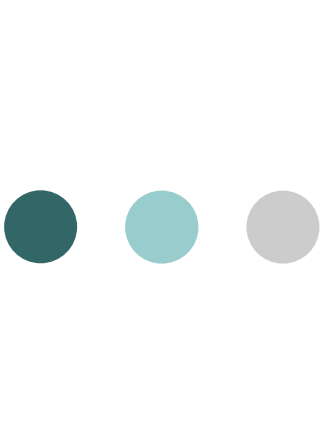


Organização de Arquivos

Processamento Co-seqüencial e Ordenação

Programa

- Introdução
- Operações básicas sobre arquivos
- Armazenagem secundária
- Conceitos básicos sobre estrutura de arquivo
- Organizando arquivos para desempenho
- Indexação
- Processamento co-seqüencial e ordenação
- B-Tree e outras organizações em árvores
- B+Tree e acesso seqüencial indexado
- Hashing
- Hashing estendido

Objetivos

- Descrever uma atividade de processamento, frequentemente utilizada, chamada de processamento co-sequencial.
- Apresentar um modelo genérico para implementar todas as variações de processamento co-sequencial.
- Ilustrar o uso do modelo para resolver diferentes problemas de processamento co-sequencial.
- Apresentar o algoritmo “K-way merge” (intercalação múltipla).
- Apresentar o heapsort como uma abordagem para sorting em RAM
- Mostrar como merging (intercalação) fornece a base para ordenação de arquivos grandes.

Processamento Co-sequencial

- Objetivo
  - Processamento coordenado de duas ou mais listas sequenciais para produzir uma única lista como saída (procedimento geral para *match* e *merge*).
- As listas de entrada são:
  - Ordenadas por chave.
  - Não possuem itens (registros) duplicados por chave.
  - “Processados paralelamente”.
- A lista de saída fica:
  - Ordenada por chave.
  - Sem registros duplicados por chave.

Processamento Co-sequencial

- Tipos de listas resultantes:
  - Interseção (Matching) dos itens das listas.
    - A lista de saída é formada só por itens que ocorrem em todas as listas de entrada.
    - Ex: Indexação (combinação de chaves)
  - União/Intercalação (Merging) dos itens das listas.
    - A lista de saída é formada por todos os itens das listas de entrada, sem duplicação de chave.
    - Os itens são intercalados por ordem de chave.

