Algoritmos e Estruturas de Dados II

Processamento Conseqüencial & Ordenação Externa

Maria Cristina F. Oliveira

(Slides baseados em material anterior meu e do Prof. Ricardo Campello)

Ordenação de arquivos

- Parte I – Operações co-sequenciais
- Parte II – MergeSort externo

Operações Co-seqüenciais

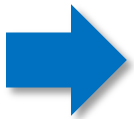
- Processamento coordenado (simultâneo) de duas ou mais “listas” de entrada sequenciais (ordenadas), gerando uma única lista como saída

Operações Co-seqüenciais

- Processamento coordenado (simultâneo) de duas ou mais "listas" de entrada sequenciais (ordenadas), gerando uma única lista como saída
- Exemplos
  - **merging** (intercalação/união) ou **matching** (intersecção) de dois conjuntos de registros mantidos em arquivos separados e ordenados por chave
  - Exemplo (só chaves dos registros)

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

entrada



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

saída

Operações Co-seqüenciais

Merging (intercalação)

1. lê uma entrada de cada lista/arquivo e as compara
2. se ambas as entradas são iguais, copia a entrada na lista/arquivo de saída e avança para a próxima entrada em cada lista
3. se uma das entradas é menor
   - copia essa entrada na saída e lê a próxima entrada da respectiva lista
4. retorna ao passo 2 até que ambas as listas terminem

Merging

código 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 compostas simplesmente de números inteiros (arquivos texto com um número por linha)

Merging

código 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;
}
```

Operações Co-seqüenciais

Matching (intersecção)

1. lê uma entrada de cada lista/arquivo e as compara
2. se ambas são iguais, copia a entrada na lista/arquivo de saída e avança para a próxima entrada em cada lista
3. se uma das entradas é menor
   - lê a próxima entrada da respectiva lista
4. retorna ao passo 2 até que uma das listas termine

Matching

código 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)

Matching

código 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;
}
```

Matching

- 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 fazer as operações de leitura/escrita em **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

Operações Co-seqüenciais

- **Multiway *merging*** (intercalação em múltiplas vias)
  - Operações co-seqüenciais como *merging* e *matching* não precisam se restringir a apenas duas listas (2-way, ou 2-vias)
  - Versões **k-way** são obtidas como generalizações de **2-way: m**uda apenas a comparação (compara k valores ao invés de 2)
    - Avança-se para a próxima entrada em toda lista cuja entrada corrente for mínima dentre todas as entradas correntes
  - Exemplo 3-vias

Lista 1	**Lista 2**	**Lista 3**	**Merging**
Adams	Anderson	Adams	Adams
Davis	Foster	Foster	Anderson
Foster	Rosewald	Rosewald	Davis
Garwich	Schmidt	Schmidt	Foster
Rosewald		Turner	Garwich
Turner			Rosewald
			…

Operações Co-seqüenciais

- **Multiway *merging*** (intercalação em múltiplas vias)
  - Operações co-seqüenciais como *merging* e *matching* não precisam se restringir a apenas duas listas (2-way, ou 2-vias)
  - Versões **k-way** são obtidas como generalizações de **2-way: m**uda apenas a comparação (compara k valores ao invés de 2)
    - Avança-se para a próxima entrada em toda lista cuja entrada corrente for mínima dentre todas as entradas correntes
  - Exemplo 3-vias

