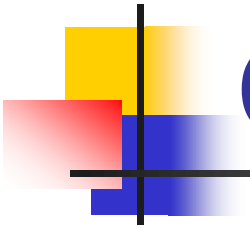


Organização de Arquivos

Leandro C. Cintra

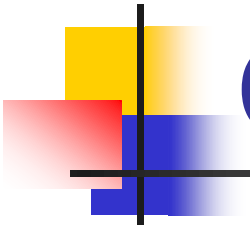
M.C.F. de Oliveira

Rosane Minghim

2004 - 2013

Fonte: Folk & Zoelick, File Structures

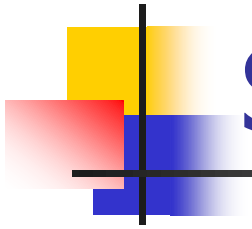
PAE 2012: Rafael M. Martins
PAE 2013: Renato Oliveira



Organização de Arquivos

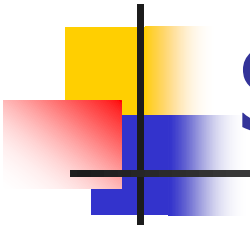
- Informações em arquivos são, em geral, organizadas logicamente em campos e registros
- Entretanto, campos e registros são conceitos lógicos, que não necessariamente correspondem a uma organização física
- Dependendo de como a informação é mantida no arquivo, campos lógicos sequer podem ser recuperados

Sequência de bytes (*stream*)



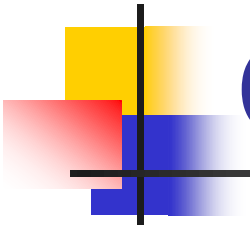
- Exemplo:
  - Suponha que desejamos armazenar em um arquivo os nomes e endereços de várias pessoas
  - Suponha que decidimos representar os dados como uma sequência de bytes (sem delimitadores, contadores, etc.)

AmesJohn123 MapleStillwaterOK74075MasonAlan90
EastgateAdaOK74820



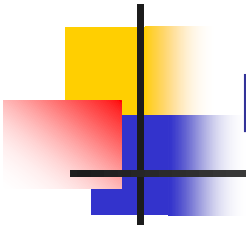
Sequência de bytes (*stream*)

- 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*)



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

(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|

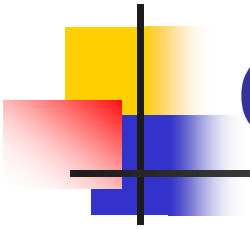
(d)

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

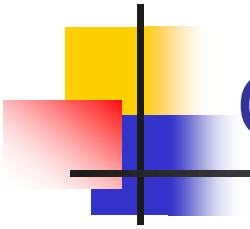
- Cada campo tem tamanho pré-determinado
- Recuperação facilitada

```
struct {
        char last[10];
        char first[10];
        char city[15];
        char state[2];
        char zip[9];
} set_of_fields;
```



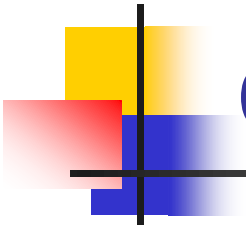
Campos com tamanho fixo

- Espaço alocado e não usado = desperdício
- Ruim para campos de dados com tamanho variável
- Razoável quando comprimento fixo ou com pouca variação



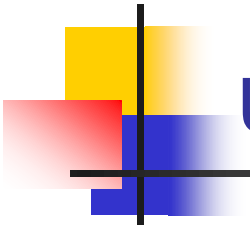
Campos com indicador de comprimento

- Tamanho de cada campo antes do dado
- Se tamanho do campo < 256 bytes
  - um único byte para indicar o comprimento



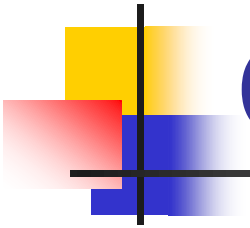
Campos separados por delimitadores

- Caracteres especiais inseridos ao final de cada campo
- Ex.: /, *tab*, #, etc...
- Espaços em branco geralmente não servem



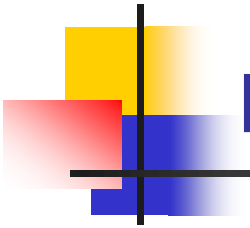
Uso de uma *tag* do tipo "keyword=value"

- Vantagem: informação (semântica)
- Facilidade de identificação de conteúdo do arquivo
- Facilidade de identificação de campos perdidos
- Possibilidade de padronização (html, XML, ...)
- Desvantagem: *keywords* ocupam espaço



Organização em registros

- **Registro**: conjunto de campos agrupado
- Nível de organização do arquivo é mais alto
- Também é uma ferramenta lógica, não física



Métodos para organização em registros

- Tamanho fixo
- Número fixo de campos
- Indicador de tamanho
- Uso de índice
- Utilizar delimitadores

Registros de tamanho fixo

Registro de tamanho fixo e campos de tamanho fixo:

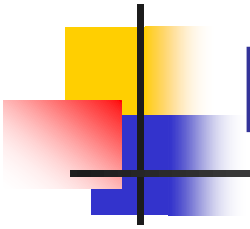
Maria	Rua 1	123	São Carlos
João	Rua A	255	Rio Claro
Pedro	Rua 10	56	Rib. Preto

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 →|

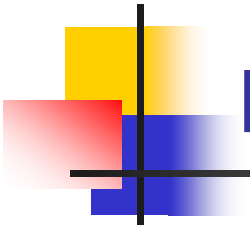
Registro com número fixo de campos:

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



Registros de tamanho fixo

- Todos os registros têm o mesmo número de bytes
- Muito comum
- Possível registros de tamanho fixo com campos de tamanho variável



Registros com número fixo de campos

- Ao invés de números fixo de bytes, número fixo de campos
- O tamanho do registro é variável
- Campos separados por delimitadores

Registros de tamanho variável

Registro iniciados por indicador de tamanho:

