GBC053 - Gerenciamento de Bancos de Dados

Aula 5
Organização de arquivos para desempenho
Compressão de dados e
recuperação de espaço em arquivos

Humberto Razente
humberto.razente@ufu.br

Compressão de dados

- Métodos para tornar um arquivo menor
  - menor armazenamento → economia em HD
  - transmissão mais rápida → diminui o tempo de acesso, permite transmissão com rede com largura de banda menor (economia de rede)
  - processamento sequencial mais rápido

Uso de uma notação diferente

- Redundância
  - uso de uma notação mais compacta
  - exemplo: para armazenar o campo UF, 2 bytes são empregados (ASCII)
    - entretanto: Brasil: 26 estados + distrito federal
      - 1 byte é suficiente para codificar estados
      - economia de 50%
    - problemas:
      - torna arquivo ilegível quando aberto por editor de textos simples
      - custo computacional para codificação/decodificação (consultas na tabela)
      - aumenta complexidade dos programas

Supressão de sequências repetidas

- Considera um valor de byte especial
  - não presente no arquivo
  - indica um código *run-length*
- Exemplo: compressão de imagem com cada pixel representado por de 8 bits
  - ler pixels em sequencia, copiando os valores para o arquivo, exceto quando o valor do pixel se repete
  - nesse caso, substitua o valor dos pixels por:
    - o indicador do código run-length
    - o valor do pixel
    - o número de repetições (até 256)
- Não garante eficiência

Supressão de sequências repetidas

- Exemplo
  - como codificar a sequência

    22 23 24 24 24 24 24 24 24 25 26 26 26 26 26 26 25 24 ?
  - Se o identificador de run-length for o 0, resultaria em:

    22 23 0 24 7 25 0 26 6 25 24

Atribuição de códigos de tamanho variável

- Código Morse: dot (.) ou dash (-)
  - tabela que associa símbolos e valores
- Entretanto, a maioria dos dados não possui distribuição de frequência previsível
- O código de Huffman constrói uma árvore binária, onde um percurso representa o código para um valor
  - a idéia do método é atribuir códigos mais curtos a símbolos com frequências mais altas
  - um código único de tamanho variável é atribuído a cada símbolo diferente

Código de Huffman

- A compressão com o código de Huffman pode ser feita
  - caracter por caracter
    - texto: compressão para 60% do tamanho original
  - palavra por palavra
    - texto: compressão para 25% do tamanho original

Código de Huffman

- Exemplo:
  - ”para cada rosa rosa, uma rosa é uma rosa”
  - símbolos: { ”para” , ”cada” , ”rosa” , ”,_” , ”uma” , ”é” }
  - frequências: 1, 1, 4, 1, 2, 1, respectivamente
- Huffman: abordagem que constrói uma árvore de codificação partindo-se de baixo para cima
  - início: n folhas → palavras do vocabulário + frequências
  - a cada iteração, as duas árvores com menores frequências são combinadas e a soma das frequências é associada à raíz

Huffman: ”para cada rosa rosa, uma rosa é uma rosa”

a)

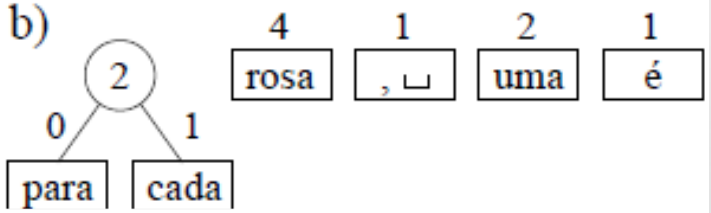
1	1	4	1	2	1
para	cada	rosa	, ␣	uma	é

Código de Huffman

a)

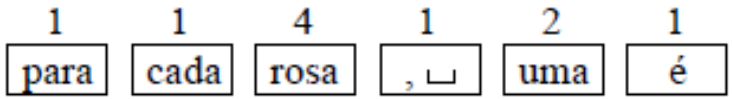
1	1	4	1	2	1
para	cada	rosa	, ␣	uma	é

b)

