# Algoritmos e Estruturas de Dados II
## Organização de Arquivos

Professora:
Josiane M. Bueno

## Sumário

- Organização de arquivos
  - Seqüência de bytes (Stream)
  - Estruturada
    - Campos
    - Registros
- Acesso a registros
  - Seqüencial
  - Direto
- Arquivos auto-descritivos e cabeçalhos

## Organização de Arquivos

- Arquivo = persistência de dados
- Dois tipos de organização:
  - *Fluxo (Stream)*, e
  - Estruturada (campos e registros).
- A organização do arquivo influi diretamente no modo de recuperar as informações nele armazenadas.
- Entretanto, a organização lógica, que não necessariamente corresponde à organização física

## Sumário

- Organização de arquivos
  - Seqüência de bytes (Stream)
  - Estruturada
    - Campos
    - Registros
- Acesso a registros
  - Seqüencial
  - Direto
- Arquivos auto-descritivos e cabeçalhos

## Seqüência de bytes (*stream*)

- Exemplo: Desejamos armazenar em um arquivo os nomes e endereços de várias pessoas:

| | | |
|---|---|---|
| Maria | João | Pedro |
| Rua 1, 123 | Rua A, 255 | Rua 10, 56 |
| São Carlos | Rio Claro | Rib. Preto |

Suponha que decidimos representar os dados como uma seqüência de bytes (sem delimitadores)

MariaRua 1123São CarlosJoãoRua A255Rio ClaroPedroRua 1056Rib. Preto

## Seqüência de bytes (*stream*)

- Uma vez escritas as informações, não existe como recuperar porções individuais (nome ou endereço)
- Desta forma, perde-se a integridade das unidades fundamentais de organização dos dados
  - Os dados são agregados de caracteres com significado próprio
  - Tais agregados são chamados campos (*fields*)

## Sumário

- Organização de arquivos
  - Seqüência de bytes (Stream)
  - Estruturada
    - Campos
    - Registros
- Acesso a registros
  - Seqüencial
  - Direto
- Arquivos auto-descritivos e cabeçalhos

## Organização em campos

- **Campo:**
  - **menor unidade lógica de informação** em um arquivo
  - uma noção lógica (ferramenta conceitual), não corresponde necessariamente a um conceito físico
- Existem várias maneiras de organizar um arquivo mantendo a identidade dos campos
  - A organização anterior não proporciona isso.

## Métodos para organização em campos

- Comprimento fixo
- Indicador de comprimento
- Delimitadores
- Uso de *tags*

## Métodos para organização em campos Exemplos

(a)

| | | | |
|---|---|---|---|
| Maria | Rua 1 | 123 | São Carlos |
| João | Rua A | 255 | Rio Claro |
| Pedro | Rua 10 | 56 | Rib. Preto |

(b)

05Maria05Rua 10312310São Carlos
04João05Rua A0325509Rio Claro
05Pedro06Rua 1002561 0Rib. Preto

(c)

Maria|Rua 1|123|São Carlos|
João|Rua A|255|Rio Claro|
Pedro|Rua 10|56|Rib. Preto|

(d)

Nome=Maria|Endereço=Rua 1|Número=123|Cidade=São Carlos|
Nome=João|Endereço=Rua A|Número=255|Cidade=Rio Claro|
Nome=Pedro|Endereço=Rua 10|Número=56|Cidade=Rib. Preto|

## Campos com tamanho fixo

- Cada campo ocupa no arquivo um tamanho fixo, pré-determinado (por ex. 4 bytes)
- O fato do tamanho ser conhecido garante que é possível recuperar cada campo

```
struct {
        char nome[10];
        char endereco[15];
        char numero[3];
        char cidade[15];
       } set_of_fields;
```

## Campos com tamanho fixo - Problemas

- O espaço alocado (e não usado) aumenta desnecessariamente o tamanho do arquivo (desperdício)
- Solução inapropriada quando se tem grande quantidade de dados com tamanho variável
- Razoável apenas se o comprimento dos campos é realmente fixo, ou apresenta pouca variação

## Campos com indicador de comprimento

- O tamanho de cada campo é armazenado imediatamente antes do dado
- Se o tamanho do campo é inferior a 256 bytes, o espaço necessário para armazenar a informação de comprimento é um único byte

## Campos separados por delimitadores

- Caractere(s) especial(ais) (que não fazem parte do dado) são escolhido(s) para ser(em) inserido(s) ao final de cada campo
- Ex.: para o campo *nome* pode-se utilizar /, tab, #, etc.
- Problema: pode ser difícil escolher um delimitador que não esteja dentro das possibilidades de valores de um campo.