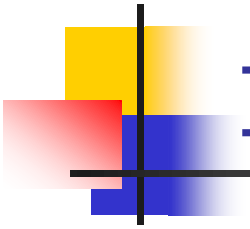
28Maria|Rua 1|123|São Carlos|25João|Rua A|255|Rio Claro|27Pedro|Rua 10|56|Rib. Preto|

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

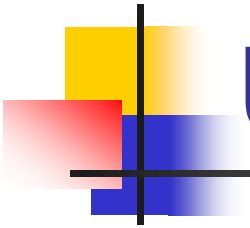
Registro delimitado por marcador (#):

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



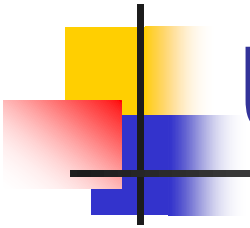
Indicador de tamanho para registros

- 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



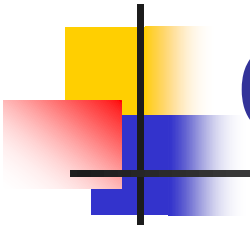
Utilizar um índice

- 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 seriam separados por delimitadores...



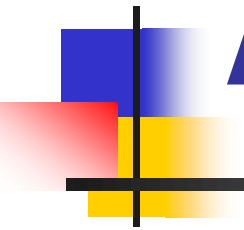
Utilizar delimitadores

- 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

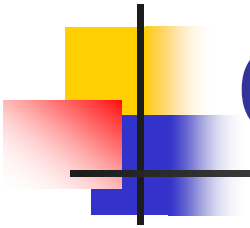


Observação

- Ver programas em C e Pascal no material didático na Web e nos livros texto (*Folk 92, 98*) que ilustram a criação de arquivos com essas várias organizações

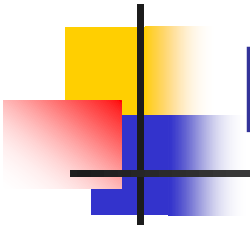


Acesso a registros



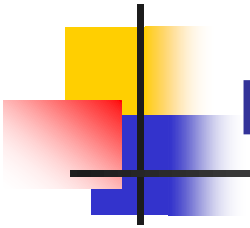
Chaves

- Chave (*key*) associada a um registro
  - **CONCEITO IMPORTANTE**
- **Chave primária:** identifica unicamente um registro
  - Ex.: Nº USP, CPF, RG, etc.
- **Chave secundária:** pode ser utilizada para buscas simultâneas por vários registros (todos os registros de mesma chave)
  - Tipicamente, não identifica unicamente um registro
- Qualquer uma pode ser chave única, mas a primária **TEM** que ser única



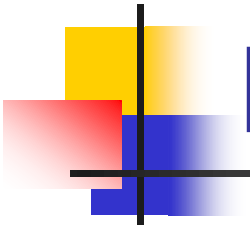
Escolha de chaves

- A chave primária deve ser *"dataless"*, isto é, sem significado
- Mudança de significado pode implicar na mudança do valor da chave
- "Ana", "ANA", ou “ana" devem levar ao mesmo registro
- **Formas canônicas** para as chaves: representação obedece uma regra



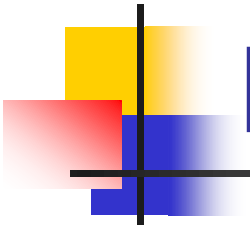
Busca: Desempenho da Busca Sequencial

- Em RAM: **número de comparações** efetuadas para obter o resultado da pesquisa
- Em arquivos: **número de acessos a disco** necessários para obter o resultado
- **Mecanismo de avaliação do custo associado ao método**: contagem do número de chamadas à função de baixo nível READ()



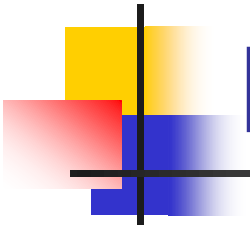
Blocagem de Registros

- 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 sequencial lendo um bloco de registros por vez, depois processar este bloco em RAM**



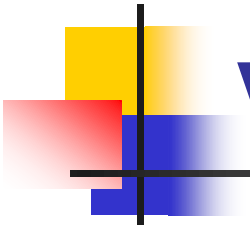
Exemplo de blocagem

- 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
- Cada READ gasta um pouco mais de tempo, mas o ganho é considerável devido à redução do número de READs (ou seja, de *seeks*)



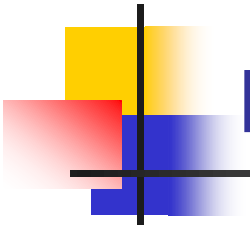
Blocagem de registros

- Melhora o desempenho, mas o custo continua diretamente proporcional ao tamanho do arquivo, i.e., é 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*



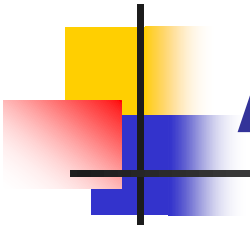
Vantagens da Busca Sequencial

- Fácil de programar
- Requer estruturas de arquivos simples



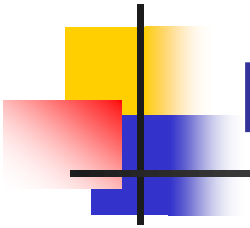
Busca sequencial é razoável

- 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)



Acesso Direto

- A alternativa mais radical ao acesso sequencial 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

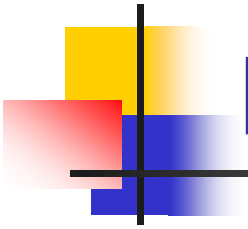


Posição do início do registro

- Para localizar a posição exata do início do registro no arquivo, pode-se utilizar um arquivo de índice separado

- Ou pode-se ter um **RRN (*relative record number*)** que fornece a posição relativa do registro dentro do arquivo

