SCC 503 - ALGORITMOS E ESTRUTURAS DE DADOS II

Arquivos - Processamento Co-sequencial

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

*Material baseado em aulas dos professores: Elaine Parros Machado de Souza, Gustavo Batista, Robson Cordeiro, Moacir Ponti Jr., Maria Cristina Oliveira e Cristina Ciferri.

Processamento Co-sequencial

- Processamento coordenado (simultâneo) de duas ou mais “listas” de entradas sequenciais, produzindo uma única lista como saída
- 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):

Entrada (arquivos ordenados)

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

Saída (arquivo ordenado)

matching	merging
Foster	Adams
Rosewald	Anderson
	Davis
	Foster
	Garwich
	Rosewald
	Schmidt
	Turner

Exemplo (só chaves dos registros)

Algoritmo - Merging

```
inicializa()     //abre arquivos lista1 e lista2, cria arquivo saída
ecinicializa variável existem_mais_chaves como verdade
lê chave1 da lista1
lê chave2 da lista2
enquanto (existem_mais_chaves) faça
    se (chave1 < chave2)
        escreve chave1 em saída
        lê chave1 de lista1
    senão se (chave1 > chave2)
        escreve chave2 em saída
        lê chave2 de lista2
    senão
        escreve chave1 em saída
        lê chave1 de lista1
        lê chave2 de lista2
fim-enquanto
finaliza()   //fecha arquivos
```

Merging

- Pontos principais
  - Inicialização
    - Abrir arquivos e inicializar variáveis para processamento correto
  - Sincronização
    - Como ler a próxima chave e avançar adequadamente em cada lista

Merging

- Pontos principais
  - Condições de fim de arquivo
    - Quando uma lista acabar, continua-se a processar a outra, que pode ser copiada diretamente na saída
  - Reconhecimento de erros
    - Chaves duplicadas ou fora de ordem

Algoritmo - Matching

```
inicializa()     // abre arquivos lista1 e lista2, cria arquivo saída e
              // inicializa variável existem_mais_chaves como verdade
lê chave1 da lista1
lê chave2 da lista2
enquanto (existem_mais_chaves) faça
    se (chave1 < chave2)
        lê chave1 de lista1

    senão se (chave1 > chave2)
        lê chave2 de lista2

    senão
        escreve chave1 em saída
        lê chave1 de lista1
        lê chave2 de lista2
fim-enquanto
finaliza()   //fecha arquivos
```

Matching

- Pontos principais
  - Inicialização
  - Sincronização
  - Condições de fim de arquivo
    - Quando uma das listas acabar, encerra-se o processamento
  - Reconhecimento de erros

Operações Co-Sequenciais

Operações Co-Sequenciais

- Implementação mais direta e simples
  - Leitura e escrita de registros um a um pode tornar as operações ineficientes

Operações Co-Sequenciais

- ➔ Na prática, como é possível reduzir o número de acessos ao dispositivo externo?
  - ◆ 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 Co-Sequenciais

- ➔ Muti-Way:
  - Operações co-sequenciais, como merging e matching, não precisam se restringir a operar em apenas duas listas (2-way)
  - Versões k-way são generalizações de 2-way
  - Mudança nos algoritmos:
    - Procura a menor chave entre todas as listas
    - Lê nova chave da lista correspondente

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

Exemplo 3-way

Multi-Way Merging

- Pode-se utilizar uma modificação da operação co-sequencial de multi-way merging para ordenar um arquivo grande em disco
  - Merging como soma (chaves repetidas) e não como união
- Permite Ordenação externa!

