Organização de Arquivos (part 2)

Prof. Dr. Lucas C. Ribas

Disciplina: Estrutura de Dados II

Departamento de Ciências de Computação e Estatística

unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

Agenda da aula

- Modelo abstrato de dados
- Cabeçalhos, metadados e taggs
- Leitura e escrita
- Desempenho em organização de arquivos
  - Compressão
  - Compactação
  - Fragmentação
  - Busca

unesp

Organização x Acesso

- Organização Física:
  - registros de tamanho variável
  - registros de tamanho fixo
- Acesso ao Arquivo:
  - Sequencial
  - Direto
- O que influencia:
  - o uso que se fará do arquivo
  - facilidades da linguagem de programação usada

unesp

Para o usuário

- Foco no conteúdo do arquivo, e não no seu formato;
- Acesso à informação, e não a registros e campos;
- Distância maior entre a organização física e lógica do arquivo

Modelos Abstratos de Dados

- Focar no conteúdo da informação, ao invés de no seu formato físico
- As informações atuais tratadas pelos computadores (som, imagens, documentos, etc.) não se ajustam bem à metáfora de dados armazenados como sequências de registros separados em campos
- É mais fácil pensar em dados deste tipo como objetos que representam som, imagens, etc. e que têm a sua própria maneira de serem manipulados
- O termo **modelo abstrato de dados** captura a noção de que o dado não precisa ser visto da forma como está armazenado - ou seja, permite uma visão dos dados orientada à aplicação, e não ao meio no qual eles estão armazenados

Registro Cabeçalho (*header record*)

- Em geral, é interessante manter algumas informações sobre o arquivo para uso futuro
- Essas informações podem ser mantidas em um cabeçalho no início do arquivo
- A existência de um registro cabeçalho torna um arquivo um objeto auto-descrito
  - O software pode acessar arquivos de forma mais flexível

unesp

Registro Cabeçalho (*header record*)

- Algumas informações típicas:
  - Número de registros
  - Tamanho de cada registro
  - Nomes dos campos de cada registro
  - Tamanho dos campos
  - Datas de criação e atualização

```
1   NR=2
2   TR=64
3   C1=Nome
4   C2=RA
5   C3=Curso
6   TC=64
7   DATE='04/10/2022'
8   Lucas|12345|Ciencia da Computacao##############################
```

unesp

Registro Cabeçalho (*header record*)

- Vantagem dessas abordagem
  - Podemos criar um programa que lê/escreve um grande numero de arquivos com diferentes características (número de campos por registro, comprimento de campos)
  - Quanto mais informações houver no *header*, menos o o programa precisa saber sobre a estrutura específica de um arquivo em particular
- Desvantagem
  - Programa que lê/escreve mais sofisticado para interpretar diferentes headers

unesp

Metadados

- São **dados que descrevem os dados primários** em um arquivo
- Exemplo: Formato FITS (*Flexible Image Transport System*)
- Armazena imagens de astronomia
- Cada imagem é precedida por um cabeçalho FITS: uma coleção de blocos de **2880 bytes** contendo registros de **80 bytes** ASCII, com dados sobre a imagem: posição do céu, data de captura, telescópio usado, etc. São chamados metadados
- O FITS utiliza o formato ASCII para o cabeçalho e o formato binário para os dados primários

```
SIMPLE = T / Conforms to basic format
BITPIX = 16 / Bits per pixel
NAXIS = 2 / Number of axes
   ...
DATE = '22/09/1989 ' / Date of file written
TIME = '05:26:53' / Time of file written
END
```

Metadados

- Vantagens de incluir metadados junto com os dados
  - Torna viável o acesso ao arquivo por terceiros (conteúdo auto-explicativo)
  - Portabilidade
    - Define-se um padrão para todos os que geram/acessam certos tipos de arquivo
    - PDF, PS, HTML, TIFF
    - Permite conversão entre padrões
- Bom uso para etiquetas e palavras-chave
  - keyword=value
  - Espaço ocupado relativo é muito pequeno em FITS: 0.02%
- Se bem descrito, arquivo pode conter muitos dados de formatos e origens diferentes
  - Acesso orientado a objetos
  - “Extensibilidade”