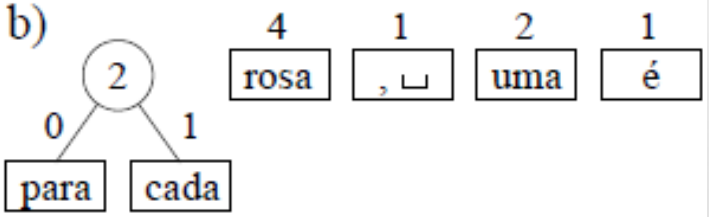


Código de Huffman

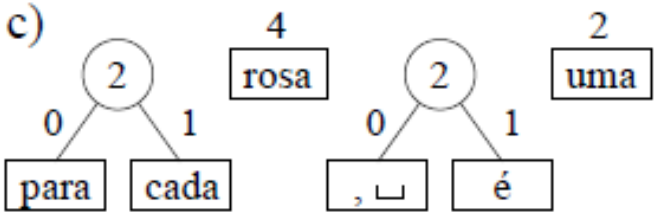
a)



b)

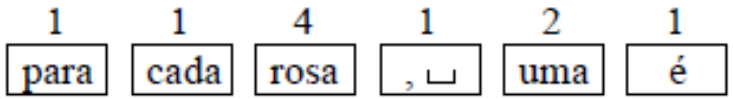


c)

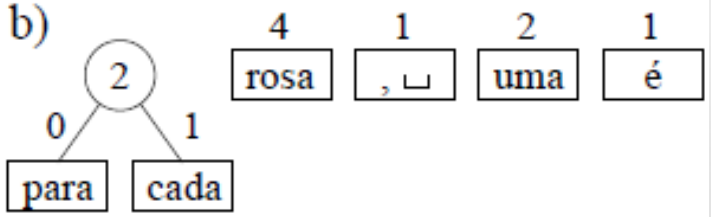


Código de Huffman

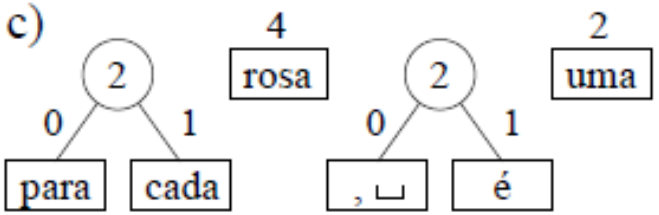
a)



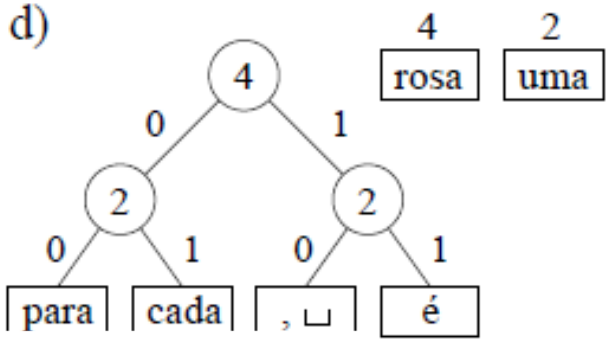
b)



c)

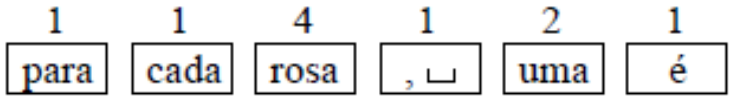


d)

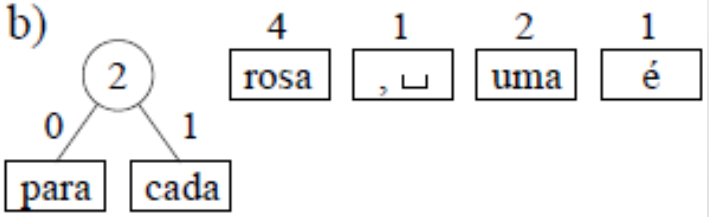


Código de Huffman

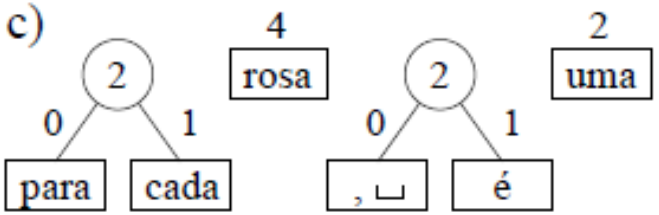
a)



