



Processamento Cosequencial: Ordenação Interna e Externa

Thiago A. S. Pardo

Leandro C. Cintra

M.C.F. de Oliveira

Cristina D. A. Ciferri



Ordenação em RAM

- Situação: arquivo cabe em RAM
- Etapas
 - leitura de todos os registros do arquivo
 - ordenação dos registros em memória
 - escrita dos registros ordenados no arquivo
- Custo
 - soma dos custos das 3 etapas
 - leitura e escrita sequenciais minimizam *seeks*
 - uso de métodos eficientes de ordenação interna



Ordenação em RAM

- Situação: paralelizando a ordenação com o processamento de entrada e saída (ou seja, leitura e escrita de registros)

parte 1 = leitura; parte 2 = ordenação

parte 1 = ordenação; parte 2 = escrita

possíveis
paralelismos



Heapsort

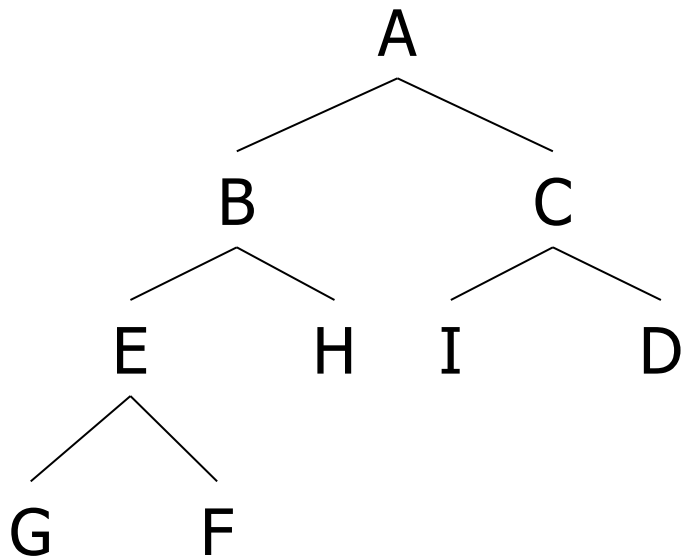
- Parte 1 = leitura; parte 2 = ordenação
 - começa ordenando um conjunto inicial de registros, construindo o *heap*
 - conforme mais dados vão chegando, eles vão sendo incorporados ao *heap*

não é necessário que o arquivo inteiro esteja carregado na memória principal



Heap

- Estrutura que mantém as chaves
- Árvore binária, implementada como vetor



1	2	3	4	5	6	7	8	9
A	B	C	E	H	I	D	G	F

Filhos de i : $2i$ e $2i+1$

Pai de j : $\lfloor j/2 \rfloor$



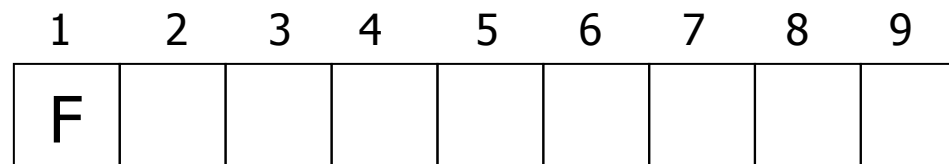
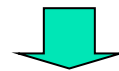
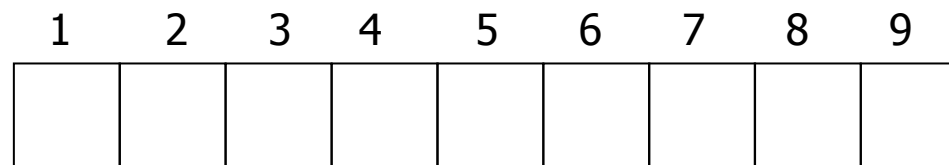
Construção do *heap*

- Algoritmo
 - insere o elemento no fim do vetor
 - enquanto o elemento for menor do que o seu pai, troca-o de lugar com o pai
- Exemplo
 - heap com 9 posições
 - chaves: F, D, C, G, H, I, B, E, A