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

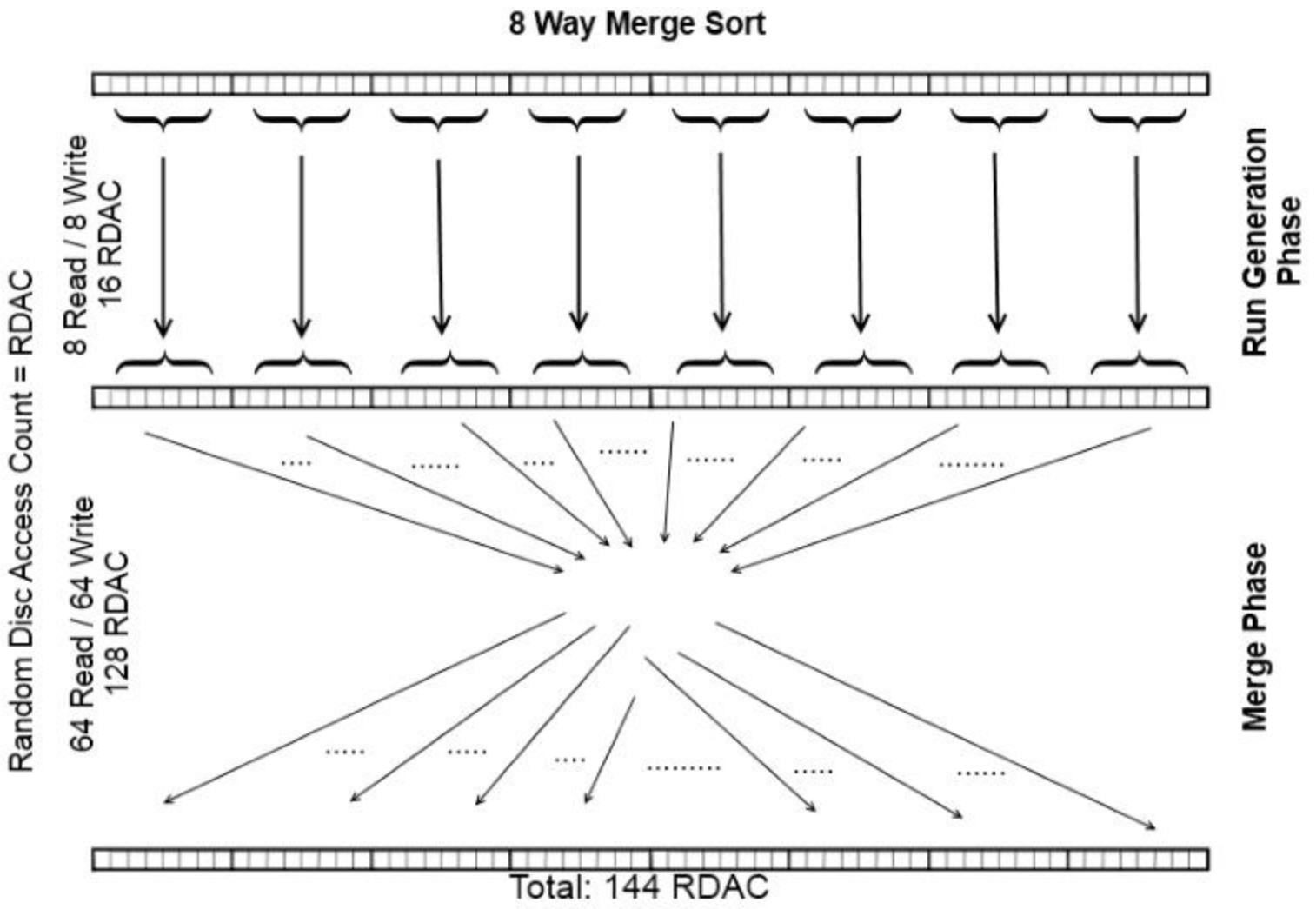
Ordenação Externa

- **Ordenação Externa via *Multiway Merging***
  - Pode-se modificar a operação coseqüencial de intercalação multifase para ordenar um arquivo grande, mantido em disco
  - **Ideia**
    - 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 esgotar o arquivo original
      - Se a RAM disponível comporta (1/k * no. de regs. do arq. original), essa etapa vai gerar **k** (sub-)arquivos ordenados (*runs*, ou rodadas)
    - Faz-se então uma intercalação multifase das rodadas, gerando o arquivo ordenado completo

Ordenação Externa

- **Intercalação em múltiplas vias das rodadas**
  - Como mencionado, opera-se com L/E de blocos em RAM para maximizar a eficiência da operação de *merging*
  - RAM disponível é sub-dividida em **k** buffers de entrada
    - 1 buffer para cada uma das **k** rodadas
    - "RAM disponível" já desconta uma porção reservada para o buffer de saída
  - Buffers são preenchidos com 1/**k** registros das respectivas rodadas
  - *Merging* é realizado em RAM
    - Cada buffer de entrada é recarregado sempre que vazio (leitura do disco)
    - Buffer de saída é descarregado no arquivo de saída sempre que cheio (escrita no disco)
  - Como cada rodada tem exatamente o tamanho da RAM disponível, o processo termina após **k** procedimentos acima

K-Way Merge Sort



Ordenação Externa

- **Exemplo (intercalação em 40-vias, ou 40-way *merge*)**
  - Arquivo com 40Gb (40.000.000 de registros de 1Kb)
  - 1GB de RAM disponível: comporta 1.000.000 de registros
  - Com 40 ordenações em RAM produz-se 40 rodadas (sub-arquivos ordenados)
    - Cada rodada com 1.000.000 registros (1/40 * 40.000.000 de registros do arquivo original
  - Com uma intercalação em 40-vias em RAM produz-se um único arquivo ordenado
    - RAM é dividida em 40 buffers de 1/40 Gb (mais o buffer de saída)

Ordenação Externa

- **Desempenho**
- Apenas a fase de *multiway merging* do exemplo anterior necessita, no mínimo, **1.600 seeks** no disco!
  - Porque?