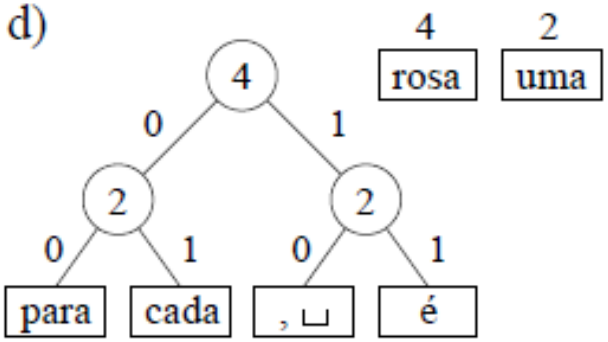
b)



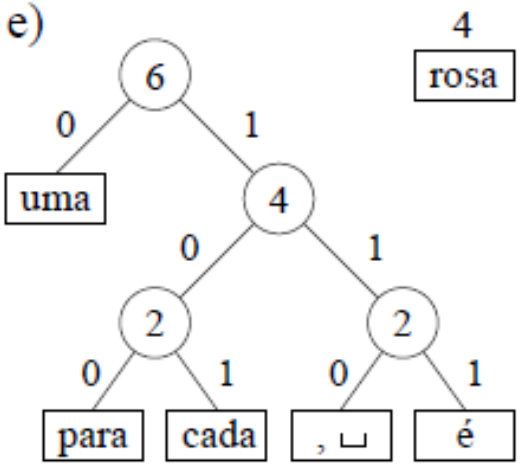
c)



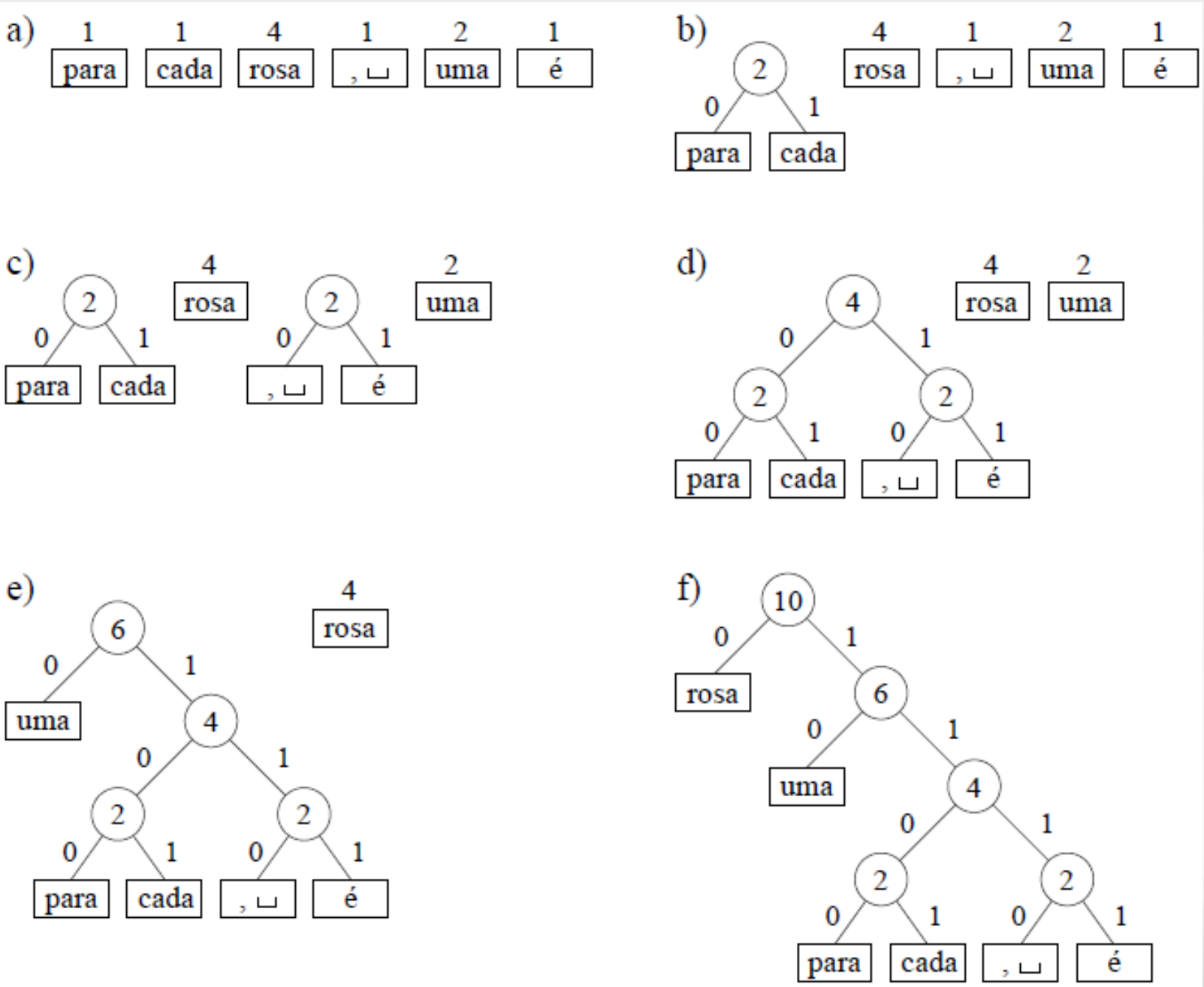
d)