Construção do *heap*

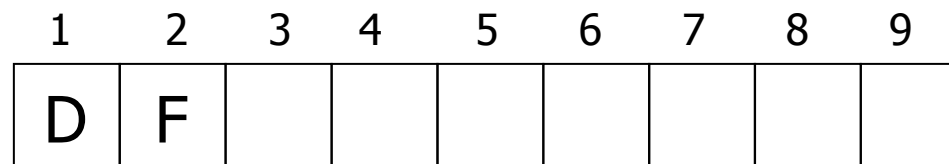
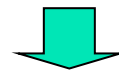
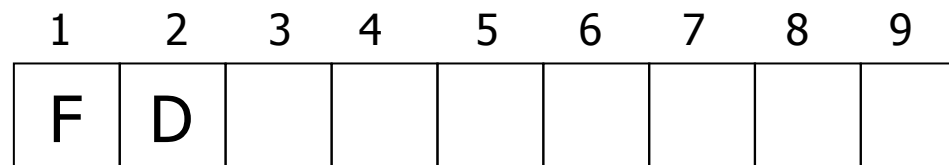
- Elemento: F





Construção do *heap*

- Elemento: D

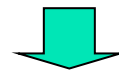




Construção do *heap*

- Elemento: C

1	2	3	4	5	6	7	8	9
D	F	C						



1	2	3	4	5	6	7	8	9
C	F	D						



Construção do *heap*

- Elemento: G

1	2	3	4	5	6	7	8	9
C	F	D	G					



Construção do *heap*

- Elemento: H

1	2	3	4	5	6	7	8	9
C	F	D	G	H				



Construção do *heap*

- Elemento: I

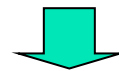
1	2	3	4	5	6	7	8	9
C	F	D	G	H	I			



Construção do *heap*

- Elemento: B

1	2	3	4	5	6	7	8	9
C	F	D	G	H	I	B		



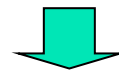
1	2	3	4	5	6	7	8	9
B	F	C	G	H	I	D		



Construção do *heap*

- Elemento: E

1	2	3	4	5	6	7	8	9
B	F	C	G	H	I	D	E	



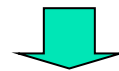
1	2	3	4	5	6	7	8	9