Ordenação Externa

- **Desempenho**
- Apenas a fase de *multiway merging* do exemplo anterior necessita, no mínimo, **1.600 seeks** no disco!
  - Porque?
  - Porque cada leitura de um bloco de uma rodada requer um *seek*!
  - Sem contar que cada escrita dos blocos no arquivo de saída também requer um *seek*!

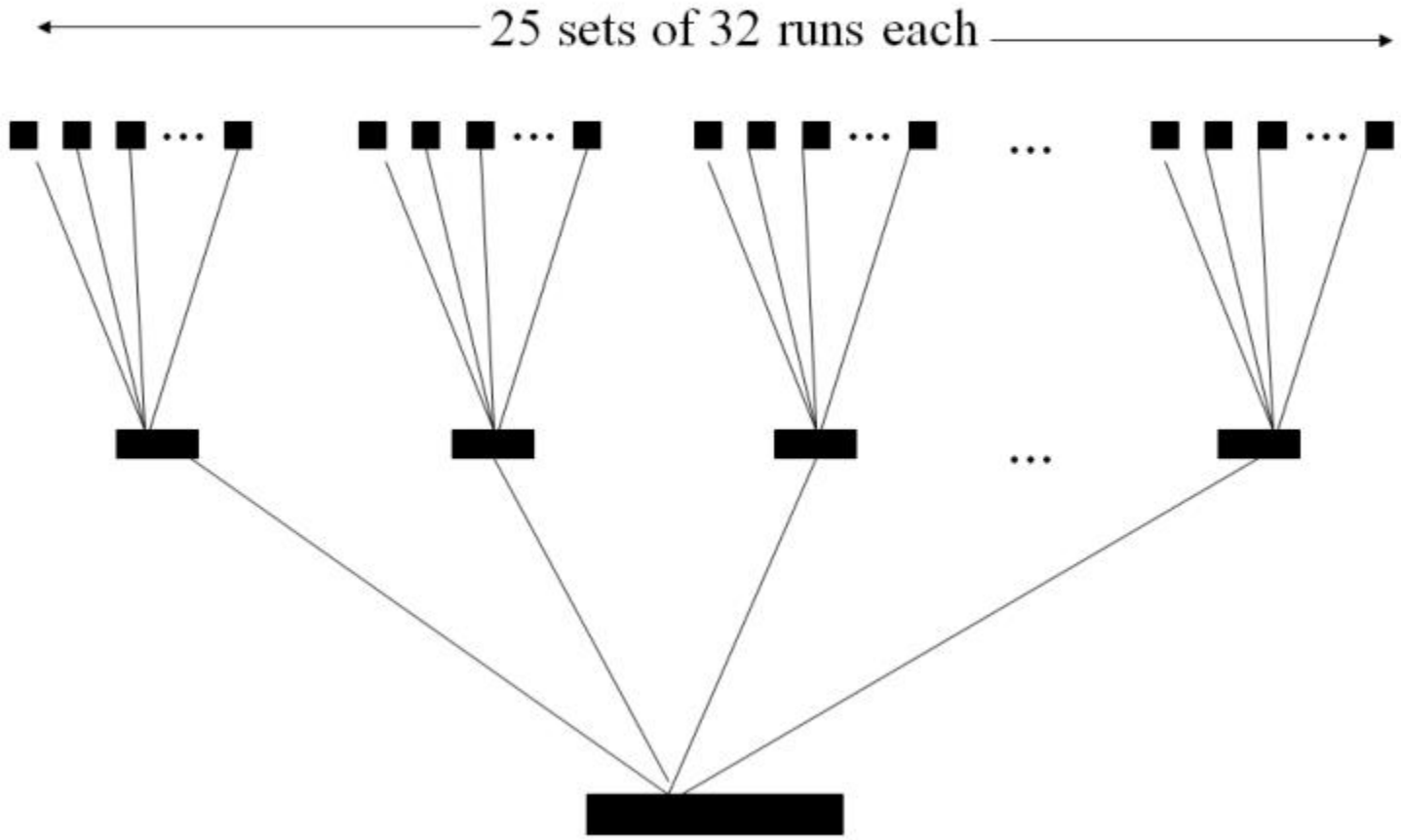
Ordenação Externa

- **Desempenho**
- Apenas a fase de *multiway merging* do exemplo anterior necessita, no mínimo, **1.600 seeks** no disco:
  - Mesmo que cada bloco de registros possa ser lido com um único *seek*, serão necessários **k** = 40 seeks para cada rodada
  - Como são 40 rodadas, tem-se ao menos 40 $\times$ 40 = 1.600 seeks
    - Isso sem contabilizar os *seeks* necessários para a escrita dos blocos ordenados no arquivo de saída
- Isso é razoável?

Ordenação Externa

- **Estratégias para melhorar o desempenho**
  - Aumentar o tamanho das rodadas e reduzir a quantidade de rodadas a serem intercaladas (k)
    - Estratégia de *Replacement Selection* (Seção 8.4.2 de Folk & Zoellick, 1987)
  - Intercalação em múltiplos passos (a seguir)
  - Leitura/Escrita em paralelo
    - Arquivos em múltiplos dispositivos de armazenamento externo

Multi-Step K-Way Merge Sort



Ordenação Externa

- **Intercalação em múltiplas etapas**
  - No exemplo anterior, ao invés de intercalar as 40 rodadas de uma só vez, será que é vantajoso fazer a intercalação em múltiplas etapas ?
  - Por exemplo
    - 1ª etapa: 5 intercalações de 8 rodadas (1.000.000 de registros cada), produzindo 5 rodadas (8.000.000 de registros cada)
      - cada intercalação em 8 vias demanda quantos *seeks*?
      - qual o número total de *seeks*, dado que são 5 intercalações?
    - 2ª etapa: 1 intercalação das 5 rodadas geradas na etapa anterior, produzindo 1 rodada (que é o arquivo original ordenado)
      - RAM total disponível comporta 1/8 de cada rodada
        - logo, cada um dos 5 buffers em RAM comporta 1/40 de cada rodada
      - quantos *seeks* são necessários por rodada?

Ordenação Externa