e)



Código de Huffman



Dicionário
0 → rosa
10 → uma
1100 → para
1101 → cada
1110 → ,␣
1111 → é

Código de Huffman

- Código gerado de tamanho variável
- Sem ambiguidades → código canônico
- Não precisa de separadores
  - ”para cada rosa rosa, uma rosa é uma rosa”
  - 11001101001110100111100

Dicionário
0 → rosa
10 → uma
1100 → para
1101 → cada
1110 → ’_
1111 → é

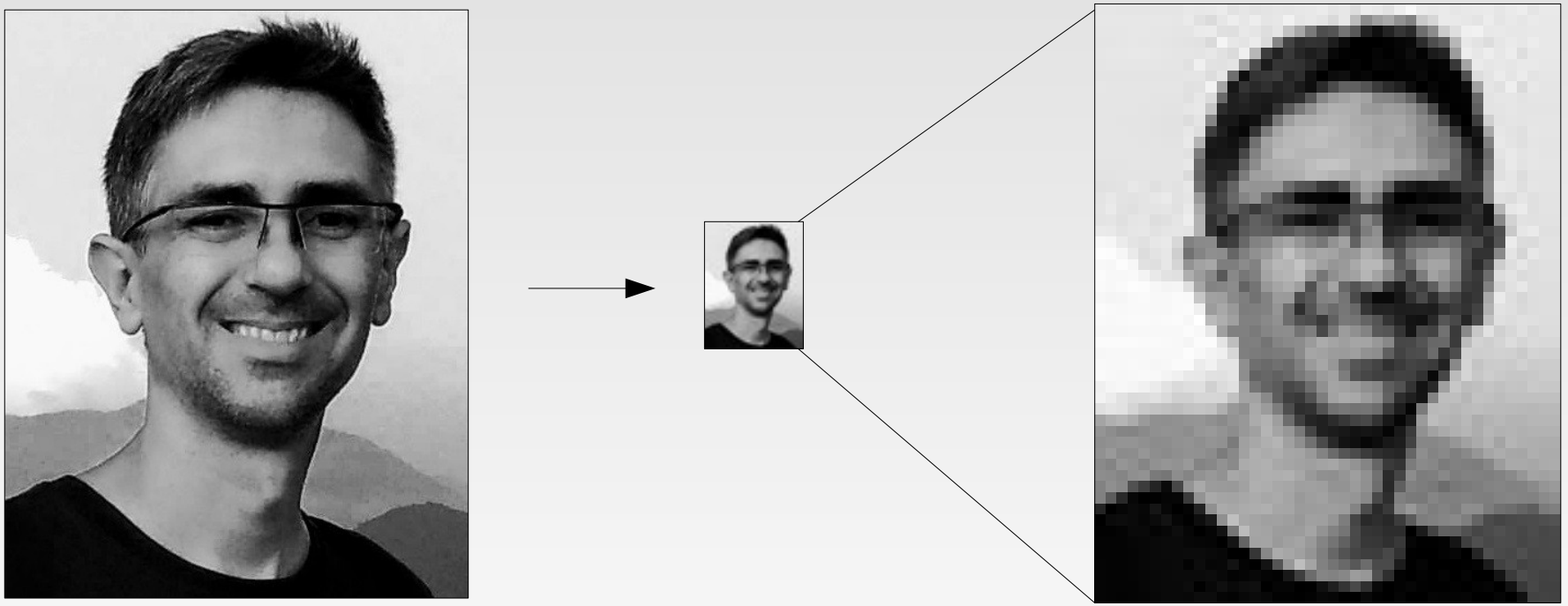
Técnicas de compressão irreversíveis

- As técnicas apresentadas anteriormente preservam todos os dados originais
  - é possível recuperar o arquivo original a partir do compactado
  - compactação se dá pelo fato de haver redundância nos dados
- Outro tipo de compressão, denominada irreversível, baseia-se na suposição de que parte dos dados podem ser sacrificados (perdidos)

Técnicas de compressão irreversíveis

- Exemplo: comprimir uma imagem de 400x400 pixels para uma imagem de 100x100 pixels
  - cada 4x4 pixels (16 pixels) será representado por 1 pixel
  - imagem de 100x100 pixels é muito semelhante a imagem de 400x400 pixels
  - entretanto, a partir dela, não é possível obter-se a mesma imagem de 400x400 pixels

Técnicas de compressão irreversíveis



Curiosidade: esteganografia

- Técnica que permite esconder uma mensagem dentro de outra mensagem
  - Exemplo: considere uma imagem como uma matriz de pixels de alta resolução de cores (24 bits por pixel)
  - É possivel codificar uma mensagem de texto, por exemplo, de modo que cada bit do texto seja incluído em 1 dos 24 bits de cada pixel, sem causar grandes diferenças na visualização da imagem para os olhos de uma pessoa?

Curiosidade: esteganografia

- A. Cheddad, J. Condella, K. Currana, P. McKevitt (2010). ”Digital image steganography: Survey and analysis of current methods”, Signal Processing, 90(3):727-752
- http://dx.doi.org/10.1016/j.sigpro.2009.08.010

Recuperação de Espaço em Arquivos

- Supondo que um registro de tamanho variável é alterado e resulta em um número de bytes maior
  - o que fazer com os dados extra?
  - reorganizar o arquivo todo para o registro caber na posição?
  - adicionar no final do arquivo e colocar um ponteiro para no registro original?
  - incluir o registro todo no final do arquivo e deixar um buraco onde estava antes?