B	E	C	F	H	I	D	G	



Construção do *heap*

- Elemento: A

1	2	3	4	5	6	7	8	9
B	E	C	F	H	I	D	G	A



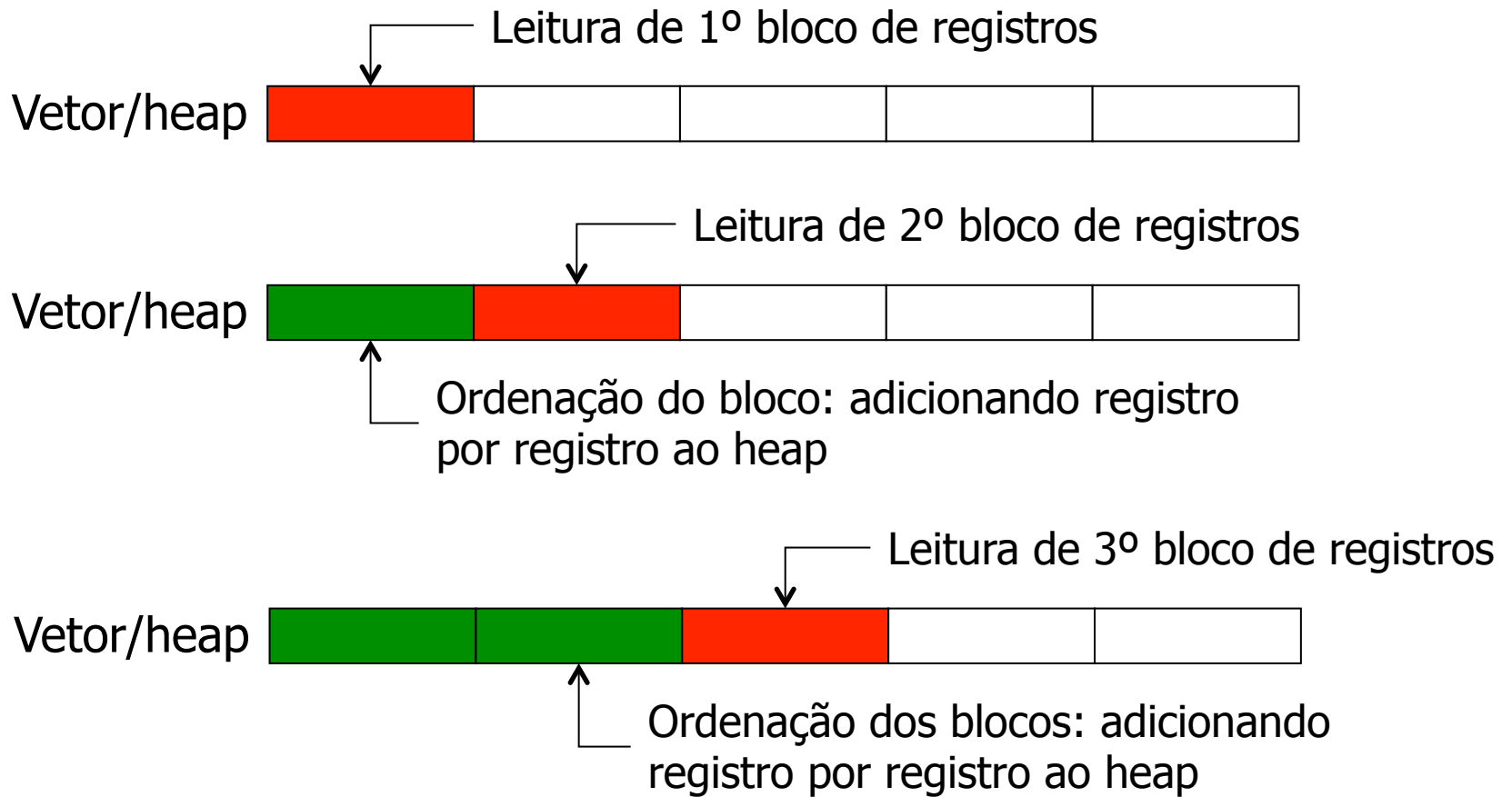
1	2	3	4	5	6	7	8	9
A	B	C	E	H	I	D	G	F



Paralelismo leitura/ordenação

- Percebe-se que
 - O **heap se rearranja** conforme novos elementos são inseridos
 - Portanto, **não é preciso ter todos os registros** para se iniciar a ordenação
 - Enquanto ordenação é feita, novos blocos de registros podem ser lidos e adicionados ao final do vetor, para serem absorvidos na sequência

Paralelismo leitura/ordenação



...



Heapsort

- Parte 1 = ordenação; parte 2 = escrita
 - recupera o registro da raiz do *heap*
 - enquanto rearranja o *heap*, grava esse registro no arquivo de saída
- Rearranjo do *heap*
 - retira o elemento da raiz
 - coloca o último elemento k do *heap* como raiz
 - enquanto k for maior do que seus filhos, troca-o de lugar com seu menor filho



