deve continuar a ser processada

9

Operações Consequenciais

Matching:

- lê uma entrada de cada lista/arquivo e as compara
- se ambas são iguais, copia a entrada na lista/arquivo de saída e avança para a próxima entrada em cada lista
- se uma das entradas é menor:
 - lê a próxima entrada da respectiva lista
- retorna ao segundo passo até que uma das listas termine

10

Matching: algoritmo

inicializa() //abre arquivos lista1 e lista2, cria arquivo saída e inicializa variável existem_mais_nomes como verdade

lê nome1 da lista1
lê nome2 da lista2

enquanto (existem_mais_nomes) faça

se (nome1 < nome2)

lê nome1 de lista1

senão se (nome1 > nome2)

lê nome2 de lista2

senão

escreve nome1 em saída

lê nome1 de lista1

lê nome2 de lista2

fim-enquanto

finaliza() //fecha arquivos

11

Matching

Exemplo:

```
#include <stdio.h>
bool MAIS_CHAVES_EXISTEM = true;

void main(void) {
    int chave1 = 0, chave2 = 0;
    FILE *pt_arq_in1, *pt_arq_in2, *pt_arq_out;
    pt_arq_in1 = fopen("arq_in1.txt", "r");
    pt_arq_in2 = fopen("arq_in2.txt", "r");
    pt_arq_out = fopen("arq_out.txt", "w");
    chave1 = input(pt_arq_in1);
    chave2 = input(pt_arq_in2);
    while (MAIS_CHAVES_EXISTEM) {
        if (chave1 < chave2) {
            chave1 = input(pt_arq_in1);
        } else if (chave2 < chave1) {
            chave2 = input(pt_arq_in2);
        } else {
            fprintf(pt_arq_out, "%d\n", chave1);
            chave1 = input(pt_arq_in1);
            chave2 = input(pt_arq_in2);
        }
    }
    // continua...
}
```

Nota: Implementação acima assume que listas são simplesmente de números inteiros (arquivos texto com um número por linha)

12

Matching

◆ Exemplo:

(continuação)

```
int input(FILE *pt_arq) {
    int flag;
    int chave;
    flag = fscanf(pt_arq, "%d", &chave);
    if (flag == EOF) MAIS_CHAVES_EXISTEM = false; // lista finalizada
    return chave;
}
```

13

Cuidados com o matching

- ◆ Garantir que as chaves estão ordenadas
- ◆ Quando uma lista acabar, encerra-se o processo

14

Matching

◆ Nota:

- Os códigos anteriores fazem a leitura e escrita de registros um a um, o que pode tornar as operações ineficientes
- Na prática, para reduzir o no. de acessos ao dispositivo externo, pode-se lidar com leitura/escrita de **blocos** de registros
 - ◆ Registros são lidos em blocos para **buffers** de memória em RAM
 - 1 buffer para cada lista/arquivo de entrada
 - ◆ Operações são executadas nos buffers em RAM
 - Cada buffer é recarregado sempre que finalizado
 - ◆ Resultado é escrito em um buffer de saída também em RAM
 - Buffer é descarregado no arquivo de saída sempre que cheio

15

Operações Conseqüenciais

◆ Multi-Way:

- Operações conseqüenciais, tais como merging e matching, não precisam se restringir a operar em apenas duas listas (2-way)
- Versões **k-way** são obtidas como generalizações de **2-way**
- Muda apenas a comparação:
 - ◆ Avança-se para a próxima entrada em toda lista cuja entrada corrente for mínima dentre todas as entradas correntes
- Exemplo 3-way:

Lista 1	Lista 2	Lista 3	Matching
Adams	Anderson	Adams	
Davis	Foster	Foster	
Foster	Rosewald	Rosewald	
Garwich	Schmidt	Schmidt	
Rosewald		Turner	
Turner			

Operações Conseqüenciais

◆ Multi-Way:

- Operações conseqüenciais, tais como merging e matching, não precisam se restringir a operar em apenas duas listas (2-way)
- Versões **k-way** são obtidas como generalizações de **2-way**
- Muda apenas a comparação:
 - ◆ Avança-se para a próxima entrada em toda lista cuja entrada corrente for mínima dentre todas as entradas correntes

■ Exemplo 3-way:

Lista 1	Lista 2	Lista 3	Matching
Adams	Anderson	Adams	Foster
Davis	Foster	Foster	Rosewald
Foster	Rosewald	Rosewald	
Garwich	Schmidt	Schmidt	
Rosewald		Turner	
Turner			

Ordenação Externa

◆ Até então...

18

Ordenação Externa

◆ Até então...

- Keysorting
 - ♦ Muitos seeks
 - ♦ Limitado ao número de chaves que poderíamos colocar na memória
 - ♦ Não serve para arquivos realmente grandes, de fato

◆ Ordenação Externa via Multi-Way Merging

- Como isso pode ser útil?