Posição de um registro com RRN

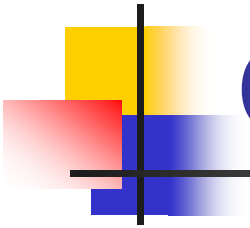


- Para utilizar o RRN, é necessário trabalhar com registros de tamanho fixo
  - Nesse caso, a posição de início do registro é calculada facilmente a partir do seu RRN:
  - *Byte offset* = RRN * Tamanho do registro
  - Por exemplo, se queremos a posição do registro com RRN 546, e o tamanho de cada registro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

Acesso a arquivos X Organização de arquivos

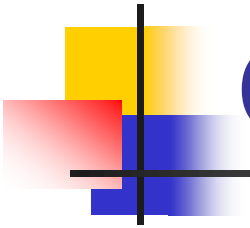
- **Organização de Arquivos**
  - registros de tamanho fixo
  - registros de tamanho variável
- **Acesso a arquivos**
  - acesso sequencial
  - acesso direto

Acesso a arquivos X Organização de arquivos

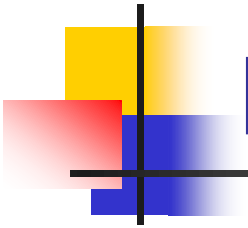


- Considerações a respeito da organização do arquivo
  - Arquivo pode ser dividido em campos?
  - Os campos são agrupados em registros?
  - Registros têm tamanho fixo ou variável?
  - Como separar os registros?
  - Como identificar o espaço utilizado e o "lixo"?
- Existem muitas respostas para estas 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

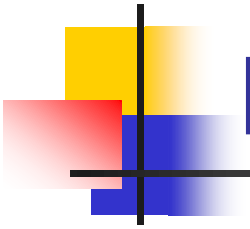


- Arquivos que devem conter registros com tamanhos muito diferentes, devem utilizar registros de tamanho variável
- Como acessar esses registros diretamente?
- Existem também limitações da linguagem
  - C permite acesso a qualquer byte, e o programador pode implementar acesso direto a registros de tamanho 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



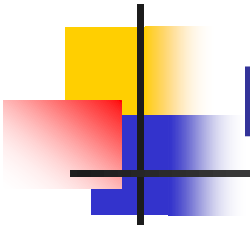
Modelos Abstratos de Dados

- **Objetivo:** Focar no conteúdo da informação, ao invés do seu formato físico
- As informações atuais tratadas pelos computadores (som, imagens, etc.) não se ajustam bem à metáfora de dados armazenados como sequências de registros separados em campos



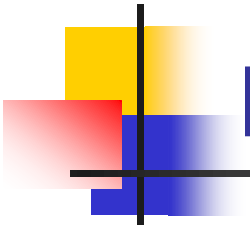
Modelos Abstratos de Dados

- 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



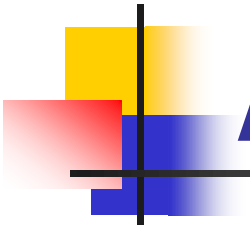
Registro Cabeçalho (*header record*)

- 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
- Algumas informações típicas são:
  - número de registros
  - tamanho de cada registro
  - campos de cada registro
  - datas de criação e atualização



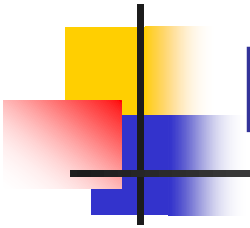
Registro Cabeçalho (*header record*)

- A existência de um registro cabeçalho torna um arquivo um objeto auto-descrito
- O software pode acessar arquivos de forma mais flexível



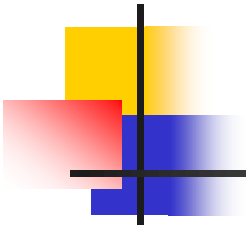
Arquivos auto-descritivos e cabeçalhos

- 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

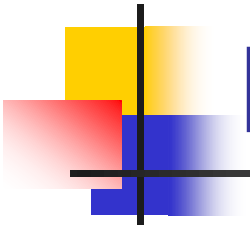
- **São dados que descrevem os dados primários em um arquivo**



Metadados

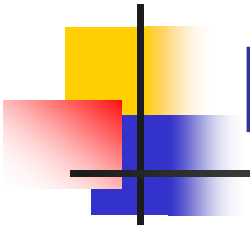
Exemplo: Formato FITS (Flexible Image Transport System)

- Armazena imagens de astronomia
- 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 per pixel
  NAXIS = 2 / Number of axes
  NAXIS1 = 256 / Ra axis dimension
  NAXIS2 = 256 / Dec axis dimension
  ...
  END



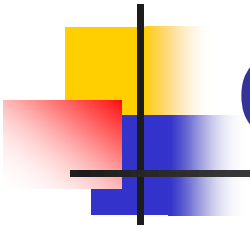
Metadados

- 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 certos tipos de arquivo
    - PDF, PS, HTML, TIFF
  - Permite conversão entre padrões



Recuperação dinâmica

- O procedimento de compactação é esporádico, i.e., um registro apagado não fica disponível para uso imediatamente
- Em aplicações interativas que acessam arquivos altamente voláteis, pode ser necessário um processo dinâmico de recuperação de espaços vazios
  - Marcar registros apagados
  - Identificar e localizar os espaços antes ocupados por esses registros, sem buscas exaustivas



Como localizar os espaços vazios?

- Registros de tamanho fixo
  - Lista encadeada de registros eliminados (Dispo)
  - Lista constitui-se de espaços vagos, endereçados por meio de seus RRNs
  - Cabeça da lista está no *header* do arquivo
  - Um registro eliminado contém o RRN do próximo registro eliminado
  - Inserção e remoção ocorrem sempre no início da lista (pilha!)