- **Intercalação em múltiplas etapas**
  - No exemplo anterior, ao invés de intercalar as 40 rodadas de uma só vez, será que é vantajoso fazer a intercalação em múltiplas etapas ?
  - Por exemplo
    - 1ª etapa: 5 intercalações de 8 rodadas (1.000.000 de registros cada), produzindo 5 rodadas (8.000.000 de registros cada)
      - cada intercalação demanda $8 \times 8 = 64$ *seeks*
      - total de $64 \times 5 =$ **320 seeks**
    - 2ª etapa: 1 intercalação das 5 rodadas geradas na etapa anterior, produzindo 1 rodada (que é o arquivo original ordenado)
      - RAM total disponível comporta 1/8 de cada rodada
        - logo, cada um dos 5 buffers em RAM comporta 1/40 de cada rodada
      - 40 *seeks* por rodada $\Rightarrow$ total de **200 seeks**

Ordenação Externa

- **Intercalação em múltiplas etapas**
  - A estratégia de intercalar em 2 etapas resultou em um total de 320+200 = **520 seeks**, contra **1.600** da estratégia anterior (única etapa de 40-vias)
    - 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...

Ordenação Externa

- As análises dos métodos de ordenação tradicionais se preocupam basicamente com o tempo de execução dos algoritmos
  - Complexidade computacional é estimada em função da quantidade de operações (comparações, trocas, etc) feitas com os dados na memória principal (primária)
  - modelo de **ordenação interna**
- Quando é preciso ordenar um arquivo 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 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** (sub-)arquivos ordenados em RAM e fazendo a intercalação (*merging*) desses arquivos
  - Ordenação externa via intercalação em múltiplas vias
- Essa abordagem, porém, possui uma limitação
  - A quantidade k e o tamanho das rodadas são determinados pela memória primária disponível e pelo tamanho do arquivo a ser ordenado
    - Intercalação pode ter que lidar com diversas rodadas de tamanho reduzido (por definição ordenadas)
    - Isso pode inviabilizar a paralelização de L/E em múltiplos dispositivos (usar múltiplos discos/fitas)