## Uso de uma *tag* do tipo *keyword=value*

- Vantagem: o campo fornece informação (semântica) sobre si próprio
- Fica mais fácil identificar o conteúdo do arquivo e campos perdidos
- Desvantagem: as *keywords* podem ocupar uma porção significativa do arquivo

## Organização em registros

- **Registro**: um conjunto de campos agrupado
- Arquivo representado em um nível de organização mais alto.
- Assim como o conceito de campo, um registro é uma ferramenta conceitual, que não necessariamente existe no sentido físico. É um outro nível de organização imposto aos dados com o objetivo de preservar o significado.

## Métodos para organização em registros

- Tamanho fixo
- Número fixo de campos
- Indicador de tamanho
- Uso de índice
- Delimitadores

## Registros de tamanho fixo ou previsível - Exemplos

Registro de tamanho fixo e campos de tamanho fixo:

| | | | |
|---|---|---|---|
| Maria | Rua 1 | 123 | São Carlos |
| João | Rua A | 255 | Rio Claro |
| Pedro | Rua 10 | 56 | Rib. Preto |

Registro de tamanho fixo e campos de tamanho variável:

Maria|Rua 1|123|São Carlos| ← Espaço vazio →|
João|Rua A|255|Rio Claro| ← Espaço vazio →|
Pedro|Rua 10|56|Rib. Preto| ← Espaço vazio →|

Registro com número fixo de campos:

Maria|Rua 1|123|São Carlos|João|Rua A|255|Rio Claro|Pedro|Rua 10|56|Rib. Preto|

## Registros de tamanho fixo

- Um dos métodos mais comuns de organização de arquivos
- Analogamente ao conceito de campos de tamanho fixo, assume que todos os registros têm o mesmo número de bytes
- Pode-se ter registros de tamanho fixo com campos de tamanho fixo ou variável (uso de delimitadores)

## Registros com número fixo de campos

- Ao invés de especificar que cada registro contém um número fixo de bytes, podemos especificar um número fixo de campos
- O tamanho do registro, em bytes, é variável
- Neste caso, os campos seriam separados por delimitadores

## Registros de tamanho variável - Exemplos

Registro iniciados por indicador de tamanho:

28Maria|Rua 1|123|São Carlos|25João|Rua A|255|Rio Claro|27Pedro|Rua 10|56|Rib. Preto|

Arquivos de dados + arquivo de índices:

Dados: Maria|Rua 1|123|São Carlos|João|Rua A|255|Rio Claro|Pedro|Rua 10|56|Rib. Preto|
Índice: 00 29 44

Registro delimitado por marcador (#):

Maria|Rua 1|123|São Carlos|#João|Rua A|255|Rio Claro|#Pedro|Rua 10|56|Rib. Preto|

## Indicador de tamanho para registros

- O indicador que precede o registro fornece o seu tamanho total, em bytes
- Os campos são separados internamente por delimitadores
- Boa solução para registros de tamanho variável

## Utilizar um índice

- Um índice externo poderia indicar o deslocamento de cada registro relativo ao início do arquivo
- Pode ser utilizado também para calcular o tamanho dos registros.
- Os campos são separados por delimitadores

## Utilizar delimitadores

- Separar os registros com delimitadores análogos aos de fim de campo
- O delimitador de campos é mantido, sendo que o método combina os dois delimitadores
- Note que delimitar fim de campo é diferente de delimitar fim de registro

## Sumário

- Organização de arquivos
  - Seqüência de bytes (Stream)
  - Estruturada
    - Campos
    - Registros
- Acesso a registros
  - Seqüencial
  - Direto
- Arquivos auto-descritivos e cabeçalhos

## Chaves

- Uma chave (*key*) está associada a um registro e permite a sua recuperação
- O conceito de chave é também uma ferramenta conceitual importante

## Chaves Primária e Secundária

- Uma **chave primária** é, por definição, a chave utilizada para identificar unicamente um registro
  - Ex. nro. USP, CPF, RG, ...
  - Sobrenome, por outro lado, não é uma boa escolha para chave primária...
- Uma **chave secundária**, tipicamente, não identifica unicamente um registro, e pode ser utilizada para buscas simultâneas por várias chaves (todos os "**Silvas**" que moram em **São Paulo**, por exemplo).