Registros de tamanho fixo

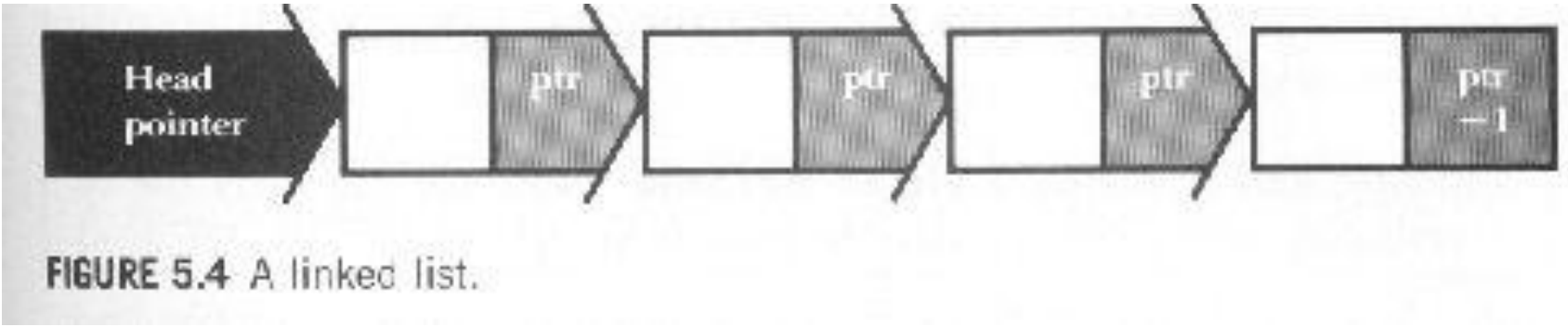
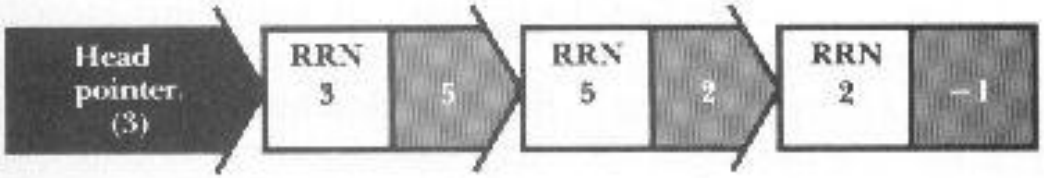


FIGURE 5.4 A linked list.



Pilha antes e depois da inserção do nó correspondente ao registro de RRN 3

Exemplo

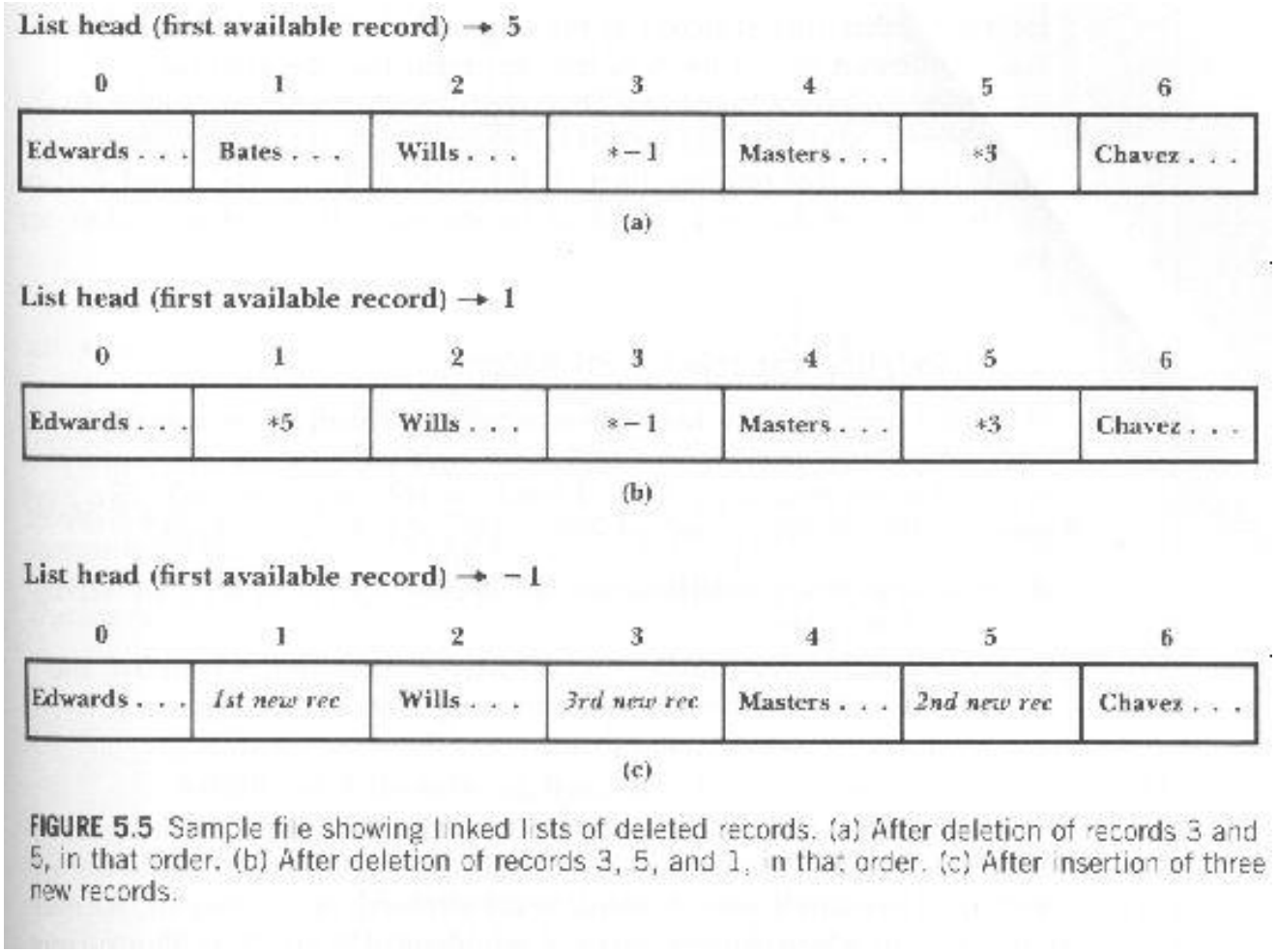
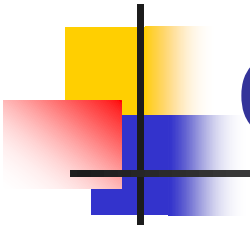


FIGURE 5.5 Sample file showing linked lists of deleted records. (a) After deletion of records 3 and 5, in that order. (b) After deletion of records 3, 5, and 1, in that order. (c) After insertion of three new records.