unesp

Desempenho

unesp

Organização de arquivos para desempenho

- Organização de arquivos visando desempenho
  - Complexidade de espaço
    - Compressão (tornar menor) e compactação (eliminar espaços vazios) de dados
    - Reuso de espaço
  - Complexidade de tempo
    - Ordenação e busca de dados

unesp

Compressão de dados

- A compressão de dados envolve a codificação da informação de modo que o arquivo ocupe menos espaço
  - Transmissão mais rápida
  - Processamento sequencial mais rápido
  - Menos espaço para armazenamento
- Algumas técnicas são gerais, e outras específicas para certos tipos de dados, como voz, imagem ou texto
  - Técnicas reversíveis vs. irreversíveis
  - A variedade de técnicas é enorme

unesp

Técnicas

- Noção diferenciada
  - Redução de Redundância
- Omissão de sequências repetidas
  - Redução de Redundância
- Código de tamanho variável
  - Código de Huffman

Notação diferenciada

- Exemplo
- Códigos de estado (SP, MG, RJ, ...), armazenados na forma de texto: 2 bytes
  - Por exemplo, como existem 27 estados no Brasil, pode-se armazenar os estados em 5 bits
  - É possível guardar a informação em 1 byte e economizar 50% do espaço
- Desvantagens?
  - Legibilidade, codificação/decodificação

unesp

Omissão de sequências repetidas

- Para a sequência
  - 22 23 24 24 24 24 24 24 24 25 26 26 26 26 26 26 25 24
- Usando um código indicador de repetição (código de run-length)
  - 22 23 ff 24 07 25 ff 26 06 25 24
- Bom para dados esparsos ou com muita repetição
  - Imagens do céu, por exemplo
- Garante **redução de espaço sempre**?
  - Toda sequencia de repetições é substituída por 3 valores: código, valor repetido, número de repetições

unesp

Códigos de tamanho fixo

- Código ASCII: 1 byte por caracter (fixo)
  - ‘A’ = 65 (8 bits = 01000001) (int) char[i]
  - Cadeia ‘ABC’ ocupa 3 bytes (01000001 01000010 01000011)
  - Ignora frequência dos caracteres

unesp

Códigos de tamanho fixo

65	01000001	U+0041	A
66	01000010	U+0042	B
67	01000011	U+0043	C
68	01000100	U+0044	D
69	01000101	U+0045	E
70	01000110	U+0046	F
71	01000111	U+0047	G
72	01001000	U+0048	H
73	01001001	U+0049	I
74	01001010	U+004A	J
75	01001011	U+004B	K
76	01001100	U+004C	L
77	01001101	U+004D	M
78	01001110	U+004E	N
79	01001111	U+004F	O
80	01010000	U+0050	P
81	01010001	U+0051	Q
82	01010010	U+0052	R
83	01010011	U+0053	S
84	01010100	U+0054	T
85	01010101	U+0055	U
86	01010110	U+0056	V
87	01010111	U+0057	W
88	01011000	U+0058	X
89	01011001	U+0059	Y
90	01011010	U+005A	Z

Códigos de tamanho variável

- Princípio: alguns valores ocorrem mais frequentemente que outros, assim, seus códigos deveriam ocupar o menor espaço possível
- Código Morse
  - Letras mais frequentes – códigos menores
  - Distribuição de frequência conhecida
  - Tabela fixa de códigos de tamanho variável
- **Código de Huffman**
  - Código de tamanho variável
  - Distribuição de frequência desconhecida
  - Tabela de códigos construída dinamicamente
- Se letras que ocorrem com maior frequência têm códigos menores, as cadeias tendem a ficar mais curtas, e o arquivo menor

unesp

Técnicas de compressão irreversíveis

- Até agora, todas as técnicas eram reversíveis
- Algumas são irreversíveis
  - Por exemplo, salvar uma imagem de 400 por 400 pixels como 100 por 100 pixels
- Compressão de Fala – codificação de voz

unesp

Técnicas de compactação

- Compactação:
  - Diminuição do tamanho do arquivo eliminando espaços não utilizados, gerados após operações de eliminação de registros

unesp

Manipulação de dados em arquivos

- Que operações básicas podemos fazer com os dados nos arquivos?
  - Adição de registros: relativamente simples
  - Eliminação de registros
  - Atualização de registros: eliminação e adição de um registro
  - O que pode acontecer com o arquivo?