## Chaves Distintas

- O ideal é que exista uma relação um a um entre chave e registro.
- Se isso não acontecer, é necessário fornecer uma maneira do usuário decidir qual dos registros é o que interessa
- A solução mais simples consiste em evitar tais confusões garantindo que a cada chave corresponda um único registro

## Escolha da Chave Primária

- A chave primária deve ser *"dataless"*, isto é, não deve ter um significado associado, e não deve **mudar nunca** (outra razão para não ter significado)
- Uma mudança de significado pode implicar na mudança do valor da chave, o que invalidaria referências já existentes baseadas na chave antiga

## Forma canônica da chave

- "Ana", "ANA", ou "ana" devem levar ao mesmo registro
- **Formas canônicas** para as chaves: uma única representação da chave que conforme com uma regra.
- Ex: A regra pode ser, ´todos os caracteres maiúsculos´.
  - Nesse caso a forma canônica da chave será ANA

## Busca seqüencial

- Busca pelo registro que tem uma determinada chave, em um arquivo
  - Lê o arquivo, registro a registro, em busca de um registro contendo um certo valor de chave

## Desempenho da Busca Seqüencial

- Na busca em RAM, normalmente consideramos como medida do trabalho necessário o número de comparações efetuadas para obter o resultado da pesquisa
- No contexto de pesquisa em arquivos, o acesso a disco é a operação mais cara e, portanto, o número de acessos a disco efetuados é utilizado como medida do trabalho necessário para obter o resultado
- **Mecanismo de avaliação do custo associado ao método**: contagem do número de chamadas à função de baixo nível READ()

## Desempenho da Busca Seqüencial

- **Exemplo:**

  Supondo que cada chamada a READ lê 1 registro, e requer um *seek* (supondo que todas as chamadas a READ têm o mesmo custo).

  Uma busca seqüencial por "ANA" em 2.000 registros requer em média, 1.000 leituras (1 se for o primeiro registro – melhor caso; 2.000 se for o último – pior caso; e 1.000, em média).

  Em geral, o trabalho necessário para buscar um registro em um arquivo de tamanho n utilizando busca seqüencial é O(n).

## Blocagem de Registros

- A parte mais lenta de uma operação de acesso a disco é o *seeking*
- A transferência dos dados, uma vez iniciada, é relativamente rápida, apesar de muito mais lenta que uma transferência de dados em RAM
- O custo de buscar e ler um registro, e depois buscar e ler outro, é maior que o custo de buscar (e depois ler) dois registros sucessivos de uma só vez
- **Pode-se melhorar o desempenho da busca seqüencial lendo um bloco de registros por vez, e então processar este bloco em RAM**

## Exemplo de blocagem

- Um arquivo com 4.000 registros cujo tamanho médio é 512 bytes cada
- A busca seqüencial por um registro, sem blocagem, requer em média 2.000 leituras
- Trabalhando com blocos de 16 registros, o número médio de leituras necessárias cai para 125
- Cada READ gasta um pouco mais de tempo, mas o ganho é considerável devido à redução do número de READs (ou seja, de *seeks*)

## Blocagem de registros

- melhora o desempenho, mas o custo continua diretamente proporcional ao tamanho do arquivo, ou seja, O(n)
- reflete a diferença entre o custo de acesso à RAM e o custo de acesso a disco
- não altera o número de comparações em RAM
- aumenta a quantidade de dados transferidos entre o disco e RAM
- economiza tempo porque reduz o número de operações de *seeking*

## Vantagens da Busca Seqüencial

- Fácil de programar
- Requer estruturas de arquivos simples

## Busca seqüencial é razoável

- Na busca por uma cadeia em um arquivo ASCII
- Em arquivos com poucos registros (da ordem de 10)
- Em arquivos pouco pesquisados
- Na busca por registros com um certo valor de chave secundária, para a qual se espera muitos registros (muitas ocorrências)