Ordenação Externa

- Análises de complexidade dos métodos de ordenação tradicionais se preocupam basicamente com o tempo de execução
  - Modelo de ordenação interna
  - Complexidade computacional estimada em função da quantidade de operações (comparações, trocas, etc) feitas utilizando a memória principal (primária) da máquina

Ordenação Externa

- No entanto.... ordenação de uma base de dados muito grande, que não cabe na memória principal, requer um modelo diferente
  - Modelo de ordenação externa
    - Assume-se que os dados devem ser recuperados a partir de dispositivos externos
  - Ordenação em memória secundária

Ordenação Externa

- Acesso à memória secundária é muito mais lento
  - A maior preocupação é minimizar a quantidade de leituras e escritas

Ordenação Externa

- Dificuldade:
  - Projeto e análise dos métodos de ordenação externa dependem fortemente da tecnologia. Ex:
    - Acesso a dados em fitas magnéticas é sequencial e mais lento, enquanto em discos tem-se o acesso direto
- Em disco 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

Ordenação Externa

- Solução
  - A análise do problema de ordenação externa usualmente é baseada em um modelo simplificado, que abstrai ao máximo os detalhes tecnológicos

Ordenação Externa

- ➔ Basicamente, preocupa-se com a quantidade de operações envolvendo a transferência de blocos de registros (de tamanho fixo) entre as memórias primária e secundária
- ➔ Operações de leitura e escrita (L/E) ou de ACESSO

Ordenação Externa

- ➔ Ordenação Externa via Muti-Way Merging:
  - ◆ Merging como soma (chaves repetidas) e não como união

Ordenação Externa

1. Carrega-se toda a RAM disponível com parte do arquivo
2. Ordena-se os registros em RAM com um algoritmo in-place
3. Escreve-se os registros ordenados em um arquivo separado
4. Repete-se os passos acima até encerrar o arquivo original

   se a RAM disponível comporta (1/k) * nro. de regs. do arq. original temos **k arquivos ordenados**

5. Aplica-se multi-way merging nos arquivos ordenados

Ordenação Externa

- ➔ Multi-Way Merging nos Arquivos Ordenados:
  - ◆ Para maximizar a eficiência da operação de merging, opera-se com L/E de blocos em RAM
  - ◆ RAM disponível é sub-dividida em k + 1 buffers
    - 1 buffer para cada um dos k arquivos de entrada (listas ordenadas)
    - 1 buffer de saída

Ordenação Externa

- ➔ Multi-Way Merging nos Arquivos Ordenados:
  - ◆ Buffers de entrada são preenchidos com registros 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

Ordenação Externa

- ➔ Multi-Way Merging nos Arquivos Ordenados:
  - ◆ Buffers de entrada são preenchidos com registros 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

Ordenação Externa

- Ordenação Externa via Muti-Way Merging:
  - Algoritmo em duas fases: ordenação + merging
  - Alto custo computacional para arquivos muito grandes, mesmo com otimizações e paralelismo

