

Ordenação e Busca em Arquivos

Cristina D. A. Ciferri

Thiago A. S. Pardo

Leandro C. Cintra

M.C.F. de Oliveira

Moacir Ponti Jr.



Ordenação

- Facilita a busca
- Pode ajudar a diminuir o número de acessos a disco



Busca Sequencial e Binária

- Busca sequencial
 - recupera cada registro do arquivo, verificando se os valores dos atributos satisfazem à condição de seleção
- Busca binária
 - recupera registros quando a condição de seleção envolve uma comparação de igualdade no atributo que determina a ordenação do arquivo



Custos: Comparações

$$C_{\text{busca_sequencial}} = n$$

- n: número de registros que são comparados
- todos os registros são varridos (pior caso)
- complexidade: $O(n)$

$$C_{\text{busca_binária}} = \log_2(n) + 1$$

- n: número de registros que são comparados
- complexidade: $O(\log n)$



Custos: Acessos a Disco

$$C_{\text{busca_sequencial}} = b$$

- b : número de blocos que contêm os registros
- todos os blocos são varridos

$$C_{\text{busca_binária}} = \log_2(b) + \lceil s/bfr \rceil - 1$$

- $\log_2(b)$: custo para localizar o primeiro registro
- $\lceil s/bfr \rceil$: blocos ocupados pelos registros que satisfazem à condição de seleção
- 1: custo para recuperar o primeiro registro



Ordenação Interna



Arquivo Completo Cabe em RAM

- Etapas

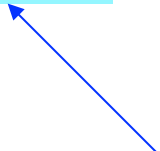
1. leitura de todos os registros armazenados em disco para a RAM
2. ordenação dos registros em RAM
 - escolha do campo base para ordenação
 - uso de um método de ordenação
3. escrita de todos os registros armazenados em RAM para o disco



Arquivo Ordenado

- Registros de tamanho fixo

A N A		R U A	b	A U G U S T O	b	P A I V A		I B A T E		b b		
A N T O N I A		R U A	b	X V	b	D E	b	M A I O		I B A T E		b
J O A O		R U A	b	A		R I O	b	C L A R O		b b b b b b b b		
M A R I A		R U A	b	1		S A O	b	C A R L O S		b b b b b b b		
P E D R O		R U A	b	X V		S A O	b	C A R L O S		b b b b b b b		



ordenação baseada em um determinado campo,
usando suas chaves



Chave (KEY)

- Está associada a um registro e permite a sua recuperação
- Chave **primária**
 - identifica univocamente um registro
 - não tem repetição
- Chave **secundária**
 - não identifica univocamente um registro
 - tem repetição



Forma Canônica da Chave

- Uma única representação para uma determinada chave
- Exemplo
 - "Ana", "ANA", ou "ana" devem indicar o mesmo registro
 - Forma canônica: todos os caracteres em letras maiúsculas → ANA



Custo

- Soma dos custos das 3 etapas
 - custo para ler o arquivo do disco para a RAM
 - custo para escrever o arquivo da RAM para o disco
 - custo do método de ordenação escolhido
 - Quicksort: $O(n \lg(n))$
 - Heapsort: $O(n \lg(n))$
 - *n*: número de registros que são comparados
- Importante
 - leitura e escrita sequenciais minimizam acessos a disco



Como Melhorar o Desempenho (custo)?

- 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



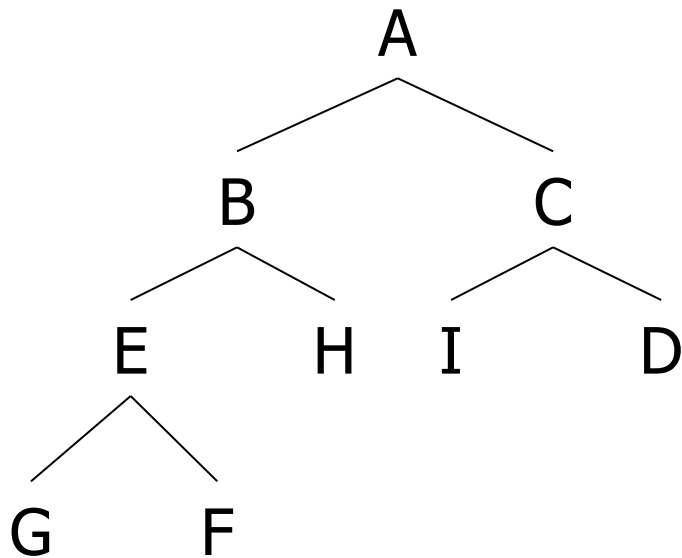
Heapsort

- 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 em RAM
 - lê uma página de disco (bloco de disco) por vez e opera sobre seus registros antes de ler a próxima página (bloco de disco)