Como localizar os espaços vazios?

- Registros de tamanho variável
  - É necessário uma busca sequencial na lista para encontrar uma posição com espaço suficiente
- Estratégias de alocação
  - First-fit
  - Best-fit
  - Worst-fit

Remoção de registros

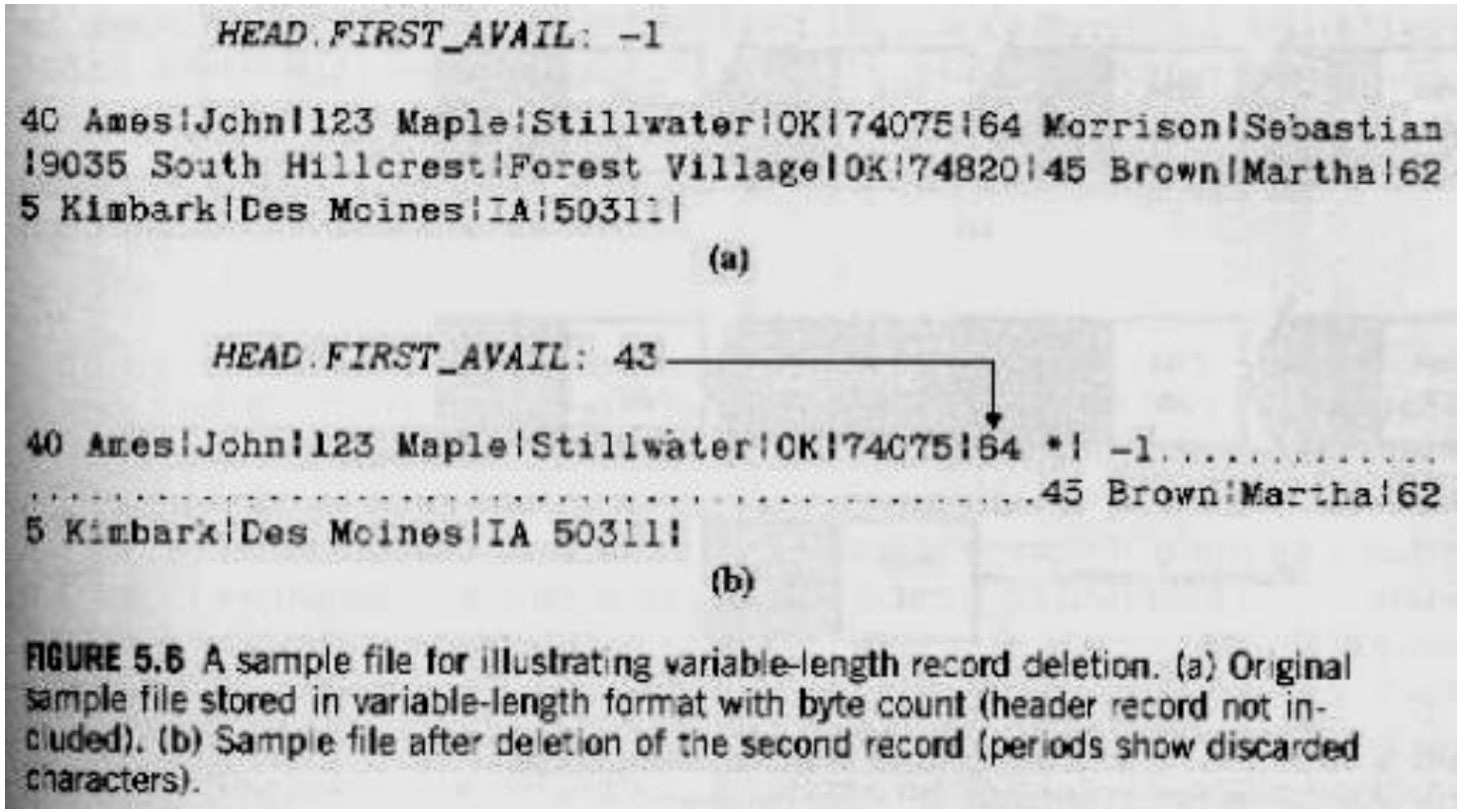
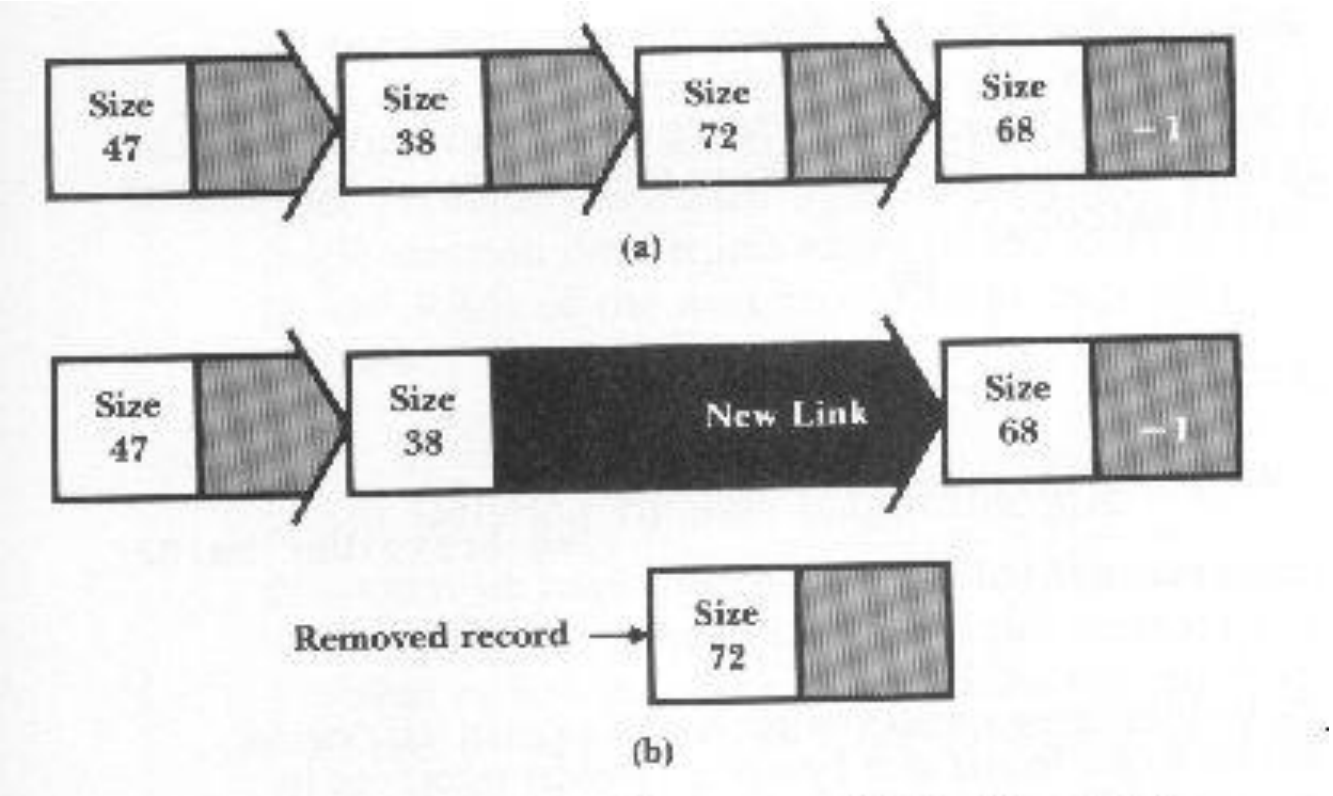