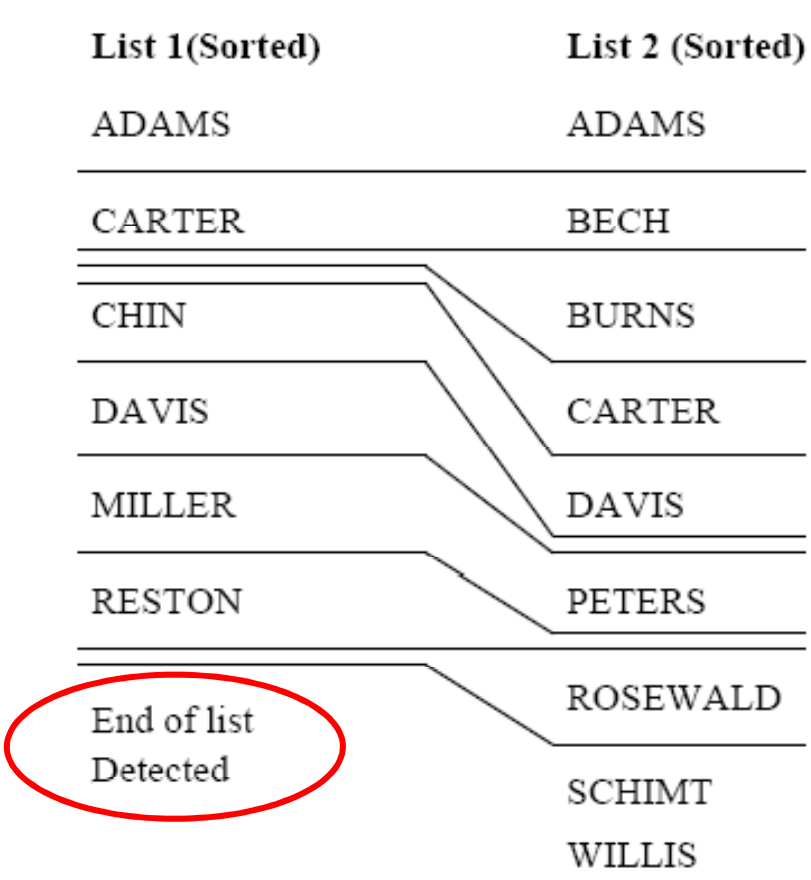
Processamento Co-sequencial

- O ALGORITMO deve conter:
  - procedimento de iniciação
    - abertura dos arquivos.
    - variável de teste de fim dos arquivos.
    - teste de sequência de chaves para cada arquivo.
  - procedimento de sincronização dos arquivos.
  - condições de parada.
  - reconhecimento de erro para cada arquivo de entrada:
    - registros duplicados por chave.
    - registros não ordenados por chave.

Processamento Co-sequencial
Interseção (Matching)

- Suponha que desejamos obter as chaves que aparecem em duas listas. Podemos utilizar o seguinte algoritmo:
  1. Se a chave da lista 1 é menor que a chave da lista 2, lemos a seguinte chave da lista 1.
  2. Se a chave da lista 1 é maior que a chave da lista 2, lemos a seguinte chave da lista 2.
  3. Se as chaves são iguais, colocamos na lista de saída e lemos a chave seguinte de ambas listas de entrada.
  4. Repetimos os passos 1 a 3 até chegar no final de uma das listas.

Processamento Co-sequencial Interseção (Matching)



Processamento Co-sequencial
Interseção (Matching)

- Procedimento de leitura e verificação:

```
Program Interseção
        type Tchave = ... ,
               Treg = record chave: Tchave, ... end;

        var reg1, reg2: Treg ;
              chave1, chave2: Tchave;

        procedure Leia_lista (arq; var reg; var chave);
        {     read (arq, reg);
              if (eof(arq) ) then existe_registro = false
              else
                    if reg.chave <= chave then    // não ordenado ou duplicação: erro!
                              { print "arquivo"+arq+" fora de ordem";
                                 pare
                              };
              chave = reg.chave
        };
```

Para quando encontra o fim da lista 1 ou da lista 2.

Reconhecimento de erro.

Processamento Co-sequencial
Interseção (Matching)

- Procedimento de iniciação:

```
{
    abra arq1;
    abra arq2;
    crie arqSaída;
    chave1 = valor_minimo;
    chave2 = valor_minimo;
    existe_registro:= true;
    Leia_lista (arq1, reg1, chave1);
    Leia_lista (arq2, reg2, chave2);
```

Abre os arquivos/listas.

