Estruturas de Dados em Arquivos: Organização de Arquivos

Prof. Paulo H. R. Gabriel

Créditos

- Parte desse material foi baseado nas aulas de:
  - Moacir Ponti Jr.
  - Thiago A. S. Pardo
  - Leandro C. Cintra
  - Maria C. F. de Oliveira

- E no livro:
  - "Algoritmos e Estruturas de Dados em Arquivos", de Marcos André S. Kutova (2017)

Arquivos, Campos e Registros

- Agenda de contatos do celular

- Nomes e telefones são armazenados em alguma região de memória
- São protegidos contra acesso indevido

- Essa área de memória é um **arquivo**

Arquivos, Campos e Registros

- Cada contato é armazenado em um **registro**
- Registros contém **campos**
  - Ao menos dois: nome e telefone
- Diversas estruturas de dados podem ser usadas aqui
- Aspectos a ser considerados:
  - Forma de acesso
  - Velocidade de acesso
  - Espaço necessário

Estruturas de Campos

Campos numéricos

- Relembrando:
  - 1 byte possui 8 bits
  - 1 caractere ocupa 1 byte

- Ou seja: se armazenarmos um inteiro (de 0 a 9) em forma de caractere, cada byte armazena um único dígito
- **Porém:** se usarmos os bits para representar o inteiro diretamente, então podemos armazenar números entre 0 e 255 [*por quê?*]

Estruturas de Campos

Campos numéricos

- Logo, cada byte extra permite:
  - Armazenar 10 números a mais, no formato de caractere
  - Armazenar 256 números a mais, no formato binário

	Capacidade (texto)	Capacidade (inteiro)
1 byte	10	256
2 bytes	100	65.536
3 bytes	1.000	16.777.216

Estruturas de Campos

Campos numéricos

- Logo, cada byte extra permite:
  - Armazenar 10 números a mais, no formato de caractere
  - Armazenar 256 números a mais, no formato binário

- Quantos bytes eu preciso para armazenar o inteiro 1550?
  - Similar à conversão de um número para binário
  - Mas, usando como quociente o 256!

Estruturas de Campos

Campos numéricos

- 1550 / 256:
  - Resultado: 6
  - Resto: 14
  - Ou seja, precisamos de 2 bits: | 6 | 14 |
- Para converter novamente: $6 * 256^1 + 14 * 256^0$

Como organizar os campos?

- Comprimento fixo
- Indicador de comprimento
- Delimitadores
- Uso de etiquetas (*tags*)

Campos de tamanho fixo

```
struct reg {
        char nome[10];
        char sobre[10];
        char cidade[15];
        char estado[2];
        char cep[9];
};
```

Campos de tamanho fixo

- Vantagem: fácil de manipular
- Desvantagem: desperdício de espaço
  - Solução inapropriada quando se tem uma grande quantidade de dados com tamanho variável
  - Razoável apenas se o comprimento dos campos é realmente fixo ou apresenta pouca variação
- Porém...
  - Não há aumento considerável no tempo de leitura dos dados
  - O sistema operacional não lê byte por byte, mas todo o setor
  - E a navegação é mais simples

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 *[por quê?]*

05Paulo07Gabriel05Rua103123 10Uberlândia

  - *Desvantagens?*

Campos separados por delimitadores

- Uso de caracteres especiais
  - Não podem fazer parte dos dados armazenados
  - Inseridos ao final de cada campo

Paulo|Gabriel|Rua 1|123|Uberlândia|

- Limitação: pode ser que o usuário digite (acidentalmente ou não) o caractere especial
  - Alternativa interessante: usar '\0' ou '\n'

Campos com etiquetas

- Vantagem: o campo fornece informação semântica sobre si próprio
  - Fica mais fácil identificar o conteúdo do arquivo
  - Fica mais fácil identificar campos perdidos

- Desvantagem: as *tags* podem ocupar uma porção significativa do arquivo

nome=Paulo|sobrenome=Gabriel|endereço=Rua1|numero=123|cidade=Uberlândia|

Estruturas de Registros

- Registro: conjunto de campos agrupados

- O arquivo representado em um nível de organização mais alto
  - Trata-se de outro nível de organização imposto aos dados com o objetivo de preservar o significado

- Assim como o conceito de campo, um registro é uma ferramenta conceitual
  - Ou seja, não existe, necessariamente, no sentido físico