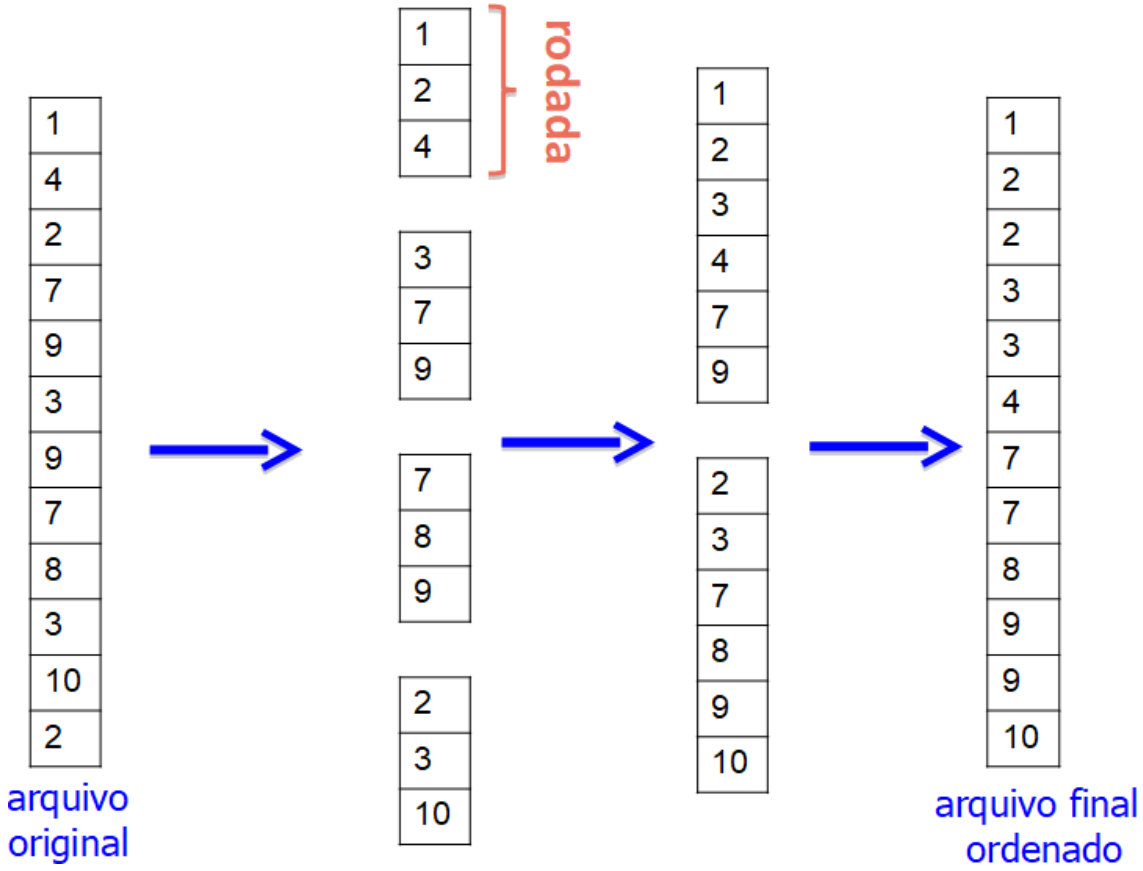
Ordenação Externa

- Seria possível intercalar 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 **MergeSort Externo**

MergeSort Externo

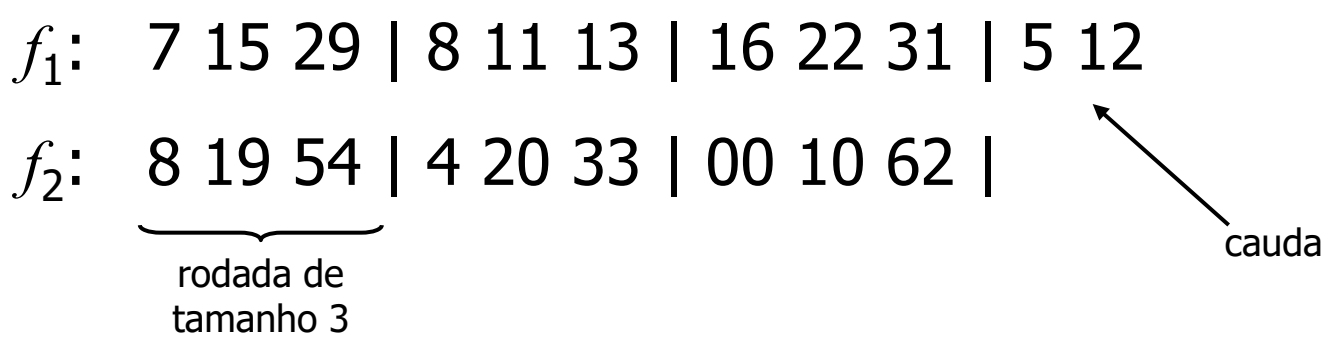
- A versão básica do algoritmo opera com 4 arquivos
  - Para discutir a ideia do algoritmo, vamos inicialmente assumir que todos os 4 arquivos estão armazenados em um único disco
- Os registros são lidos de 2 arqs. de origem e reescritos de forma parcialmente ordenada em 2 arqs. de destino
- Os arquivos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