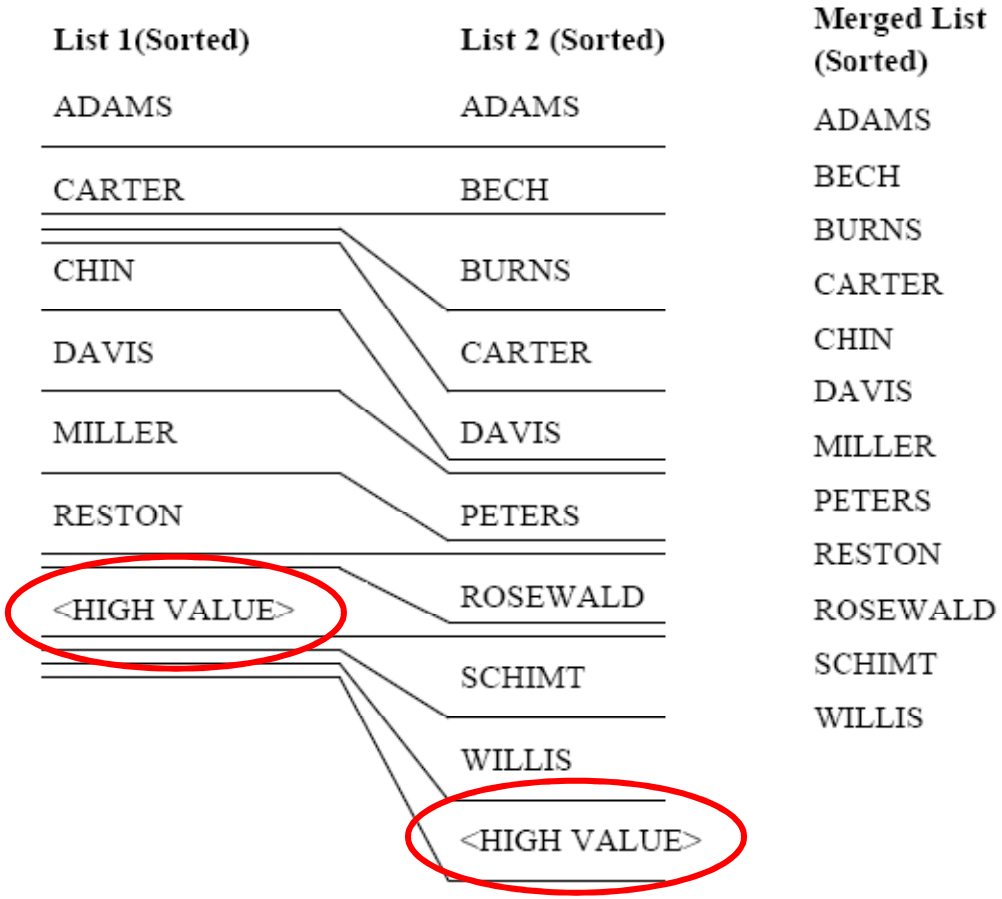
Valor_minimo é uma constante. É o menor valor que a chave poderia assumir.

Processamento Co-sequencial Interseção (Matching)

- Procedimento de sincronização dos arquivos:

```
        while existe_registro do            // Condição de parada.
              {
              if (reg1.chave < reg2.chave) then
                          Leia_lista (arq1, reg1, chave1);      // Lê próximo item da lista 1
              else
                          if (reg1.chave > reg2.chave) then
                             Leia_lista (arq2, reg2, chave2);       // Lê próximo item da lista 2
                          else
                          {
                          escreve (reg1, arqsaída);       // Grava o elemento comum às listas
                          Leia_lista (arq1, reg1, chave1);
                          Leia_lista (arq2, reg2, chave2);
                          };
              };
}.                                                    // Fim do programa Interseção.
```

Processamento Co-sequencial União/Intercalação (Merging)



Processamento Co-sequencial
União/Intercalação (Merging)

```
Program Intercalacao
    type Tchave = ... ;
         Treg = record chave: Tchave, ... end;
    var reg1, reg2: Treg ;
        chave1, chave2: Tchave;

    procedure Leia_lista (arq; var reg; var chave);
    {   read (arq, reg);
        if (eof(arq)) then
        {     reg.chave = valor_máximo;
              if acabou then existe_registro = false ;
              acabou = true;
         }
         else
              if reg.chave <= chave then                    // não ordenado: erro!
                   { print "arquivo" + arq + " fora de ordem"; pare };
         chave := reg.chave
};
```

Garante que o conteúdo restante da outra lista, será jogado na lista de saída.

Quando terminar uma lista apenas (acabou ainda é false), existe_registro ainda será true.

Reconhecimento de erro.

Processamento Co-sequencial
União/Intercalação (Merging)

- Procedimento de iniciação:

```
{
    abra arq1;
    abra arq2;
    crie arqSaída;
    chave1 = valor_minimo;
    chave2 = valor_minimo;
    existe_registro = true;
    acabou = false;
    Leia_lista (arq1, reg1, chave1);
    Leia_lista (arq2, reg2, chave2);
```

Abre os arquivos/listas.

Valor_minimo é uma constante. É o menor valor que a chave poderia assumir.