Recuperação de Espaço em Arquivos

- A deterioração de um arquivo ocorre quando há
  - alteração de registros
  - remoção de registros

- Principalmente em:
  - alteração de registros de tamanho variável
  - remoção de registros de tamanho fixo ou variável

Recuperação de Espaço em Arquivos

- Soluções:
  - remoção: marcar registro como removido
    - aplicação deve estar preparada para apresentar apenas registros não marcados como removidos
    - permite a recuperação desses registros
    - processamento pode ser executado para compactar o arquivo → reescrevê-lo sem a inclusão dos registros marcados como removidos

Recuperação de Espaço em Arquivos

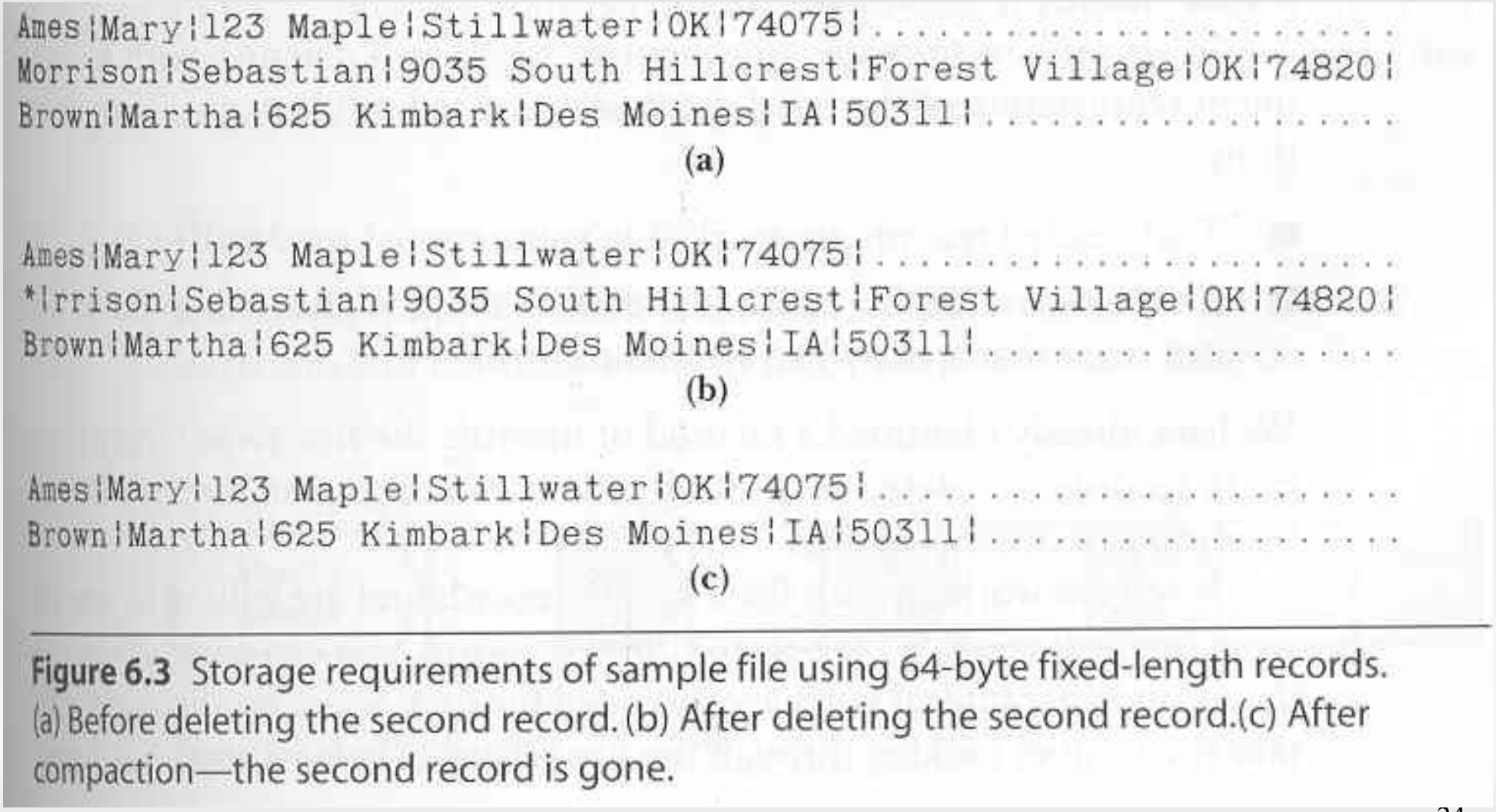


Figure 6.3 Storage requirements of sample file using 64-byte fixed-length records. (a) Before deleting the second record. (b) After deleting the second record.(c) After compaction—the second record is gone.