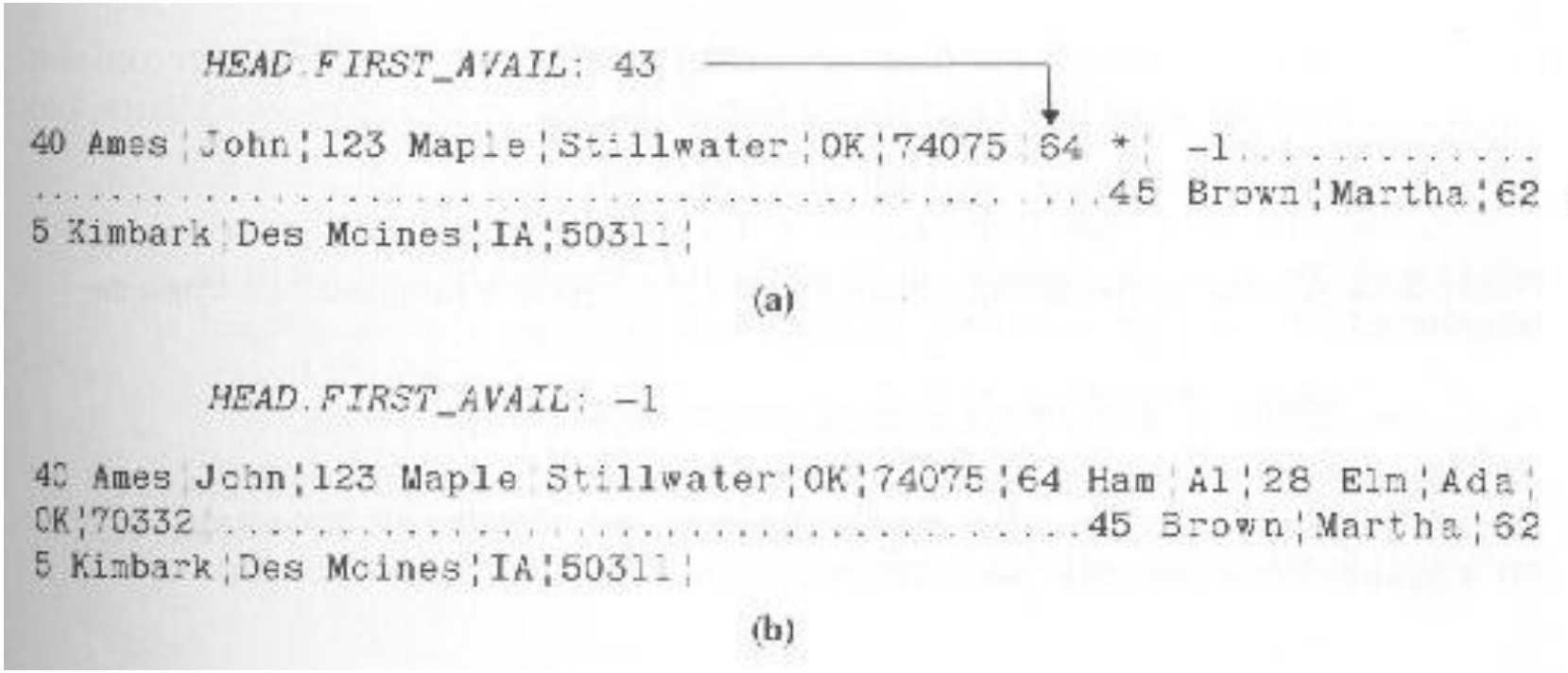
FIGURE 5.6 A sample file for illustrating variable-length record deletion. (a) Original sample file stored in variable-length format with byte count (header record not included). (b) Sample file after deletion of the second record (periods show discarded characters).

Remoção da lista



Remoção de registro de tamanho variável.
(a) Antes da remoção. (b) Depois da remoção.

Fragmentação



(a) Depois da remoção do segundo registro (caracteres não-utilizados são substituídos por pontos).
(b) Depois da adição do novo registro.