MergeSort


1
4
2
7
9
3
9
7
8
3
10
2
arquivo original
1
2
4
3
7
9
7
8
9
2
3
10
rodada
1
2
3
4
7
9
2
3
7
8
9
10
1
2
2
3
3
4
7
7
8
9
9
10
arquivo final ordenado

MergeSort Externo

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



MergeSort Externo

- Inicialmente vamos assumir que os arquivos estão 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 blocos de registros são lidos sequencialmente de ambos os arquivos e intercalados
  - algoritmo de *intercalação* **por rodadas**
- Isso significa que cada rodada de um arquivo é intercalada com a rodada correspondente do outro arquivo, formando uma rodada com o dobro do tamanho

MergeSort Externo

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	

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	

MergeSort 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	

MergeSort 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 dobra a cada passagem, tem-se que após i passagens o tamanho da rodada é $k = 2^i$, e 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
∅																						

Desempenho (Interno)

- Portanto, o número de passagens necessárias é tal que $2^i \geq n$
- Logo, qualquer $i \geq \log n$ número de passagens são suficientes:
  - Ou seja, não mais que $\lceil \log n \rceil$ 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 MergeSort 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 ~= $4(n/2)$ ~= $2n$
- 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 $n >> b$

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

  - 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

Comparação de Desempenho

- O número de passagens necessárias é tal que:
  - $1.000.000 \times 2^i > 40.000.000 \Rightarrow i > \log_2 40 \Rightarrow i = 6$

- 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...

Comparação de Desempenho

- Por outro lado, sabemos que a ordenação desse mesmo arquivo por meio de intercalação em 40 vias de 40 arquivos com 1.000.000 de registros cada (ordenados em RAM) requer 1.600 **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 1.600 acessos de escrita
- **Total de Acessos: 3.200**

Comparação de Desempenho

- Tem-se então o no. de **acessos** estimado em:
  - **3.200** de 25Mb para **intercalação em múltiplas vias**; e
  - **1.454** de 330Mb para **MergeSort Externo**...
- MergeSort Externo poderia ser melhor ?
  - Múltiplos dispositivos de memória secundária

Exercícios

- Modifique o código em C da operação **merging** de forma tal que o usuário não precise definir o valor limitante superior denominado VALOR_ALTO, ou seja, de forma que essa variável seja eliminada.

- Modifique o código em C da operação **merging** de forma a produzir uma operação que resulte na soma das listas originais, isto é, que resulte em uma lista de saída que inclua as entradas repetidas em ambas as listas de entrada.

- Modifique os códigos em C das operações **merging** e **matching** para o caso mais geral de processamento simultâneo de múltiplas listas (**multi-way**).

Exercícios

- O exemplo visto de ordenação de um arq. de 40Gb via intercalação em 2 etapas utilizou uma configuração denominada de 8:8:8:8:8, que significa 5 intercalações em 8-vias (1º etapa) seguidas de uma intercalação em 5-vias (2º etapa)
  - Calcule o limitante inferior para o número de *seeks* em uma estratégia 5:5:5:5:5:5:5:5, ou seja, 8 intercalações em 5 vias seguidas de uma intercalação em 8 vias.
- Calcule a quantidade estimada de registros transferidos em cada *seek* das duas diferentes estratégias do exercício anterior
  - Note que sua resposta deve ser calculada de forma independente para cada uma das duas etapas do procedimento de intercalação

Exercícios

- Escolha uma seqüência de 31 números não ordenados e ilustre passo-a-passo, em detalhes, a ordenação dessa seqüência de números através de MergeSort 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 MergeSort Externo? Qual é esse no. de passagens? Quantos acessos de L/E são realizados pelo algoritmo?

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.**

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