Processamento Co-sequencial
União/Intercalação (Merging)

- Procedimento de sincronização dos arquivos:

```
while existe_registro do
{
    if (reg1.chave < reg2.chave) then
    {    escreve (arqsaida, reg1) ; Leia_lista (arq1, reg1, chave1)
    }
    else
        if (reg2.chave < reg1.chave) then
        {    escreve (arqsaida, reg2); Leia_lista (arq2, reg2, chave2)
         };
        else
        {    escreve (arqsaída, reg1,);
             Leia_lista (arq1, reg1, chave1); Leia_lista (arq2, reg2, chave2)
         } ;
    };
}.
```

Grava item da lista 1 na lista de saída, e lê o próximo item da lista 1.

Grava item da lista 2 na lista de saída, e lê o próximo item da lista 2.

Termina execução quando chegar ao fim das duas listas.

Itens iguais. Grava na lista de saída, e lê o próximo item da lista 1 e lista 2.

Processamento Co-sequencial
União/Intercalação (Merging)

- VALORES CONSTANTES:
  - *valor_mínimo*
    - é o menor valor que o campo de chave poderia assumir.
  - *valor_máximo*
    - é o maior valor que o campo de chave poderia assumir.
  - *valor_mínimo* e *valor_máximo*
    - não ocorrem como valor de chave de nenhum dos registros dos arquivos de entrada.

Processamento Co-sequencial União/Intercalação (Merging)

- *valor_minimo*
  - O uso de tal constante permite:
    - executar *Leia_lista* sem cair na condição de erro:

```
if reg.chave <= chave then
    { print “arquivo” + arq + “ fora de ordem”; pare };
```

    - onde
      - chave é parâmetro formal.
      - O parâmetro real (chave1 ou chave2) é iniciado com *valor_minimo.*

Processamento Co-sequencial União/Intercalação (Merging)

- *valor_maximo*
  - O uso de tal constante permite:
    - que seja atribuído o *valor_máximo* a regx.chave do arquivo x que findou.
    - que o algoritmo intercalação leia e transfira os demais registros do arquivo arqy que não findou para o arquivo de saída, x ≠ y.

```
if (reg1.chave < reg2.chave) then
{     escreve (arqsaida, reg1)
      Leia_lista (arq1, reg1, chave1)  }
else
                  if (reg2.chave < reg1.chave) then
      {    escreve (arqsaida, reg2);
           Leia_lista (arq2, reg2, chave2) ; }
```

Processamento Co-sequencial União/Intercalação (Merging)

- Variável *acabou*:
  - É iniciada com *false*.
  - Recebe *true* quando um dos arquivos chega ao fim.
    - A chave do registro desse arquivo fica com valor_máximo

      (regx.chave = valor_máximo)
- Variável existe_*registro:*
  - É iniciada com *true*.
  - Recebe *false* quando os dois arquivos chegam ao fim.
    - Encerra o loop (while existe_registro...)
    - Termina a intercalação

Processamento Co-sequencial União/Intercalação (Merging)

- Reduzindo o número de comparações
  - Podemos escrever um algoritmo similar com menos comparações sem usar a variável *valor_maximo*.
  - O critério de parada do loop seria:
    - Quando atingir o final de umas das listas.
  - Neste caso precisaríamos de uma finalização

```
while not(eof(arq1))
{
    escreve (arqsaida, reg1); Leia_lista (arq1, reg1, chave1);
}
while not(eof(arq2))
{
    escreve (arqsaida, reg2); Leia_lista (arq2, reg2, chave2);
}
```

Exemplo

- Livro-Razão: um livro contendo contas às quais débitos e créditos são lançados a partir de livros originais.

- PROBLEMA:
  - Desenvolver um programa para o livro-razão (atualização + listagem), como parte de um sistema de contabilidade.

Exemplo

- Dois arquivos estão envolvidos neste processo:
  - **Arquivo Mestre: arquivo do livro-razão**
    - Resumo mensal do balanço de cada uma das contas
  - **Arquivo de Transação: arquivo do diário**
    - Contém as transações mensais que são colocadas no livro-razão.
- Uma vez que o arquivo diário está completo para um determinado mês, ele deve ser lançado no livro-razão.
- Esse lançamento envolve associar cada transação à sua conta no livro-razão.