Combatendo a fragmentação

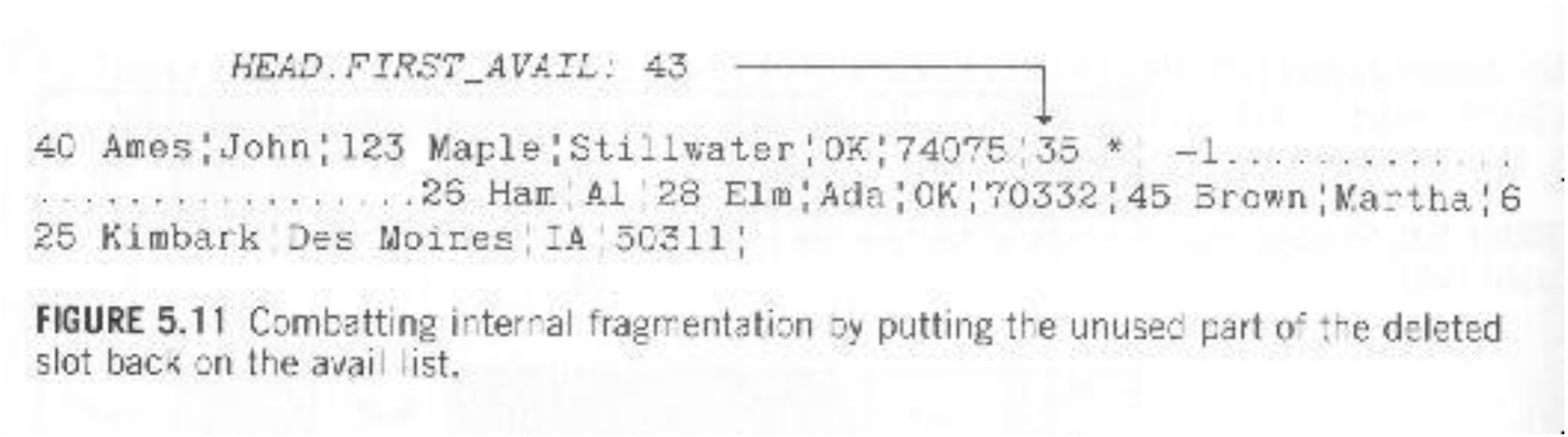
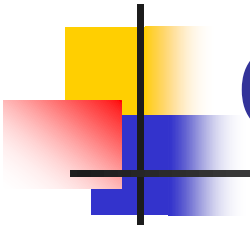


FIGURE 5.11 Combatting internal fragmentation by putting the unused part of the deleted slot back on the avail list.

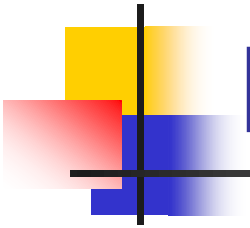
Combatendo a fragmentação interna, colocando partes não-utilizadas do espaço deletado de volta na lista



Compactação

- **Compactação:** consiste na busca por regiões do arquivo que não contém dados e posterior recuperação desses espaços perdidos

- Os espaços vazios são provocados, por exemplo, pela eliminação de registros



Eliminação de registros

- Deve existir um mecanismo que permita reconhecer quando uma área corresponde a um registro que foi eliminado
- Geralmente, isso é feito colocando um marcador especial no lugar do registro apagado
- Quando o procedimento de compactação é ativado, o espaço de todos os registros marcados é recuperado de uma só vez
- Se existe espaço suficiente, a maneira mais simples de compactar é executando um programa de cópia de arquivos que "pule" os registros apagados

Processo de compactação

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|.................
```

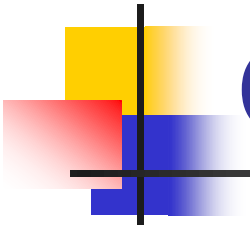
(a)

```
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)

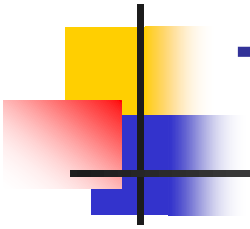
```
Ames|John|123 Maple|Stillwater|OK|74075|.....................
Brown|Martha|625 Kimbark|Des Moines|IA|50311|.................
```

(c)



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
- Algumas técnicas são gerais, outras específicas para certos tipos de dados, como voz, imagem ou texto
- Técnicas reversíveis *vs.* irreversíveis
- A variedade de técnicas é enorme



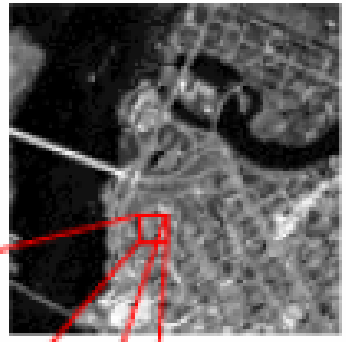
Técnicas

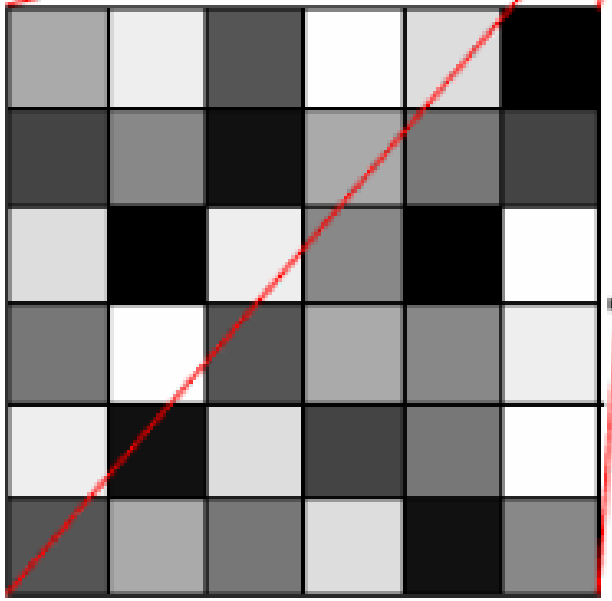
- **Redução de Redundância**
- **Omissão de sequências repetidas**
- **Códigos de tamanho variável: Código de Huffman**

Redução de Redundância

- Exemplo:
  - Códigos de estado, armazenados na forma de texto: 2 bytes
  - Mas como existem 50 estados (nos EUA!), pode-se armazenar os estados em 6 bits
  - Pode-se, então, guardar a informação em 1 byte e economizar 50% do espaço
  - Desvantagens?
    - Legibilidade, codificação/decodificação...