unesp

Compactação

- Compactação
  - Busca por regiões do arquivo que não contêm dados
  - Posterior recuperação desses espaços perdidos
  - Os espaços vazios são provocados, por exemplo, pela eliminação de registros

unesp

Eliminação de registros

- Devem existir mecanismos que
  - Permitam reconhecer áreas que foram apagadas
  - Permitam recuperar e utilizar os espaços vazios
- Possibilidades?
  - Uso de Marcadores Especiais e Recuperação Esporádica
  - Geralmente, áreas apagadas são marcadas com um marcador especial
  - Eventualmente o procedimento de compactação é ativado, e o espaço de todos os registros marcados é recuperado de uma só vez
    - Maneira mais simples de compactar: executar um programa de cópia de arquivos que "pule" os registros apagados (se existe espaço suficiente para outro arquivo)
    - Ou compactação “in loco”: mais complicada e demorada

unesp

Processo de compactação: exemplo

FIGURE 5.3 Storage requirements of sample file using 64-byte fixed-length records. (a) Before deleting the second record. (b) After deleting the second record. (c) After compaction—the second record is gone.

```
Ames¦John¦123 Maple¦Stillwater¦OK¦74075¦.......................
Morrison¦Sebastian¦9035 South Hillcrest¦Forest Village¦OK¦74820¦
Brown¦Martha¦625 Kimbark¦Des Moines¦IA¦50311¦..................
```

(a)

```
Ames¦John¦123 Maple¦Stillwater¦OK¦74075¦.......................
*¦rrison¦Sebastian¦9035 South Hillcrest¦Forest Village¦OK¦74820¦
Brown¦Martha¦625 Kimbark¦Des Moines¦IA¦50311¦..................
```

(b)

```
Ames¦John¦123 Maple¦Stillwater¦OK¦74075¦.......................
Brown¦Martha¦625 Kimbark¦Des Moines¦IA¦50311¦..................
```

(c)

Recuperação dinâmica

- Muitas vezes, o **procedimento de compactação é esporádico**
  - Um registro apagado não fica disponível para uso imediatamente
- Em aplicações interativas que acessam **arquivos altamente voláteis**, pode ser necessário um **processo de recuperação de espaço, tão logo ele seja criado (dinâmico)**
  - Marcar registros apagados
  - Na inserção de um novo registro, identificar espaço deixado por eliminações anteriores, sem buscas exaustivas
- Requisitos para reaproveitar espaço em novas inserções:
  - Reconhecer espaço disponível no arquivo
  - Pular diretamente para esse espaço, se existir