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 : $\lfloor j/2 \rfloor$



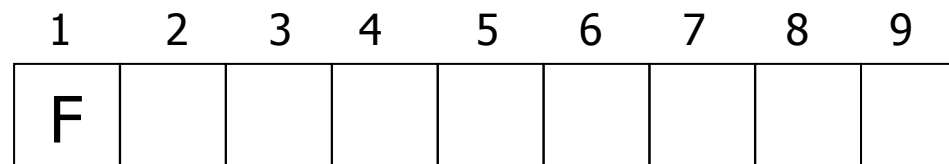
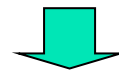
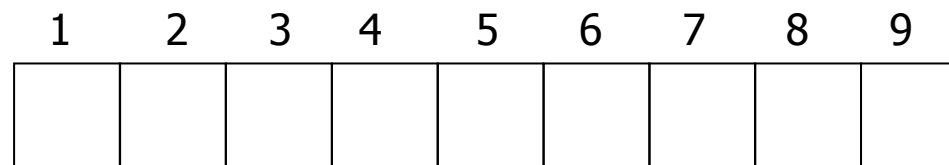
Construção do heap

- 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

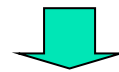




Construção do heap

- Elemento: D

1	2	3	4	5	6	7	8	9
F	D							



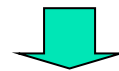
1	2	3	4	5	6	7	8	9
D	F							



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

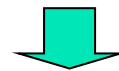
1	2	3	4	5	6	7	8	9
C	F	D	G	H	I			



Construção do heap

- Elemento: B

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



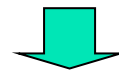
1	2	3	4	5	6	7	8	9
B	F	C	G	H	I	D		



Construção do heap

- Elemento: E

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



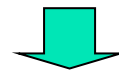
1	2	3	4	5	6	7	8	9
B	E	C	F	H	I	D	G	



Construção do heap

- Elemento: A

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



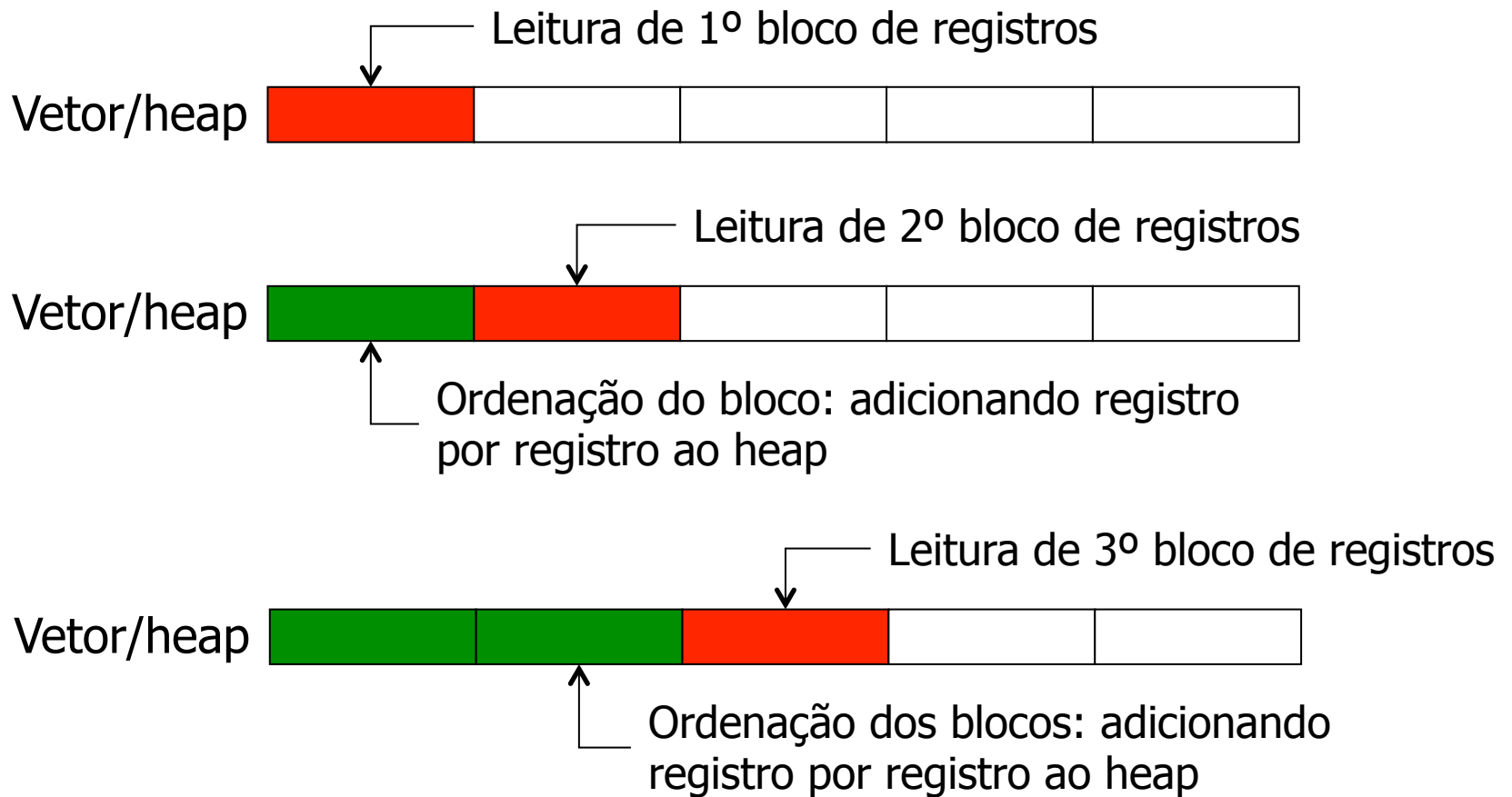
1	2	3	4	5	6	7	8	9
A	B	C	E	H	I	D	G	F



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	---

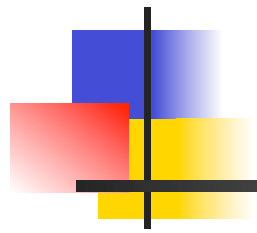
Recupera-se raiz B, colocando em seu lugar G