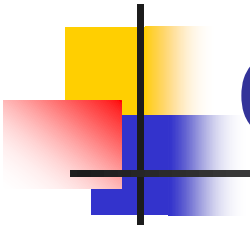
Omissão de sequências repetidas



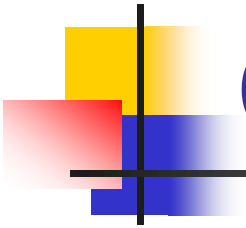


170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Omissão de sequências repetidas

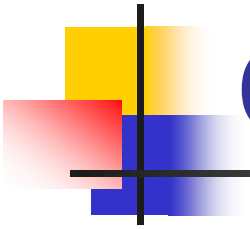


- Para a sequência
  - 22 23 24 24 24 24 24 24 24 25 26 26 26 26 26 26 26 25 24
- Usando 0xff como código de *run-length*
  - 22 23 ff 24 07 25 ff 26 06 25 24
- Garante redução de espaço sempre?



Código de Huffman

- No código ASCII: 1 byte por caracter (fixo)
  - ‘A’ = 65 (8 bits)
  - Cadeia ‘ABC’ ocupa 3 bytes
- Huffman: exemplo de código de tamanho variável
  - Ideia: valores mais frequentes são associados a códigos menores



Código de Huffman

- Se letras que ocorrem com frequência têm códigos menores, as cadeias tendem a ficar mais curtas...
- Requer informação sobre a frequência de ocorrência de cada símbolo a ser codificado
- Muito usado para codificar texto

Código de Huffman

- Exemplo

  Alfabeto: {A, B, C, D}

  Freqüência: A > B > C = D

  Possível codificação:

  A = 0, B = 1 1 0, C = 1 0, D = 1 1 1

  Cadeia: A B A C C D A

  Código: 0 1 1 0 0 1 0 1 1 1 0

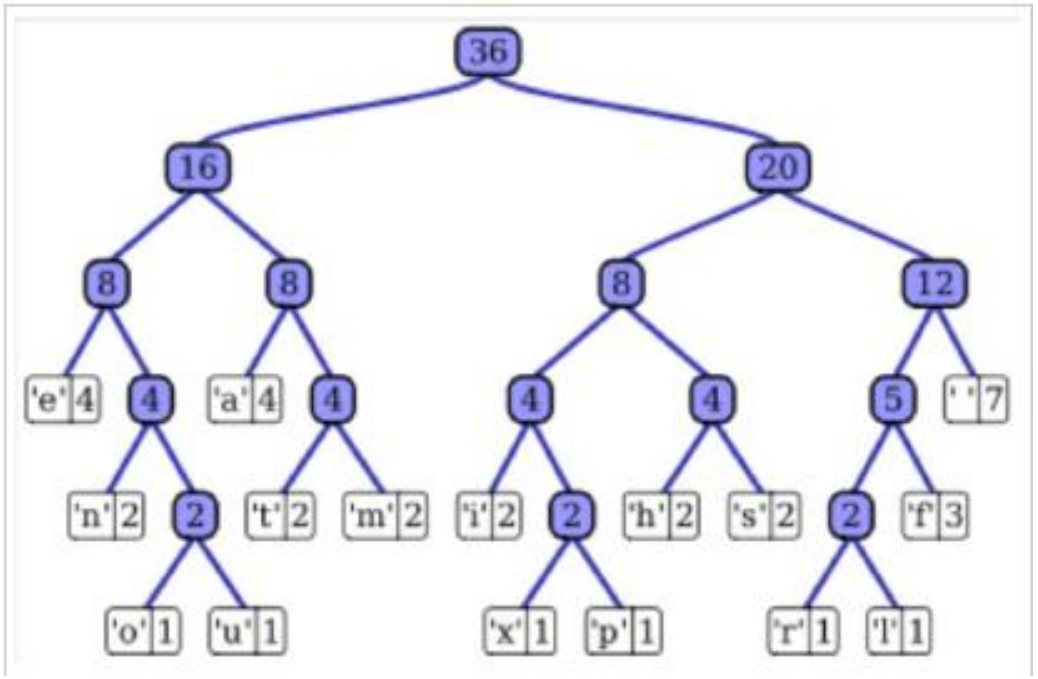
  Codificação não pode ser ambígua!

  Ex. A = 0, B = 0 1, C = 1

  A C B A = 0 1 0 1 0

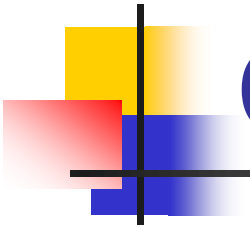
  É possível decodificar??

Código de Huffman Exemplo



"this is an example of a huffman tree".
Fonte: wikipedia

Carac	Freq	Cód
espaço	7	
a	4	
e	4	
f	3	
h	2	
i	2	
m	2	
n	2	
s	2	
t	2	
l	1	
o	1	
p	1	
r	1	
u	1	
x	1	



Código de Huffman

```
enquanto tamanho(alfabeto) > 1:
    S0 := retira_menor_probabilidade(alfabeto)
    S1 := retira_menor_probabilidade(alfabeto)
    X := novo_nó
    X.filho0 := S0
    X.filho1 := S1
    X.probabilidade := S0.probabilidade + S1.probabilidade
    insere(alfabeto, X)
fim enquanto
X = retira_menor_símbolo(alfabeto) # nesse ponto só existe um símbolo.
para cada folha em folhas(X):
        código[folha] := percorre_da_raiz_até_a_folha(folha)
fim para
```