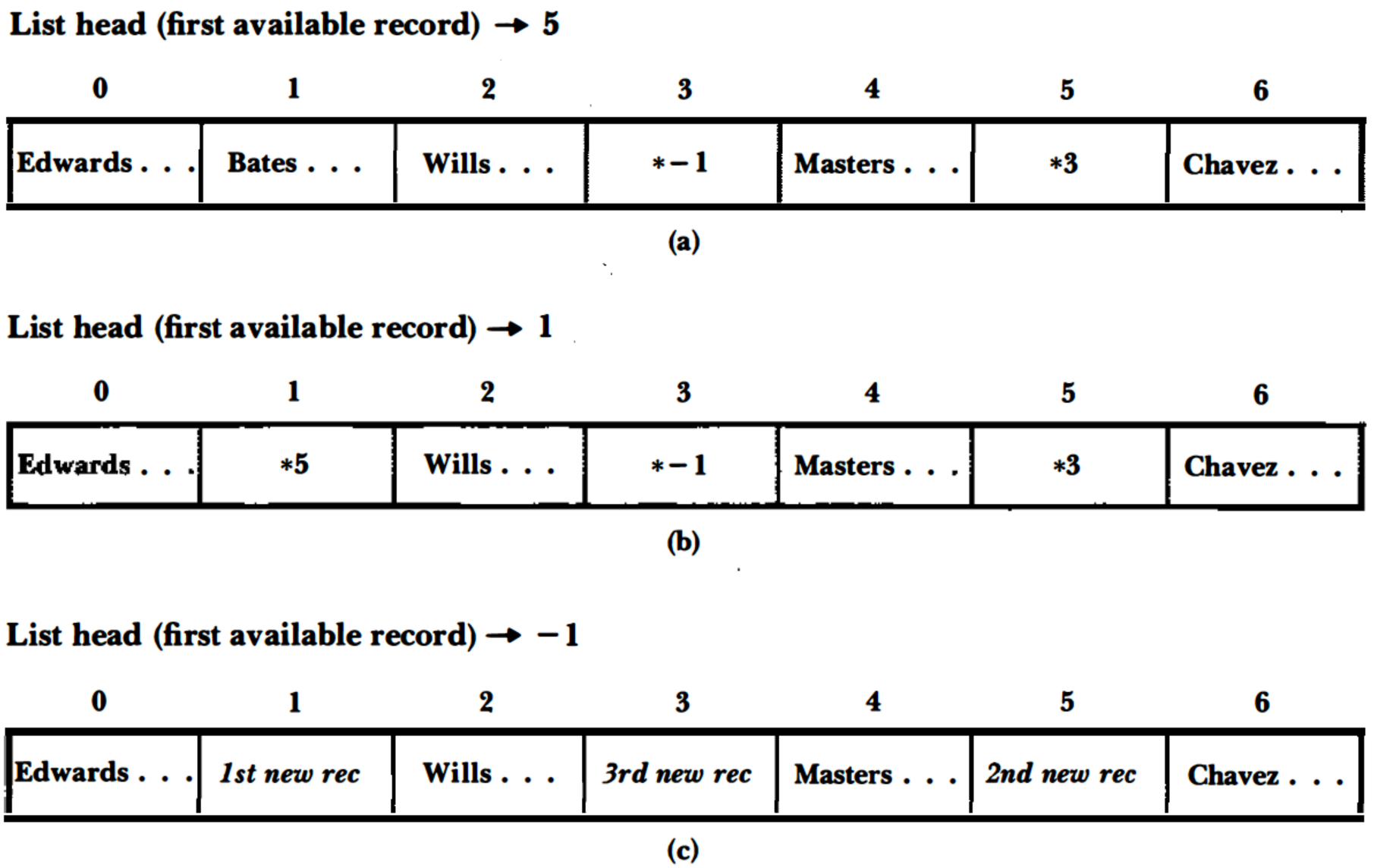
unesp

Como localizar os espaços vazios?

- Registros de tamanho fixo (possuem RRN)
  - Encadear os registros eliminados no próprio arquivo, formando uma Lista de Espaços Disponíveis (LD)
    - LD constitui-se de espaços vagos, endereçados por meio de seus RRNs
    - Cabeça da lista está no *header* do arquivo
    - Um registro eliminado contém a marca de eliminado e o RRN do próximo registro eliminado (que serve como ponteiro para o próximo elemento desta lista)
    - Inserção e remoção ocorrem sempre no início da LD (pilha) Por que?

unesp

Registros de tamanho fixo - Exemplo



Registros de tamanho variável

- Supondo arquivos com indicação do número de bytes antes de cada registro
- Marcação dos registros eliminados via um marcador especial
- Lista LD... mas não dá para usar RRNs
  - Tem que se usar a posição de início no arquivo

unesp

Registros de tamanho variável

- Estratégias de alocação de espaço
  - First-fit: pega-se o primeiro que servir, como feito anteriormente
    - Desvantagem?
      - Fragmentação interna

unesp

Registros de tamanho variável

- Estratégias de alocação de espaço
  - First-fit: pega-se o primeiro que servir, como feito anteriormente
  - Solução para a Fragmentação Interna?
    - Colocar o espaço que sobrou na lista de espaços disponíveis