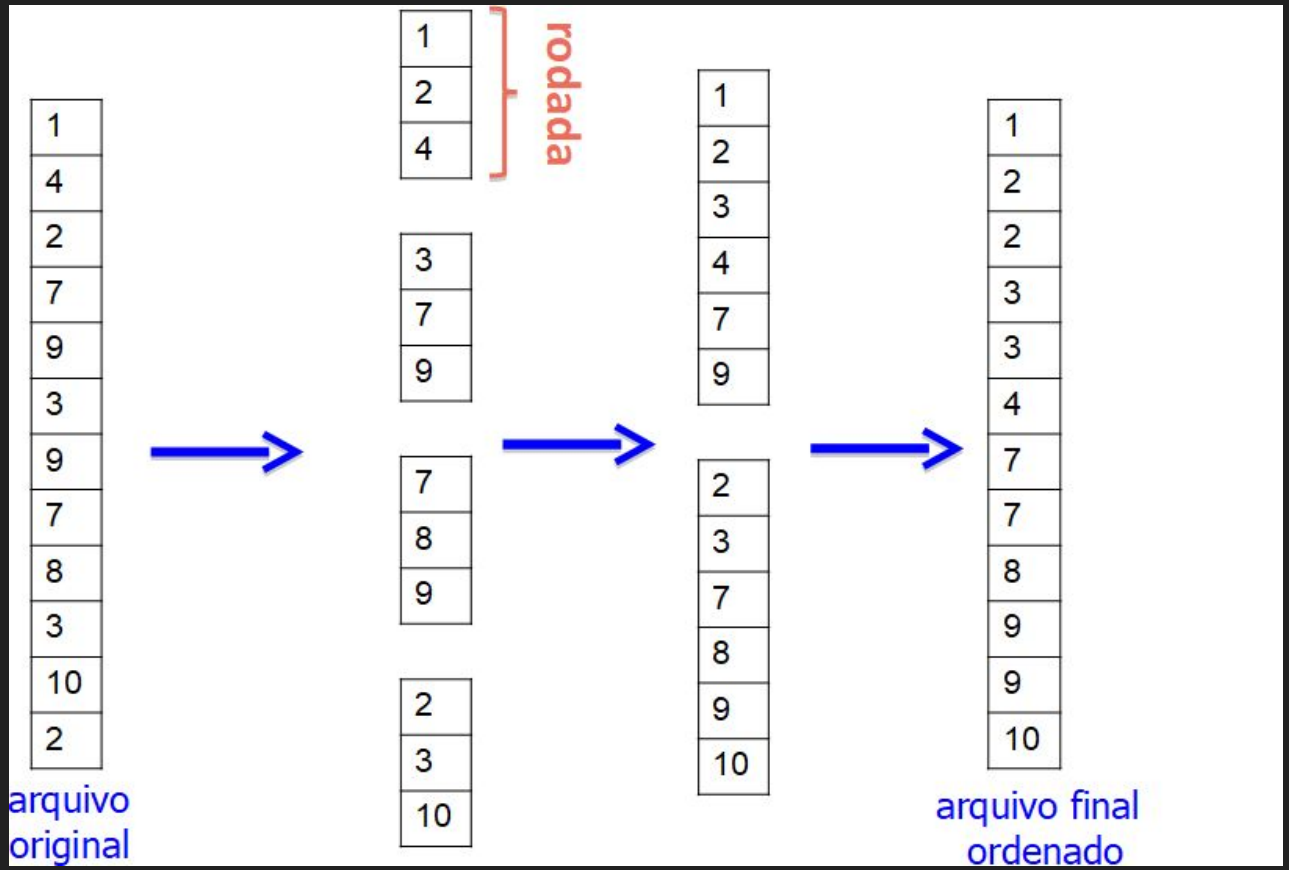
Ordenação Externa

- Ordenação externa por MERGE-SORT Externo
  - Melhor desempenho para arquivos muito grandes
  - Algoritmo clássico

Merge Sort Externo

Merge Sort Externo

- Ideia geral:
  - Criar subconjuntos ordenados do arquivo (chamados de rodadas – do inglês run)
  - Realizar o merge dessas rodadas e criar rodadas ordenadas de tamanho maior
  - Repetir até ordenar ordenar o arquivo todo (rodada do tamanho do arquivo)



Merge Sort Externo

Merge Sort Externo

- Versão básica do algoritmo
  - Opera com 4 arquivos armazenados em um único disco (além do arquivo original que pretende-se ordenar)
  - Registros são lidos de 2 arquivos de origem e reescritos de forma parcialmente ordenada em 2 arquivos de destino

Merge Sort Externo

- Versão básica do algoritmo
  - Os arquivos de origem e destino se alternam nas sucessivas iterações do algoritmo.
  - Inicialmente, os arquivos de origem são organizados em rodadas de tamanho unitário

Merge Sort Externo

- Versão básica do algoritmo
  - Registros são lidos sequencialmente, um a um, de ambos os arquivos e intercalados
    - Algoritmo merging por rodadas