Estruturas de Registros

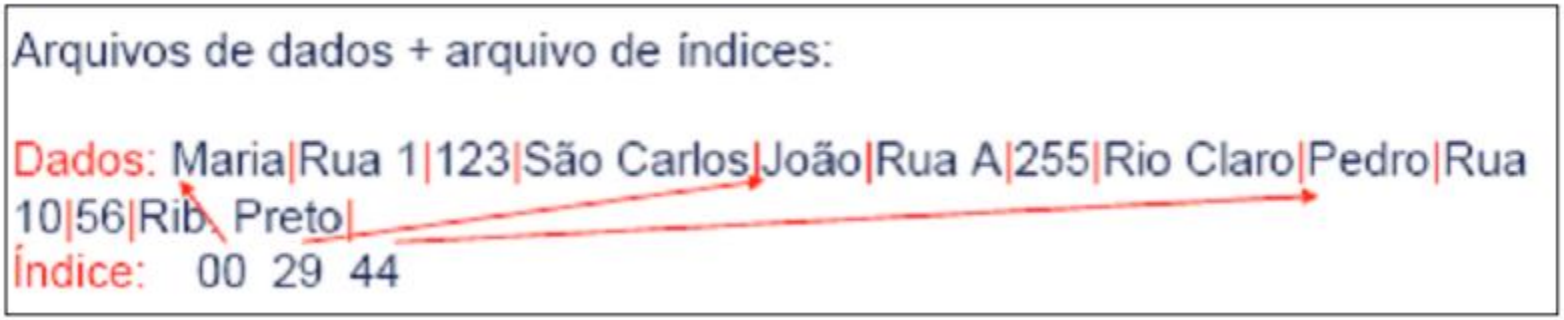
A organização de registros segue o mesmo padrão da organização de campo

- Registros de tamanho fixo
  - Mais comum
- Indicador de comprimento
  - Pode trazer ambiguidades
- Delimitadores
  - Diferente do delimitador de campo!
- Uso de etiquetas (*tags*)
  - Pode aumentar o tamanho do arquivo

Estruturas de Registros

- Outra opção: uso de índices!

- 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 seriam separados por delimitadores



Acesso a Registros

- Quando fazemos uma pesquisa em memória RAM, normalmente adotamos como medida do trabalho necessário o número de comparações efetuadas para obter o resultado da pesquisa
- Na pesquisa em arquivos, o acesso a disco é a operação mais cara
- Portanto, o número de acessos a disco efetuados é adotado 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

Algumas suposições:

- Cada chamada a READ lê um registro e requer um *seek*
- Todas as chamadas a READ têm o mesmo custo

Busca Sequencial

- Nessa busca, procuramos pelo registro que tem uma determinada *chave* em um arquivo
  - A chave é um valor único associado a cada registro
- Lemos o arquivo registro a registro, em busca de um registro contendo um certo valor de chave
- Vantagens:
  - Fácil de programar
  - Requer estruturas de arquivos simples

Exemplo de Busca Sequencial

- Queremos buscar um registro em um arquivo com 2.000 registros

- Precisamos, em média, de 1.000 leituras
  - 1 leitura se for o primeiro registro
  - 2.000 se for o último
  - 2.000/2 = 1.000, em média (supondo igual probabilidade de busca por qualquer registro)

- No pior caso, o trabalho necessário para buscar um registro em um arquivo é proporcional a n

Blocagem de Registros

- Sabemos que a operação *seek* é lenta
- Porém, a transferência dos dados do disco para a RAM é relativamente rápida...
  - (ainda 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
- Assim, pode-se melhorar o desempenho da busca sequencial lendo um bloco de registros por vez e, só então, processar este bloco em RAM

Exemplo de Blocagem

- Seja um arquivo com 4.000 registros cujo tamanho médio é 512 bytes cada
  - A busca sequencial 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 (são 250 blocos)
- Cada READ gasta um pouco mais de tempo, mas o ganho é considerável devido à redução do número da operação *seek*

Blocagem de Registros

- Melhora o desempenho, mas o custo continua diretamente proporcional ao tamanho do arquivo
- Reflete a diferença entre o custo de acesso à RAM e o custo de acesso a disco
  - Aumenta a quantidade de dados transferidos entre o disco e RAM
- Não altera o número de comparações em RAM
- Economiza tempo porque reduz o número de operações *seek*

Blocagem de Registros

Atenção:

- Agrupam-se bytes em campos, campos em registros e, agora, registros em blocos
  - Os níveis de organização hierárquica estão aumentando!

- Entretanto, agrupar registros em blocos aumenta apenas o desempenho
  - Os demais agrupamentos se relacionam à organização lógica da informação

Acesso Direto

- A alternativa mais radical ao acesso sequencial é o acesso direto
- O acesso direto implica em realizar um seek direto para o início do registro (ou setor) desejado e ler o registro imediatamente
- O trabalho necessário é constante, pois um único acesso traz o registro, independentemente do tamanho do arquivo