Registros de tamanho variável

- Alternativa: escolher o espaço mais justo possível
  - Best-fit: pega-se o mais justo
    - Desvantagem?
      - O espaço que sobra é tão pequeno que não dá para reutilizar
      - Fragmentação externa
    - É conveniente organizar a lista LD de forma ascendente segundo o tamanho dos registros
    - O espaço mais “justo” é encontrado primeiro

unesp

Registros de tamanho variável

- Alternativa: escolher o maior espaço possível
  - Worst-fit: pega-se o maior
    - **Diminui** a fragmentação externa
    - Lista LD organizada de forma descendente
      - O processamento pode ser mais simples

unesp

Observações

- Estratégias de alocação só fazem sentido com registros de tamanho variável
- Se espaço está sendo desperdiçado como resultado de fragmentação interna, então a escolha é entre first-fit e best-fit
  - A estratégia **worst-fit** piora esse problema
- Se o espaço está sendo desperdiçado devido à fragmentação externa, deve-se considerar a worst-fit

unesp

Ordenar para otimizar tempo

- Lembre-se: acessar memória externa custa muito!
  - Se um acesso acesso a RAM demorasse 20 segundos, o correspondente ao disco demoraria 58 dias!
- Ordenação torna a busca mais eficiente
  - Mas ordenar também envolve um custo
  - Se cada comparação envolver um seek, então a ordenação tb será proibitiva

unesp

Ordenação e busca de arquivos

unesp

Ordenação e busca de arquivos

- É relativamente fácil buscar elementos em conjuntos ordenados
- A ordenação pode ajudar a diminuir o número de acessos a disco
- Já vimos busca sequencial
  - O(n) → Muito ruim para acesso a disco!
  - É beneficiada por buffering
- E a busca binária?
  - Modo de funcionamento?
  - Complexidade de tempo?
    - Dominada pelo tempo (número) de seeking

unesp

Busca binária

- Requisito: arquivo ordenado
- Dificuldade: aplicar um método de ordenação conhecido nos dados em arquivo
- Alternativa: ordenar os dados em RAM
  - Leitura sequencial, por setores (bom!)
  - Ainda é necessário: ler todo o arquivo e ter memória interna disponível

Busca binária

- Limitações
  - Registros de tamanho fixo para encontrar “o meio”
  - Manter um arquivo ordenado é muito caro
    - Custo pode superar os benefícios da busca binária
    - Inserções em “*batch mode*” : ordena e merge
  - Requer mais do que 1 ou 2 acessos
    - Por exemplo, em um arquivo com 1.000 registros, são necessários aproximadamente 10 acessos (~$\log_2 1000$) em média → ainda é ruim!

unesp

Problemas a solucionar

- Reordenação dos registros do arquivo a cada inserção
  - Uso de índices e/ou hashing
  - Outras estruturas de dados (árvores)
- Quando o arquivo não cabe na RAM
  - Keysort – variação de ordenação interna
  - Alternativa para não reordenar o arquivo

unesp

Keysorting

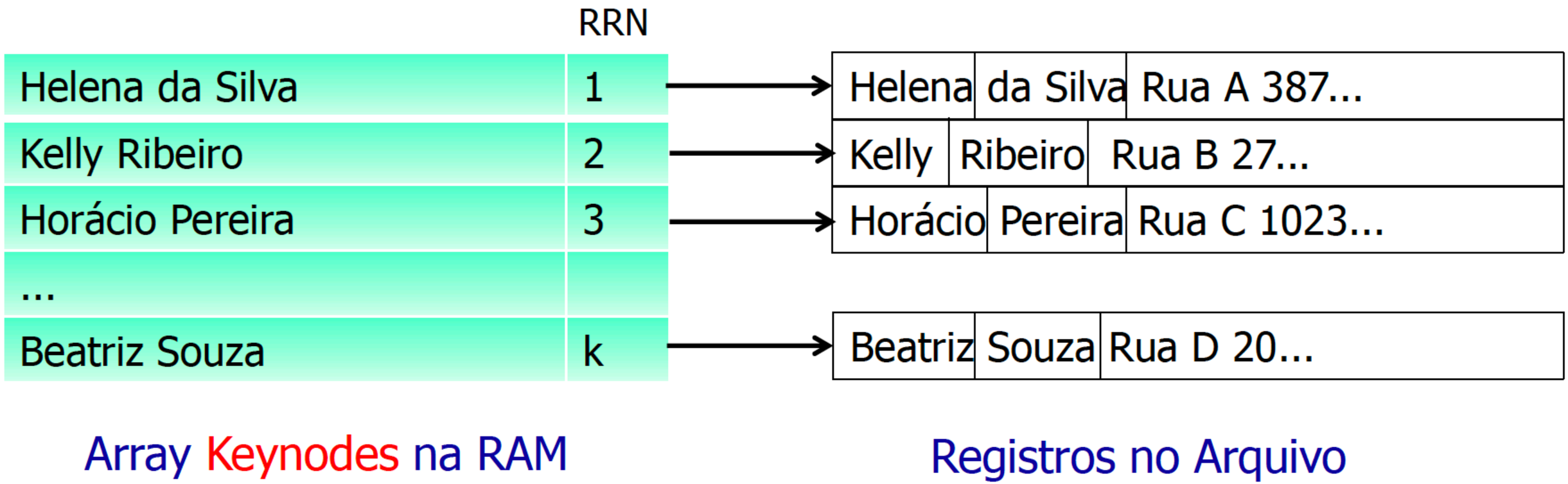
- Ordenação por chaves
- Ideia básica
  - Não é necessário que se armazenem todos os dados na memória principal para se conseguir a ordenação
  - Basta que se armazenem as chaves