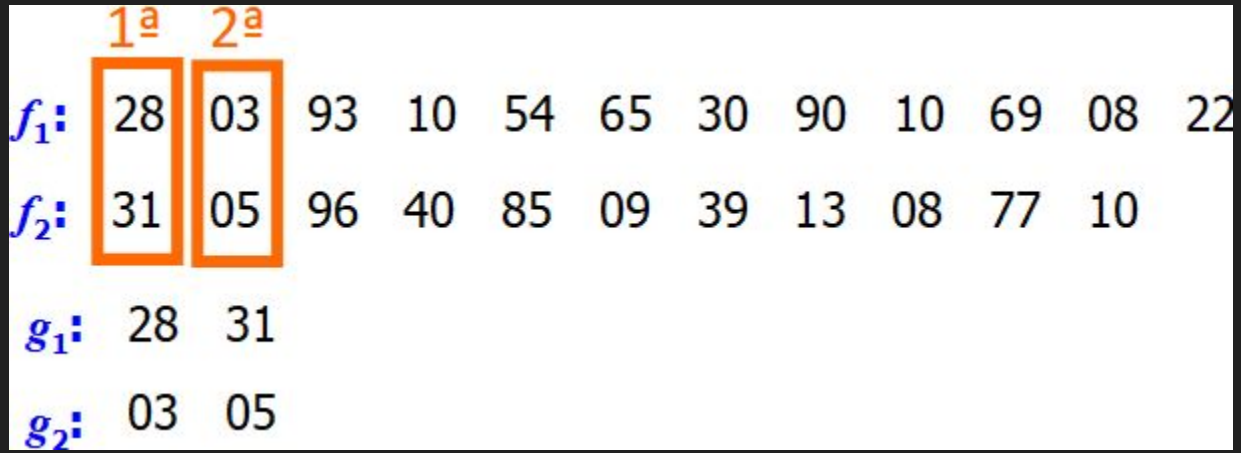
Merge Sort Externo

- Versão básica do algoritmo
  - Para cada rodada de f1 é realizado o merge com a rodada correspondente de f2, resultando numa rodada ordenada com o dobro do tamanho
    - Rodadas resultantes são armazenadas em arquivos de destino, g1 e g2 , alternadamente...

Início: arquivo original é dividido em 2 "arquivos de origem" - f_1 e f_2

rodada de tamanho 1

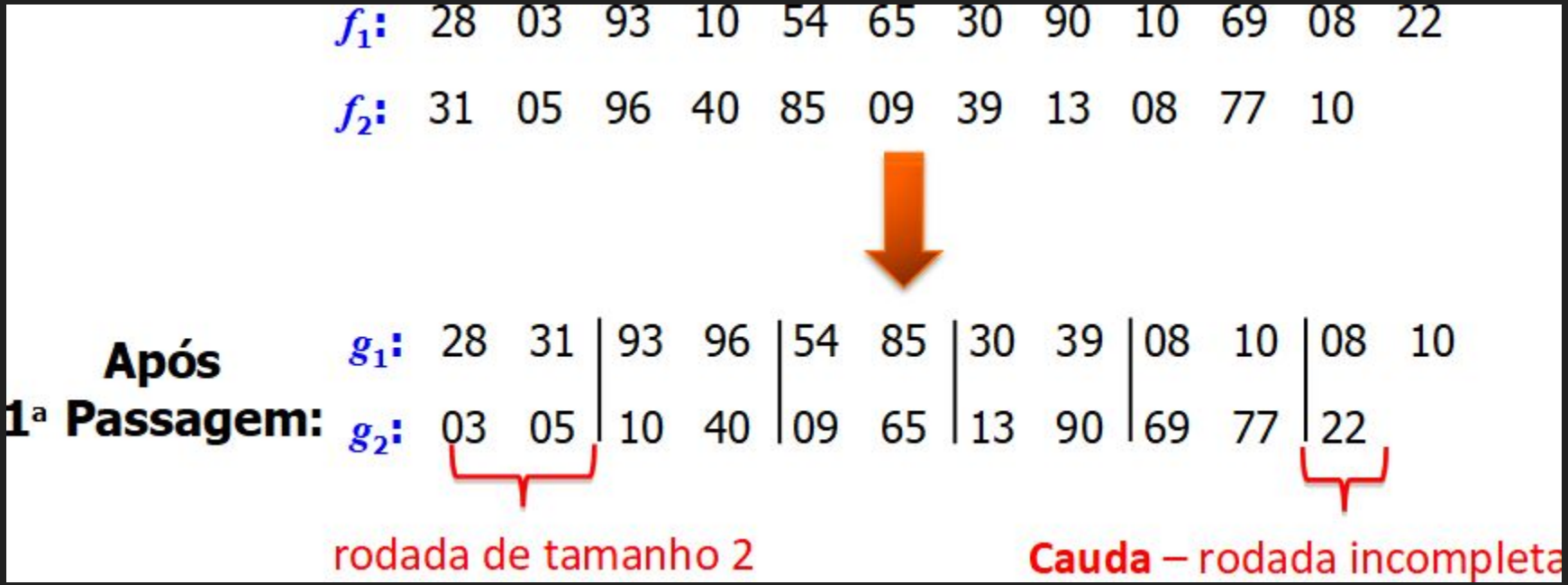
f_1:	28	03	93	10	54	65	30	90	10	69	08	22
f_2:	31	05	96	40	85	09	39	13	08	77	10	