19

Ordenação Externa

◆ Ordenação Externa via Multi-Way Merging:

- Pode-se utilizar uma modificação da operação consecutiva de merging multi-way para ordenar um arquivo grande em disco
- **Idéia:**
 - ♦ Carrega-se toda a RAM disponível com parte do arquivo
 - ♦ Ordena-se os registros em RAM com um algoritmo *in-place*
 - ♦ Escreve-se os registros ordenados em um arquivo separado
 - ♦ Repete-se os passos acima até encerrar o arquivo original
 - Se a RAM disponível comporta $(1/k * \text{no. de regs. do arq. original})$, então ter-se-ão **k** arquivos ordenados
 - ♦ Aplica-se então multi-way merging nos arquivos ordenados

20

Ordenação Externa

◆ Multi-Way Merging nos Arquivos Ordenados:

- Conforme descrito anteriormente, para maximizar a eficiência da operação de merging, opera-se com L/E de blocos em RAM
- RAM disponível é sub-dividida em **k** buffers de entrada
 - ♦ 1 buffer para cada um dos **k** arquivos ordenados
 - ♦ "RAM disponível" já desconta uma porção reservada para buffer de saída
- Buffers são preenchidos com $1/k$ regs. dos respectivos arquivos
- Merging é realizado em RAM
 - ♦ Cada buffer de entrada é recarregado sempre que vazio
 - ♦ Buffer de saída é descarregado no arquivo de saída sempre que cheio
- Como cada arquivo ordenado é exatamente do tamanho da RAM disponível, processo termina após **k** procedimentos acima

Ordenação Externa

◆ Exemplo (40-way):

- Arquivo com 40Gb (40.000.000 de registros de 1Kb)
- 1GB de RAM disponível
 - ♦ RAM comporta 1.000.000 de registros
- Com 40 ordenações em RAM produz-se 40 arquivos ordenados
 - ♦ Cada arquivo com 1.000.000 de registros do arquivo original
- Com 40 merges em RAM produz-se um único arquivo ordenado
 - ♦ RAM é dividida em 40 blocos de 1/40 Gb
- Quantos seeks?

22

Ordenação Externa

◆ Desempenho:

◆ Apenas a fase de multi-way merging do exemplo anterior necessita, no mínimo, **1600 seeks** no disco:

- Mesmo que cada bloco de registros possa ser lido com um único seek, serão necessários **k = 40** seeks para cada arquivo
- Como são 40 arquivos, tem-se ao menos $40 \times 40 = 1600$ seeks
 - ♦ Fora os seeks para escrita do arquivo de saída

23

Ordenação Externa

◆ Estratégias para Melhorar o Desempenho:

- ...

24

Ordenação Externa

◆ Estratégias para Melhorar o Desempenho:

- Merging em múltiplos passos (a seguir)

25

Ordenação Externa

◆ Merging em Múltiplos Passos:

- No exemplo anterior, ao invés de fazer o merging dos 40 arquivos em um único de uma só vez, será que é vantajoso fazer o merging em múltiplas etapas ? Por exemplo:
 - ◆ 1ª etapa: 5 mergings de 8 arquivos, produzindo 5 arquivos
 - ...
 - ◆ 2ª etapa: 1 merging dos 5 arquivos, produzindo 1 arquivo
 - ...

26

Ordenação Externa

◆ Merging em Múltiplos Passos:

- No exemplo anterior, ao invés de fazer o merging dos 40 arquivos em um único de uma só vez, será que é vantajoso fazer o merging em múltiplas etapas ? Por exemplo:
 - ◆ 1ª etapa: 5 mergings de 8 arquivos, produzindo 5 arquivos
 - cada merging demanda $8 \times 8 = 64$ seeks
 - total de $64 \times 5 = 320$ seeks
 - ◆ 2ª etapa: 1 merging dos 5 arquivos, produzindo 1 arquivo
 - RAM disponível comporta 1/8 de cada arquivo
 - logo, cada um dos 5 blocos em RAM comporta 1/40 de cada arquivo
 - 40 seeks por arquivo \Rightarrow total de **200 seeks**

27

Ordenação Externa

◆ Merging em Múltiplos Passos:

- A estratégia de 2 passos no exemplo resultou em um total de $320+200 = 520$ seeks, contra **1600** da estratégia anterior
 - ◆ No entanto, esses valores são apenas **limitantes inferiores**:
 - Consideram que um bloco de registros sempre pode ser lido em um único seek, independente de seu tamanho
 - ◆ Mesmo com essa simplificação, em cada caso os seeks estarão associados à transmissão de quantidades diferentes de registros:
 - As análises anteriores ignoram o tempo de transmissão de registros, dentre vários outros aspectos que dependem de hardware e software
- Escolha de uma determinada estratégia em um projeto deve considerar em detalhes todos esses aspectos...

28

Bibliografia

- ◆ M. J. Folk and B. Zoellick, *File Structures: A Conceptual Toolkit*, Addison Wesley, 1987.

29