Keysorting - Método

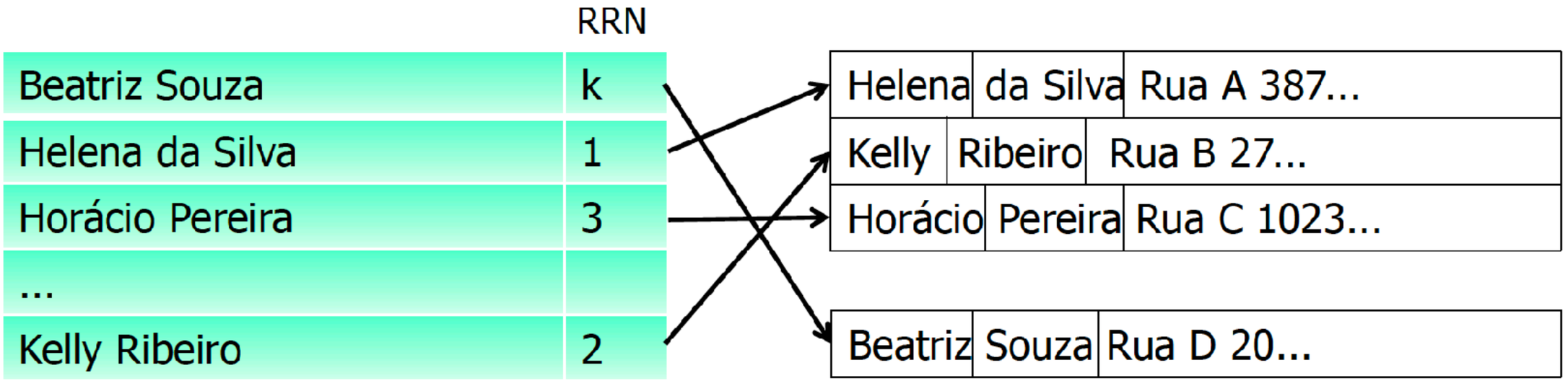
- 1. Cria-se na memória interna um vetor, em que cada posição tem uma chave do arquivo e um ponteiro para o respectivo registro no arquivo (RRN, se reg. tamanho fixo, ou byte inicial, se tam. variável)
- 2. Ordena-se o vetor na memória interna
- 3. Cria-se um novo arquivo com os registros na ordem em que aparecem no vetor ordenado na memória principal

unesp

Keysorting - antes de ordenar



Keysorting - após ordenar Keynodes



Array Keynodes na RAM

Registros no (in)Arquivo

para i = 1 até N (número de registros)
    seek o arquivo até o registro cujo RRN é Keynodes[i].RRN
    leia esse registro (in-buffer) na RAM
    escreva esse registro (out-buffer) no arquivo de saída