Merge-Sort Externo (1)

Merge Sort Externo

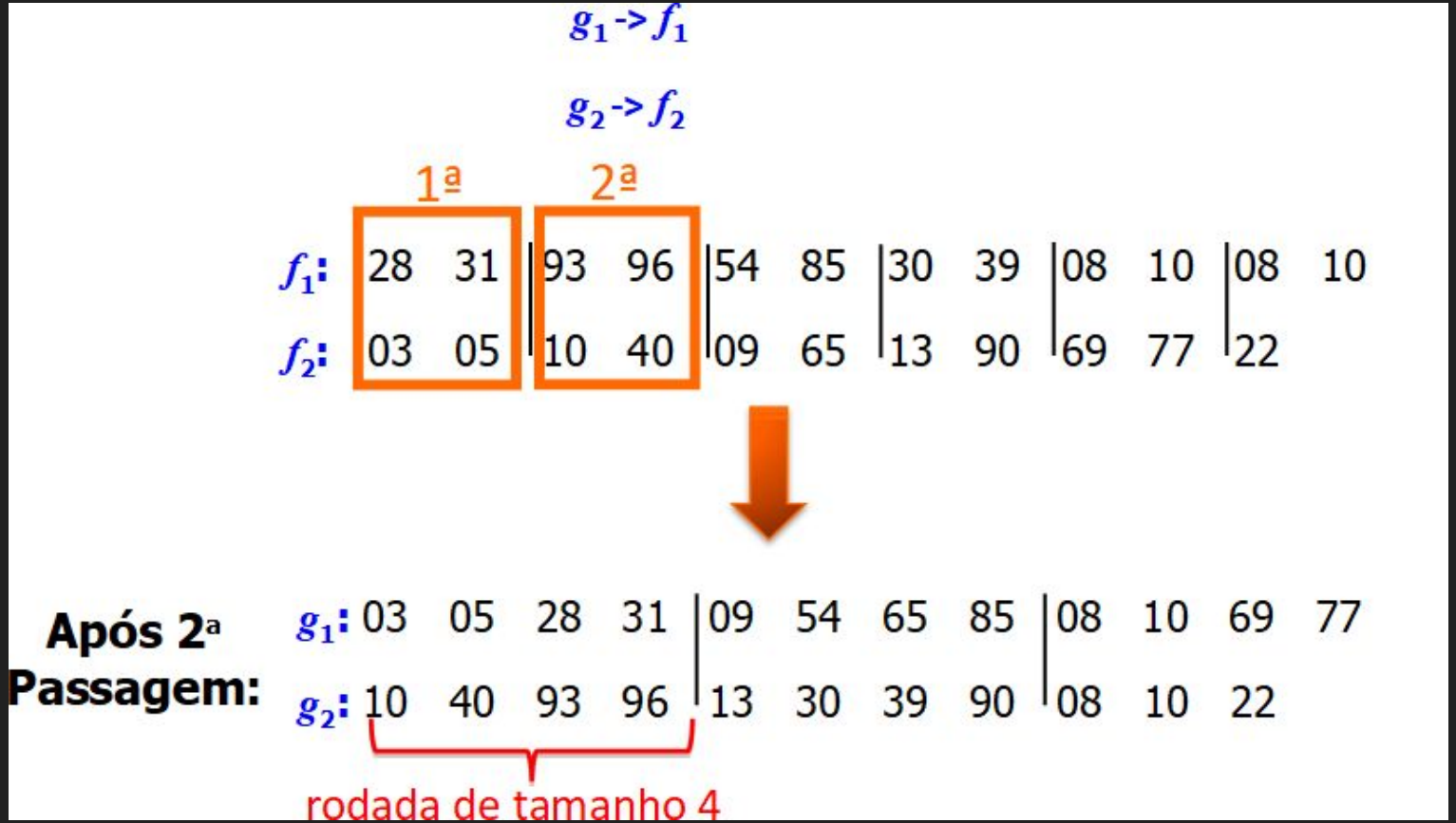
- Versão básica do algoritmo
  - Ao final de cada passagem completa pelos arquivos de origem, tem-se os arquivos de destino, g1 e g2 , organizados em rodadas com o dobro do tamanho dos arquivos de origem