Exemplo

1	2	3	4	5	6	7	8	9
A	B	C	E	H	I	D	G	F

Recupera-se raiz A, colocando em seu lugar F

1	2	3	4	5	6	7	8	9
F	B	C	E	H	I	D	G	---

Enquanto grava A no arquivo ordenado, rearranja heap

1	2	3	4	5	6	7	8	9
B	E	C	F	H	I	D	G	---



Exemplo

1	2	3	4	5	6	7	8	9
B	E	C	F	H	I	D	G	---

Recupera-se raiz B, colocando em seu lugar G

1	2	3	4	5	6	7	8	9
G	E	C	F	H	I	D	---	---

Enquanto grava B no arquivo ordenado, rearranja *heap*

1	2	3	4	5	6	7	8	9
C	E	D	F	H	I	G	---	---

E assim por diante, até *heap* esvaziar



Ordenação Externa

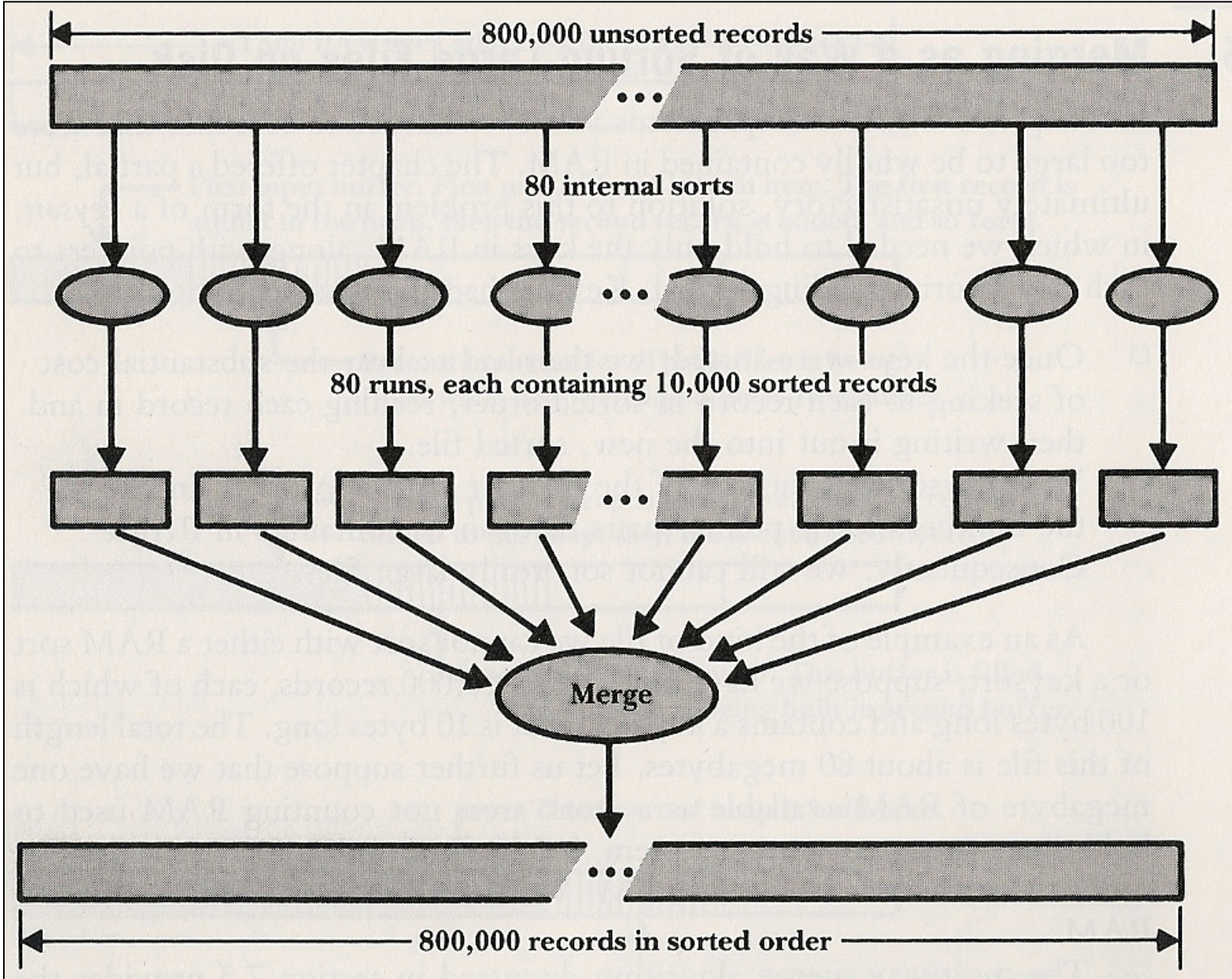
- Funcionalidade
 - classifica registros de arquivos armazenados em disco
- Restrição
 - tamanho do arquivo é maior do que o tamanho da memória principal disponível
- Algoritmo típico
 - *sort-merge externo*