Keysorting - Limitações

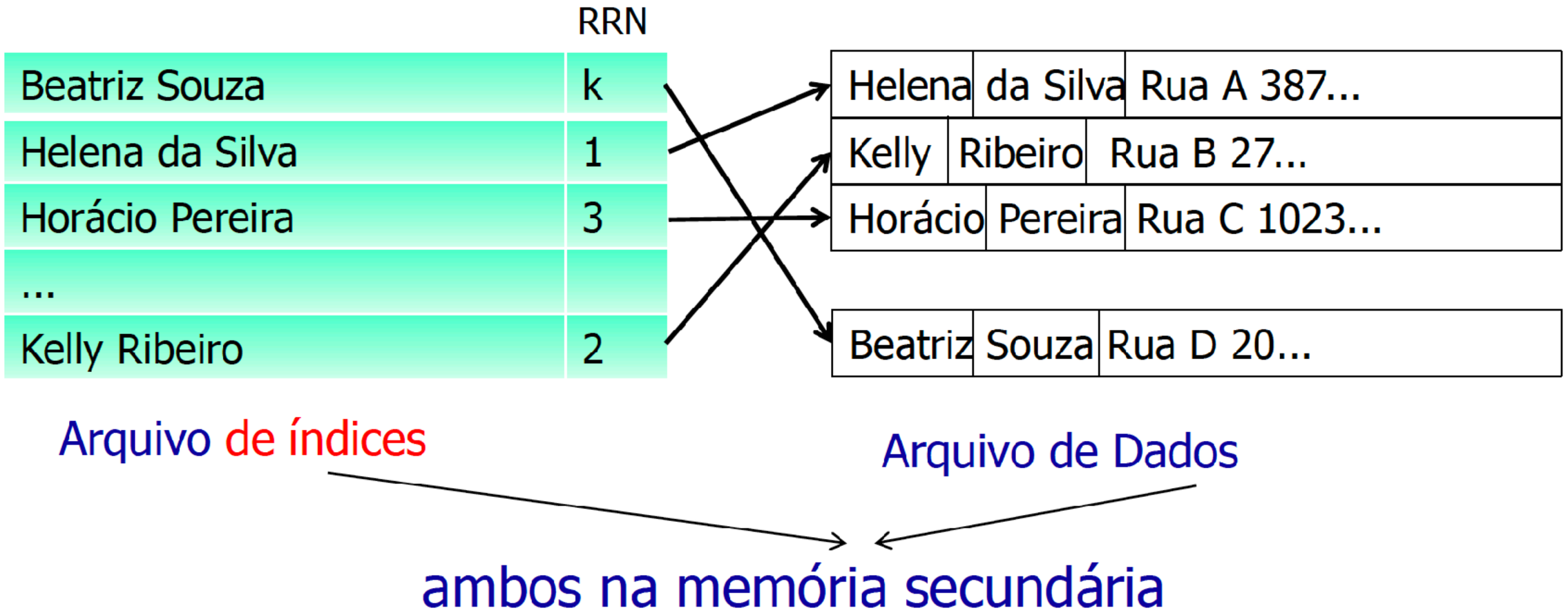
- Inicialmente, é necessário ler as chaves de todos os registros no arquivo
- Depois, para se criar o novo arquivo, devem-se fazer vários seeks no arquivo para cada posição indicada no vetor ordenado
  - Mais uma leitura completa do arquivo
  - Não é uma leitura sequencial
  - Alterna-se leitura no arquivo antigo e escrita no arquivo novo

unesp

Keysorting - Questões

- Por que criar um novo arquivo?
  - Temos que ler todos os registros (e não sequencialmente!) para reescrevê-los no novo arquivo!!
- Não vale a pena usar o vetor ordenado como um índice?

Usar Keynodes como Arquivo de Índices



Leitura recomenda

- Leitura recomendada: FOLK, M.J. File Structures, Addison-Wesley, 1992.

  Capítulos 5.

unesp

Referências

- FOLK, M.J. File Structures, Addison-Wesley, 1992.
- File Structures: Theory and Pratice”, P. E. Livadas, Prentice-Hall, 1990;
- Contém material extraído e adaptado das notas de aula dos professores Moacir Ponti, Thiago Pardo, Leandro Cintra e Maria Cristina de Oliveira.