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



Arquivo Completo não Cabe em RAM

- 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



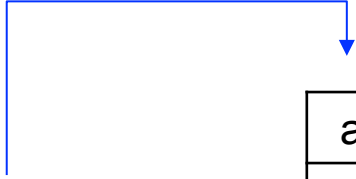
Sort-Merge Externo

- 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

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

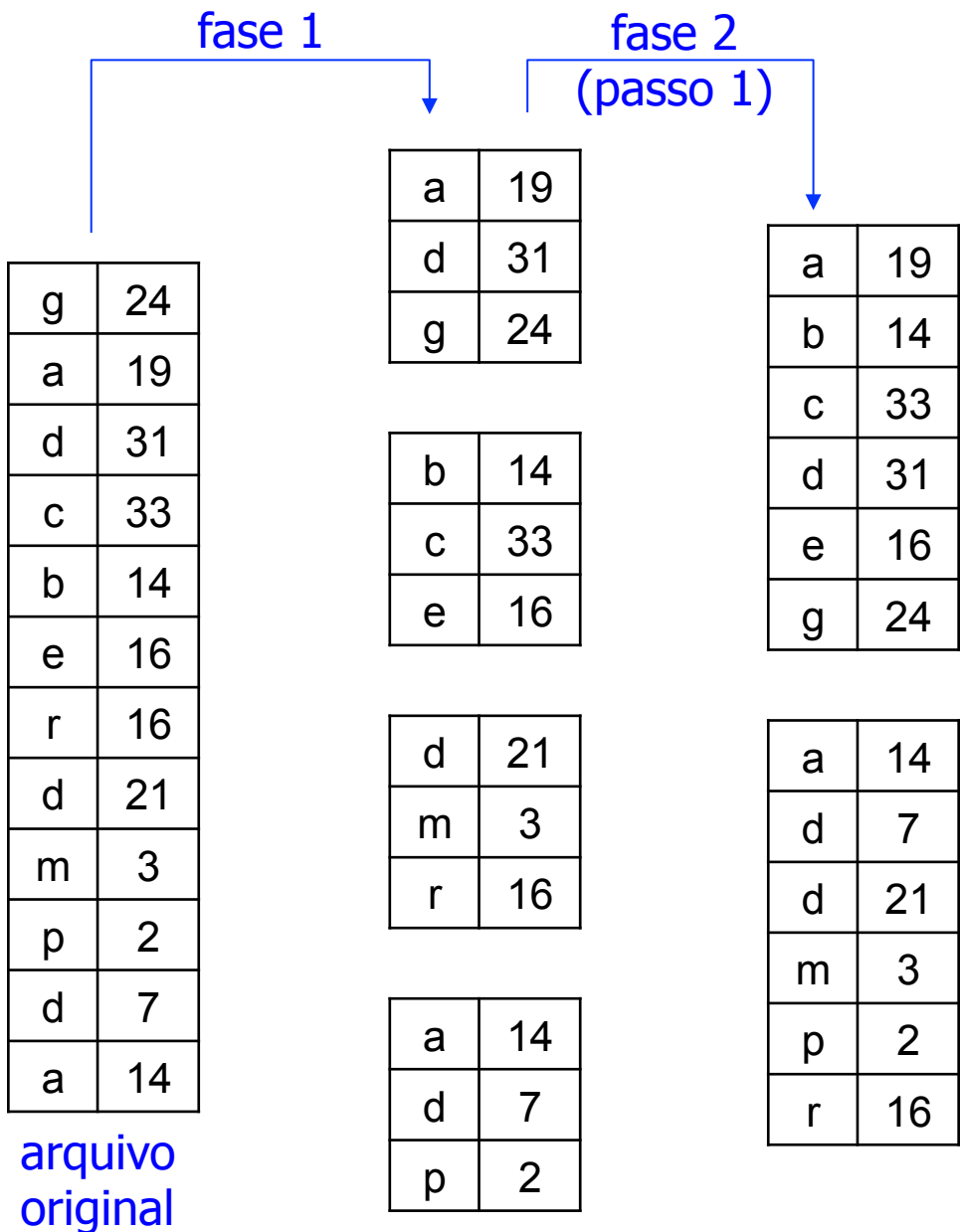
arquivo
original

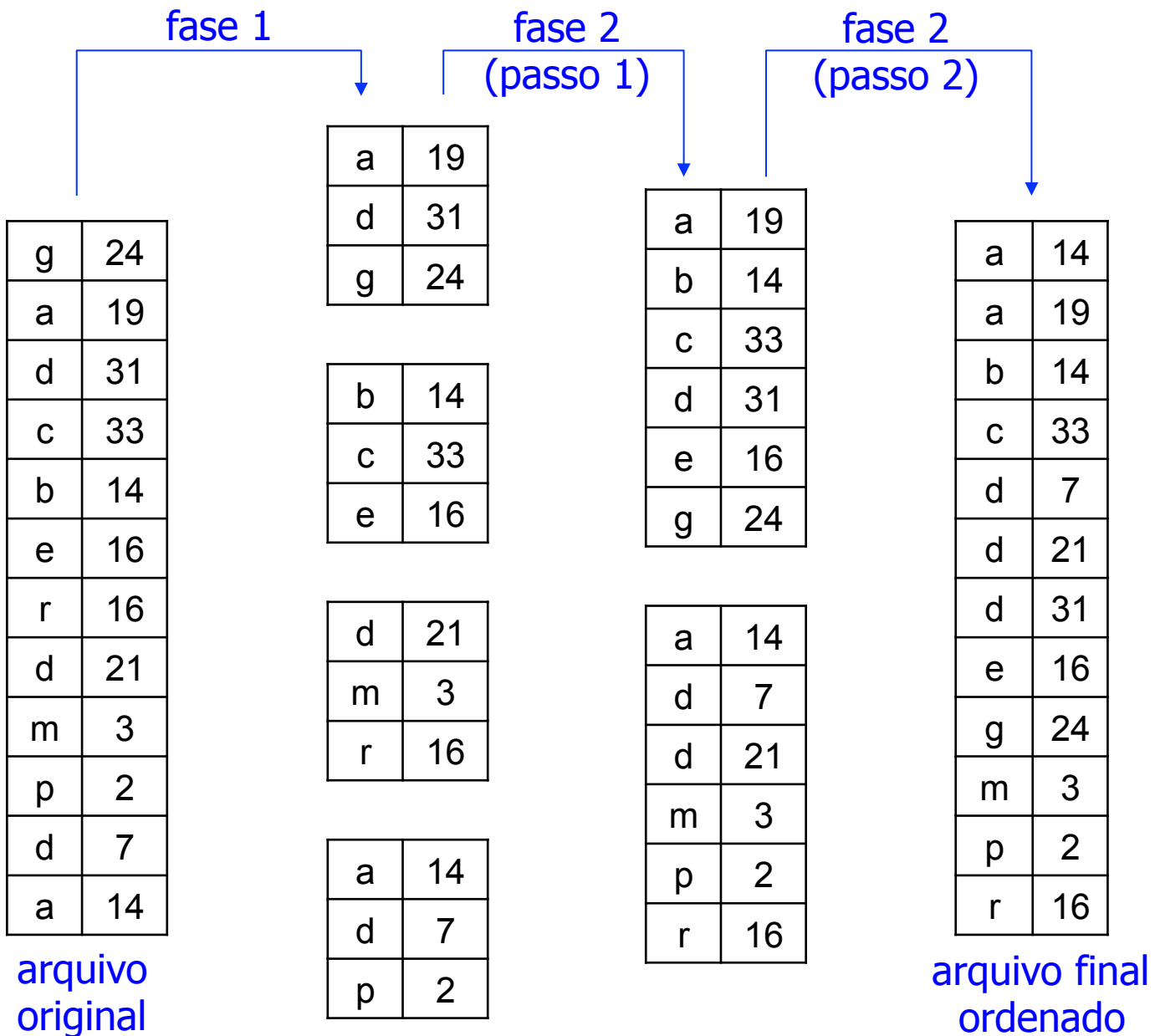
a	19
d	31
g	24

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