pode ordenar arquivos
realmente grandes



Sort-Merge Externo

- Fase 1
 - cria subarquivos ordenados (i.e., *runs*) a partir do arquivo original
- Fase 2
 - combina os subarquivos ordenados em subarquivos ordenados maiores até que o arquivo completo esteja ordenado
 - usa-se *multiway merging*





Algoritmo

// inicializações

$i \leftarrow 1$ // tamanho do arquivo original em blocos

$j \leftarrow b$ // tamanho do buffer em blocos

$k \leftarrow n_b$

$m \leftarrow \lceil (j/k) \rceil$

número de subarquivos iniciais gerados, de acordo com o tamanho do arquivo original e o tamanho do buffer de memória

merge-sort externo requer espaço disponível nos buffers da memória principal



Algoritmo

// fase 1 – fase de ordenação

enquanto ($i \leq m$) faça {

 leia os próximos k blocos do arquivo no buffer *ou*

 os blocos remanescentes caso não hajam k blocos

 ordene os registros que estão no buffer usando algum

 algoritmo de ordenação interna

 escreva os dados ordenados em um subarquivo temporário

$i \leftarrow i + 1$

}

- resultado da fase 1
 - m subarquivos ordenados armazenados em disco



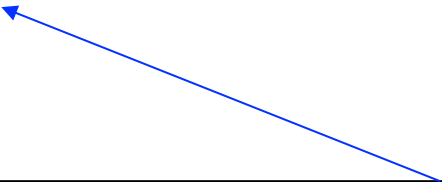
Algoritmo

// fase 2 – fase de combinação dos subarquivos ordenados

$i \leftarrow 1$

$p \leftarrow \lceil \log_{k-1} m \rceil$ // número de passos

$j \leftarrow m$

- 
- *merge-sort* externo
 - combina os subarquivos ordenados em subarquivos ordenados maiores em um ou mais passos
 - grau de combinação
 - número de subarquivos ordenados que podem ser combinados em cada passo



Algoritmo

enquanto ($i \leq p$) faça {

$n \leftarrow 1$

$q \leftarrow \lceil (j/(k-1)) \rceil$ // número de subarquivos a serem escritos no passo

 enquanto ($n \leq q$) faça {

 leia os próximos $k-1$ subarquivos ordenados remanescentes
 (do passo anterior) um bloco por vez

 combine os subarquivos ordenados

 escreva os dados ordenados em um subarquivo temporário

 }

$j \leftarrow q;$

$i \leftarrow i + 1$

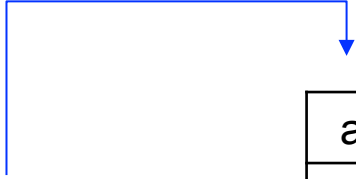
}

- resultado da fase 2
 - arquivo original ordenado armazenado em disco

g	24
a	19
d	31
c	33
b	14
e	16
r	16
d	21
m	3
p	2
d	7
a	14

[arquivo original](#)