Reuso dinâmico

- Há aplicações em que há a necessidade de reuso imediato do espaço ocupado por registros removidos
  - marcação dos registros removidos
  - inclusão de novos registros no local de registros removidos
    - acesso sequencial para procurar por espaços a cada inserção é caro
    - solução é manter uma lista de elementos removidos, inclusive dos seus tamanhos

Reuso dinâmico

- Como?
  - se tamanho variável → número do registro não tem muita utilidade para a otimização
    - solução: *byte offset*
  - uso de uma pilha ou fila, usando o espaço dos próprios registros do arquivo
    - no header do arquivo, guarda-se o *byte offset* para o topo da pilha de registros marcados como excluídos
    - no registro excluído, coloca-se além da marcação da exclusão, o *byte offset* do próximo excluído
      - -1 pode ser usado para indicar último elemento da pilha

Reuso dinâmico

List head (first available record) → 5

0	1	2	3	4	5	6
Edwards . . .	Bates . . .	Wills . . .	*−1	Masters . . .	*3	Chavez . . .

(a)

List head (first available record) → 1

0	1	2	3	4	5	6
Edwards . . .	*5	Wills . . .	*−1	Masters . . .	*3	Chavez . . .

(b)

List head (first available record) → −1

0	1	2	3	4	5	6
Edwards . . .	*1st new rec . . .*	Wills . . .	*3rd new rec . . .*	Masters . . .	*2nd new rec . . .*	Chavez . . .

(c)

Figure 6.5 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.

Fragmentação interna

- Considere registros de tamanho fixo
  - os pontos no final do registro representam espaços em branco
    - desperdício de espaço
    - é parte do custo de se usar registros de tamnho fixo

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

Figure 6.8 Storage requirements of sample file using 64-byte fixed-length records.

```
40 Ames¦Mary¦123 Maple¦Stillwater¦OK¦74075¦64 Morrison¦Sebastian
¦9035 South Hillcrest¦Forest Village¦OK¦74820¦45 Brown¦Martha¦62
5 Kimbark¦Des Moines¦IA¦50311¦
```

Figure 6.9 Storage requirements of sample file using variable-length records with a count field.

Fragmentação interna

- Pode-se concluir que registros de tamanho variável resolvem o problema do desperdício
  - mas o que acontece após atualizações/remoções?
- Se um registro de tamanho menor substituir um registro maior também ocorre fragmentação interna
  - solução: quebrar registro em duas partes
    - uma para o novo registro
    - restante de volta para a lista de disponibilidade

Fragmentação externa

- Ainda assim, na realocação de um outro registro nesse espaço, pode haver uma sobra que está tão fragmentada que não suporta um registro
  - fragmentação externa
- Soluções
  - compactação: reescrever o arquivo
  - estratégias de alocação

Fragmentação externa – estratégias de alocação

- First-fit
  - procura-se na lista de disponíveis até encontrar um espaço que caiba o registro
    - essa estratégia não se preocupa se o espaço disponível é do tamanho do registro ou muito maior
- Best-fit
  - mantém-se a lista de disponíveis ordenada em ordem ascendente de espaço
    - permite percorrer a pilha até encontrar o menor espaço para comportar o registro
    - também gera fragmentação externa
    - alto custo para manter a lista ordenada a cada inserção

Fragmentação externa – estratégias de alocação

- Worst-fit
  - mantém-se a lista de disponíveis ordenada em ordem descendente de espaço
    - insere o registro no maior espaço disponível (o primeiro da lista)
    - reinsere espaço remanescente na lista ordenada
    - também gera fragmentação externa
    - alto custo para manter a lista ordenada a cada inserção

Leitura complementar

- Capítulo **”Organizing Files for Performance”** do livro
  - Folk et al. ”File Structures: An Object-Oriented Approach with C++”, Editora Pearson, 3ª edição, 1998