Merge-Sort Externo (2)

Merge Sort Externo

- Versão básica do algoritmo
  - Arquivos g1 e g2 tornam-se então os arquivos de origem e o processo se repete
- O tamanho das rodadas dobra a cada passagem
  - Após i passagens o tamanho da rodada é $k = 2^i$, e quando k >= n (onde n é a quantidade total de registros a serem ordenados) tem-se:



Merge-Sort Externo (3)

$g_1 \rightarrow f_1$ f_1: 03 05 28 31 | 09 54 65 85 | 08 10 69 77

$g_2 \rightarrow f_2$ f_2: 10 40 93 96 | 13 30 39 90 | 08 10 22

3ª Passagem

g_1: 03 05 10 28 31 40 93 96 | 08 08 10 10 22 69 77

g_2: 09 13 30 39 54 65 85 90 |

4ª Passagem

g_1: 03 05 09 10 13 28 30 31 39 40 54 65 85 90 93 96

g_2: 08 08 10 10 22 69 77

5ª Passagem

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

g_2: ∅

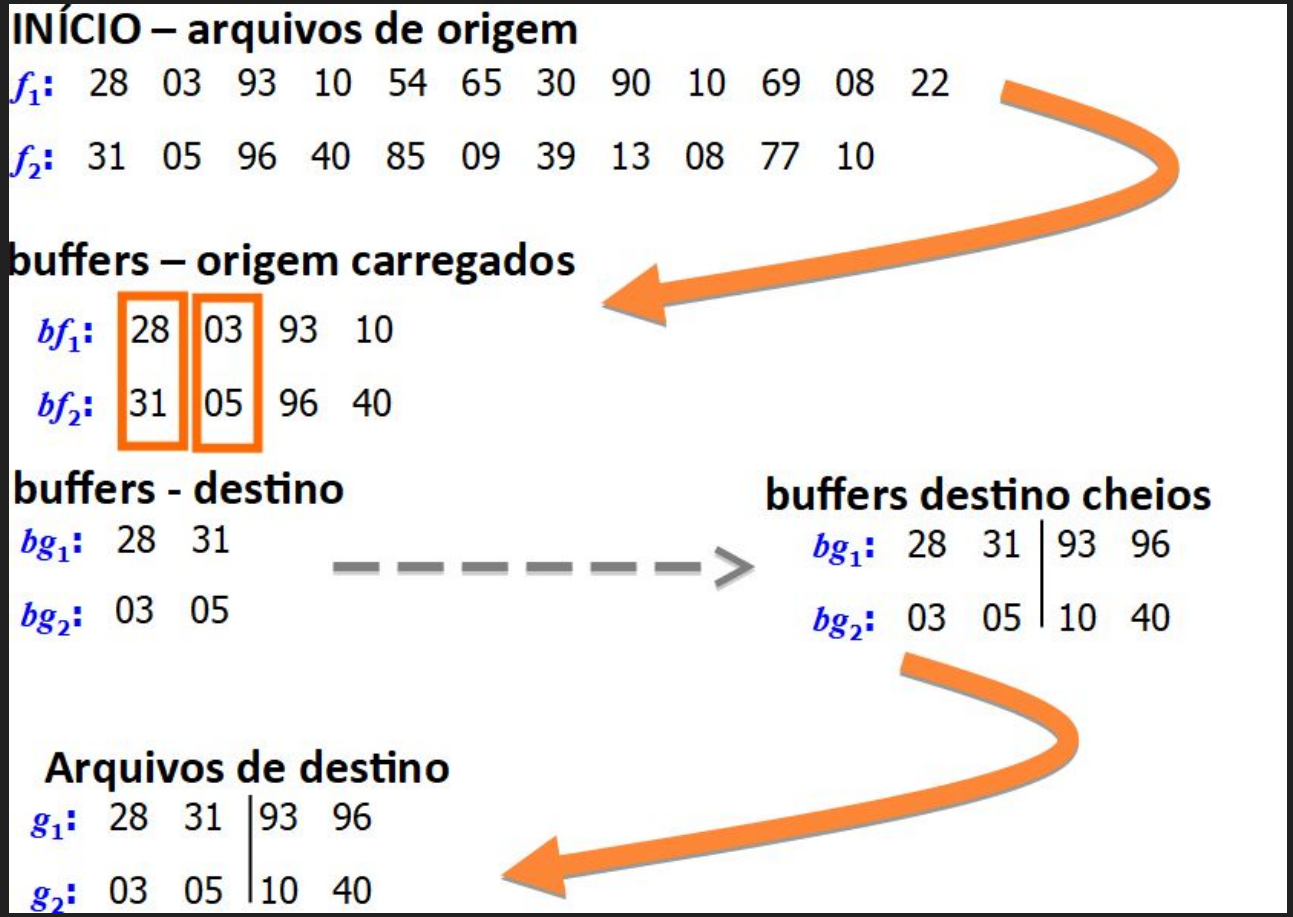
Desempenho

Desempenho (interno)

- Número i de passagens necessárias é tal que $2^i >= n$
- Logo, i >= log n passagens são suficientes:
  - Ou seja, número mínimo de passagens => Teto[log n]
- Como são n registros e o merging 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)
  - Mesma complexidade do Merge-Sort recursivo para ordenação interna

Desempenho (interno)

- Cada passagem requer leitura de 2 arquivos e escrita em 2 arquivos, cada um com aproximadamente n/2 registros
  - Número de acessos em cada passagem ~=4(n/2) ~=2n
- É possível reduzir o número de acessos?
  - Leituras e escritas podem ser feitas em blocos de registros, com uso de buffers em memória RAM
  - 4 buffers: 2 para arquivos de origem e 2 para arquivos de destino



Exemplo: buffers de tamanho 4

Desempenho (interno)

- Leituras e escritas em blocos de registros com uso de buffers em memória RAM
  - O número de leituras e escritas de blocos em cada passagem é em torno de 2n/b, onde b é o tamanho do bloco (capacidade de registros)
  - O número total de leituras e escritas de blocos em todo o processo de ordenação (ou seja, em log n passagens) é em torno de (2n log n)/b
    - ordem O(n log n) mesmo assumindo que n >> b

Desempenho (interno)