## Sumário

- Organização de arquivos
  - Seqüência de bytes (Stream)
  - Estruturada
    - Campos
    - Registros
- Acesso a registros
  - Seqüencial
  - Direto
- Arquivos auto-descritivos e cabeçalhos

## Acesso Direto

- A alternativa mais radical ao acesso seqüencial é o **acesso direto**
- O acesso direto implica em realizar um *seeking* direto para o início do registro desejado (ou do setor que o contém) e ler o registro imediatamente
- É O(1), pois um único acesso traz o registro, independentemente do tamanho do arquivo

## Posição do início do registro

- Para localizar a posição exata do início do registro no arquivo, pode-se utilizar um arquivo índice separado
- Ou pode-se ter um **RRN (*relative record number*) (ou byte offset)** que fornece a posição relativa do registro dentro do arquivo

## Posição de um registro com RRN

- Para utilizar o RRN, é necessário trabalhar com registros de tamanho fixo
  - Nesse caso, a posição de início do registro é calculada facilmente a partir do seu RRN:
    *Byte offset* = RRN * Tamanho do registro
  - Por exemplo, se queremos a posição do registro com RRN 546, e o tamanho de cada registro é 128, o *Byte offset* é 546 x 128 = 69.888

## Acesso a arquivos X Organização de arquivos

- **Organização de Arquivos**
  - registros de tamanho fixo
  - registros de tamanho variável
- **Acesso a arquivos**
  - acesso seqüencial
  - acesso direto

## Acesso a arquivos X Organização de arquivos

- Considerações a respeito da organização do arquivo
  - arquivo pode ser dividido em campos?
  - os campos são agrupados em registros?
  - registros têm tamanho fixo ou variável?
  - como separar os registros?
  - como identificar o espaço utilizado e o "lixo"?
- Existem muitas respostas para estas questões
  - a escolha de uma organização em particular depende, entre outras coisas, do que se vai fazer com o arquivo

## Acesso a arquivos X Organização de arquivos

- Arquivos que devem conter registros com tamanhos muito diferentes, devem utilizar registros de tamanho variável
- Como acessar esses registros diretamente?
  - Arquivo de índice
- Existem também as limitações da linguagem
  - C permite acesso a qualquer byte, e o programador pode implementar acesso direto a registros de tamanho variável
  - Pascal exige que o arquivo tenha todos os elementos do mesmo tipo e tamanho, de maneira que acesso direto a registros de tamanho variável é difícil de ser implementado

## Sumário

- Organização de arquivos
  - Seqüência de bytes (Stream)
  - Estruturada
    - Campos
    - Registros
- Acesso a registros
  - Seqüencial
  - Direto
- Arquivos auto-descritivos e cabeçalhos

## Registro Cabeçalho (*header record*)

- Em geral, é interessante manter algumas informações sobre o arquivo para uso futuro. Essas informações podem ser mantidas em um *header* no início do arquivo
- Algumas informações típicas são:
  - número de registros
  - tamanho de cada registro
  - campos de cada registro
  - datas de criação e atualização
  - A existência de um registro *header* torna um arquivo um objeto auto-descrito. O software pode acessar arquivos de forma mais flexível

## Arquivos auto-descritivos e cabeçalhos

- É possível colocar informações elaboradas nos cabeçalhos dos arquivos, de modo que o arquivo fique auto-descritivo
- Exemplo de informações no cabeçalho
  - nome de cada campo
  - largura de cada campo
  - número de campos por registro
  - quais campos são opcionais

## Metadados

- **São dados que descrevem os dados primários em um arquivo**
- Exemplo: Formato FITS(Flexible Image Transport System)
  - Armazena imagens astronômicas
  - Um cabeçalho FITS é uma coleção de blocos de **2880 bytes** contendo registros de **80 bytes** ASCII, no qual cada registro contém um metadado
  - 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 perpixel
    NAXIS = 2 / Number of axes
    NAXIS1 = 256 / Ra axis dimension
    NAXIS2 = 256 / Dec axis dimension
    .
    .
    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-explicável´)
  - Portabilidade
    - define-se um padrão para todos os que geram/acessam o arquivo.
    - PDF, PS, HTML, TIFF
    - permite conversão entre padrões

## Bibliografia


- Folk & Zoelick, File Structures;
- Transparências Leandro C. Cintra e M.C.F. de Oliveira.