fase 1



g	24
a	19
d	31
c	33
b	14
e	16
r	16
d	21
m	3
p	2
d	7
a	14

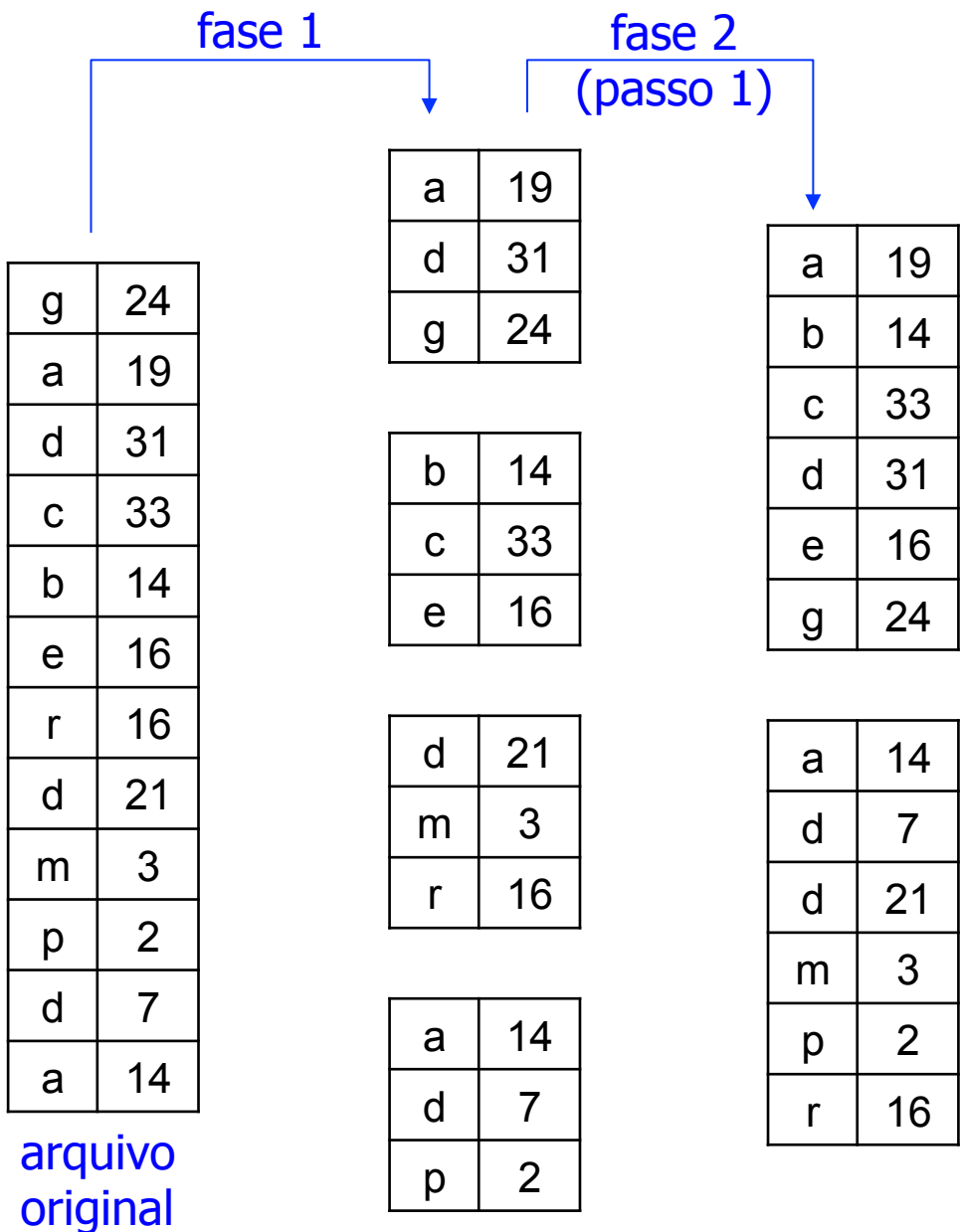
arquivo
original

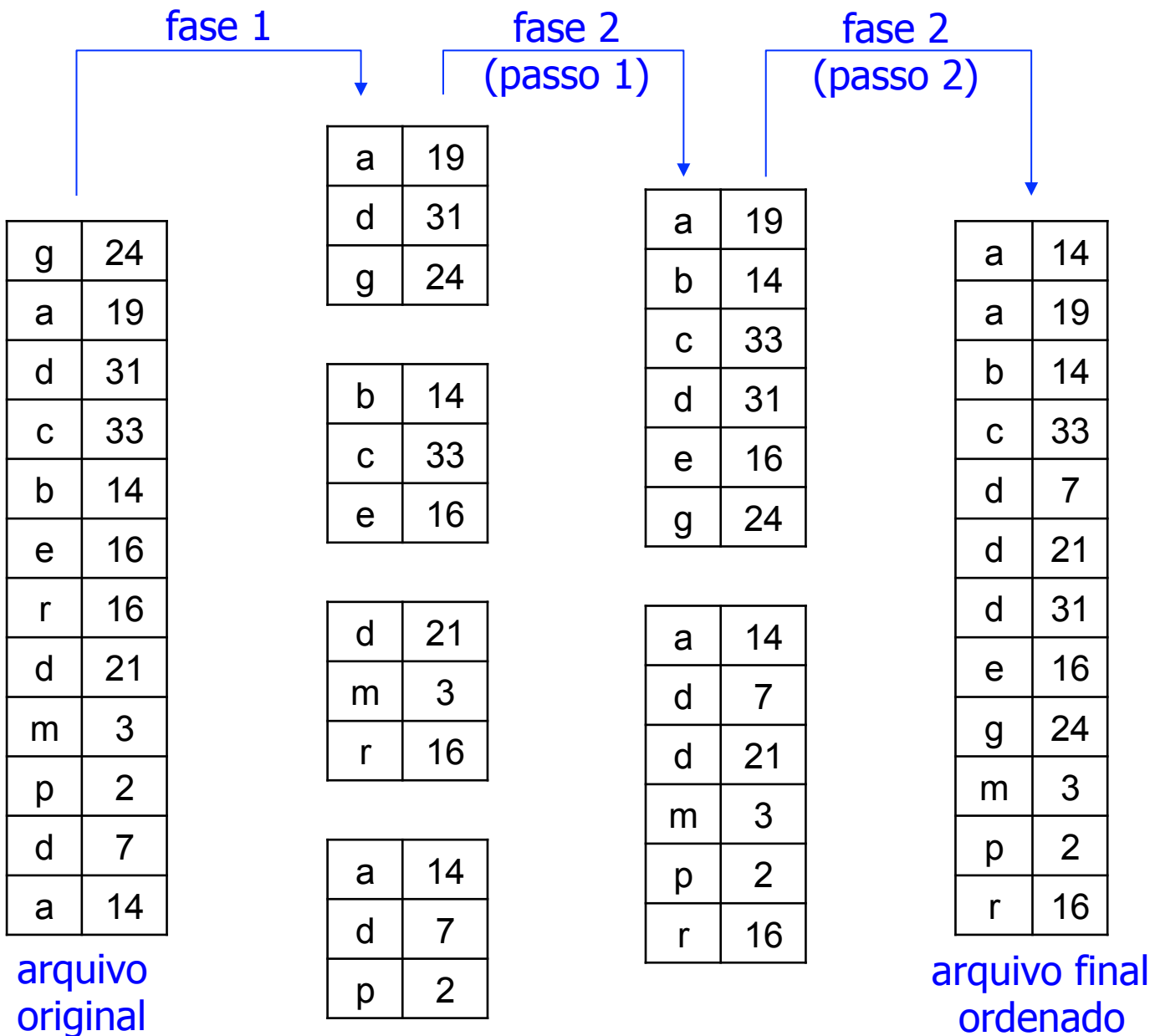
a	19
d	31
g	24

b	14
c	33
e	16

d	21
m	3
r	16

a	14
d	7
p	2







Custo do Algoritmo

$$(2 * b) + (2 * (b * (\log_{d_m} b)))$$

- $(2 * b)$
 - número de acessos a disco na **fase 1**
- cada bloco do arquivo é acessado duas vezes
 - fase de leitura dos dados para a memória
 - fase de escrita dos dados ordenados no disco



Custo do Algoritmo

$$(2 * b) + (2 * (b * (\log_{d_m} b)))$$

- $(2 * (b * (\log_{d_m} b)))$
 - número de acessos a disco na **fase 2**
- cada bloco dos subarquivos é acessado várias vezes, dependendo do grau de combinação
 - fase de leitura dos dados para a memória
 - fase de escrita dos dados ordenados no disco