Exemplo

- Amostra do livro-razão:

Account Number	Account Title	Jan	Feb	Mar	Apr
101	checking account #1	1032.00	2114.00	5219.00	
102	checking account #2	543.00	3094.17	1321.20	
510	auto expense	195.00	307.00	501.00	
540	office expense	57.00	105.25	138.37	
550	rent	500.00	1000.00	1500.00	
⋮	⋮	⋮	⋮	⋮	⋮

Exemplo

- Amostra dos lançamentos no diário:

Account Number	Check Num-ber	Date	Description	Debit/Credit
101	1271	April 2, 01	Auto expense	- 79.00
510	1271	April 2, 01	Tune-up	79.00
101	1272	April 3, 01	Rent	- 500.00
550	1272	April 3, 01	Rent for April	500.00
102	670	April 4, 01	Office expense	- 32.00
540	670	April 4, 01	Printer cartridge	32.00
101	1273	April 5, 01	Auto expense	- 31.00
510	1273	April 5, 01	Oil change	31.00
⋮	⋮	⋮	⋮	⋮

Exemplo

- Exemplo da listagem do livro-razão:

```
101 Checking account #1
    1271    | April 2, 01 | Auto expense     -  79.00
    1272    | April 3, 01 | Rent             - 500.00
    1273    | April 5, 01 | Auto expense     -  31.00
   Prev. Bal.:    5,219.00       New Bal.:   4,609.00

102 Checking account #2
      ⋮
510 Auto expense
      ⋮
540 Office expense
      ⋮
550 Rent
      ⋮
```

Exemplo

- Como implementar o processo de lançamento no livro-razão?
  - Usar o número da conta como uma **chave** para relacionar as transações do diário aos registros do livro-razão.
  - Ordenar o arquivo diário pela chave (nro da conta)
  - Processar o livro-razão e o diário **co-sequencialmente** (processamos as duas listas sequencialmente em paralelo).

Exemplo

- Tarefas a serem realizadas:
  - Atualizar o arquivo livro-razão com o saldo correto para cada conta do mês corrente.
  - Produzir uma listagem como no exemplo (lista as contas com seu saldo atual, e também as respectivas transações do diário daquele mês).

Exemplo

- Do ponto de vista das contas do livro-razão:
  - Merging (até mesmo as contas que não “match” vão para a listagem – existem contas no livro sem entrada no diário)
- Do ponto de vista das contas do diário:
  - Matching (contas que não “match” constituem um erro – existem contas no diário não listadas no livro)

O método de lançamento é uma combinação de merging e matching.

Exemplo

- O Algoritmo:

Item(1) = sempre armazena o reg corrente do arquivo mestre
Item(2) = sempre armazena o reg corrente do arquivo transações

- Leia o primeiro registro do arq mestre
- Imprima a linha título para a primeira conta
- Leia o primeiro registro do arquivo transações
- Enquanto (existirem reg no mestre OU reg no transações) faça {
    Se item(1) < item(2) então {
        Encerre este registro do mestre
            - Imprima o balanço da conta, atualiza o reg mestre
            - Leia o próximo registro do arq mestre
            - Se leitura com sucesso, então imprima a linha título
                para a nova conta }

Exemplo

- O Algoritmo:

```
Se item(1) = item(2) então {
     Transação “match” o mestre
                - Adiciona o valor da transação no balanço da conta para o novo mês
                - Imprima a descrição da transação
                - Leia o próximo reg de transação }
Se item(1) > item(2) então {
     Transação sem registro no mestre
                - Imprima uma msg de erro
                - Leia o próximo reg de transação }
}
```

Próxima Aula...

- Apresentar o algoritmo “K-way merge” (intercalação múltipla).
- Apresentar o heapsort como uma abordagem para sorting em RAM
- Mostrar como merging (intercalação) fornece a base para ordenação de arquivos grandes.