- Cuidado na utilização de buffers de origem
  - Não intercalar registros de rodadas diferentes lidos em um mesmo bloco
  - Controlar o tamanho da rodada corrente e o número de registros processados de cada um dos arquivos de origem

Otimização

Otimização

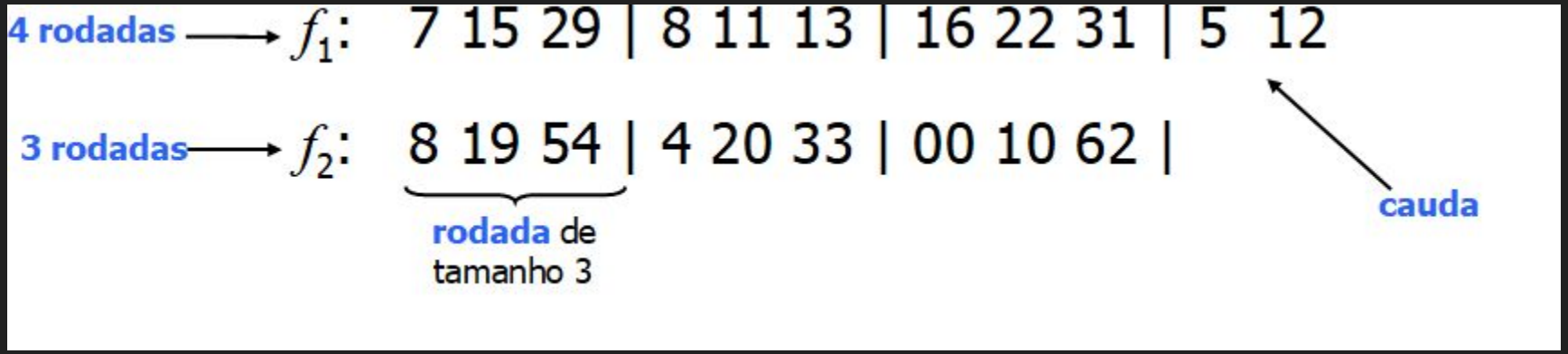
- ➔ Iniciar os arquivos f1 e f2 já organizados em rodadas de tamanho maior (ao invés de rodadas unitárias)
  - ◆ Resultado
    - Um número menor de passagens pelos arquivos

Otimização

- Na prática
  - Realizar uma passagem inicial no arquivo original para ler, ordenar em memória principal e re-escrever nos arquivos de origem (f1 e f2) subconjuntos com o maior número possível de registros ordenados (limitado pela memória primária disponível)
  - Utiliza-se o potencial de ordenação em memória interna
  - Aplica-se ordenação externa apenas em arquivos (de origem/destino) cujas rodadas superam a capacidade interna de memória

Otimização

- ➔ Observação:
  - ◆ O arquivo original é dividido nos arquivos f1 e f2 , de origem, com as seguintes propriedades:
    - A quantidade de rodadas de f1 e f2 (incluindo eventual cauda) difere em no máximo 1
    - No máximo um dentre f1 e f2 possui uma cauda



Exemplo de cauda entre f1 e f2 com rodada de tamanho 3

Otimização

- Ex: supondo 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 esse arquivo em dois arquivos f1 e f2 iniciais ordenados em rodadas de 10.000 registros
  - Cada arquivo de origem (f1 e f2) contendo 50 rodadas

Otimização

- ➔ Apenas 7 passagens adicionais pelos dados são suficientes, uma vez que $10.000 * 2^7 = 1.280.000 > 1$ milhão
  - ◆ OBS: Com rodadas iniciais unitárias, 20 passagens seriam necessárias

Referências

- A. V. Aho, J. E. Hopcroft & J. Ullman, Data Structures and Algorithms, Addison Wesley, 1983.
- M. J. Folk and B. Zoellick, File Structures: A Conceptual Toolkit, Addison Wesley, 1987.
- N. Ziviani, Projeto de Algoritmos, Thomson, 2a. Ed, 2004.