Arquivos: 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

- 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

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

Registro Cabeçalho (*header record*)

- Algumas <u>informações típicas</u>
  - Número de registros
  - Tamanho de cada registro
  - Nomes dos campos de cada registro
  - Tamanho dos campos
  - Datas de criação e atualização

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

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

Metadados

- 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"

Suponha que:

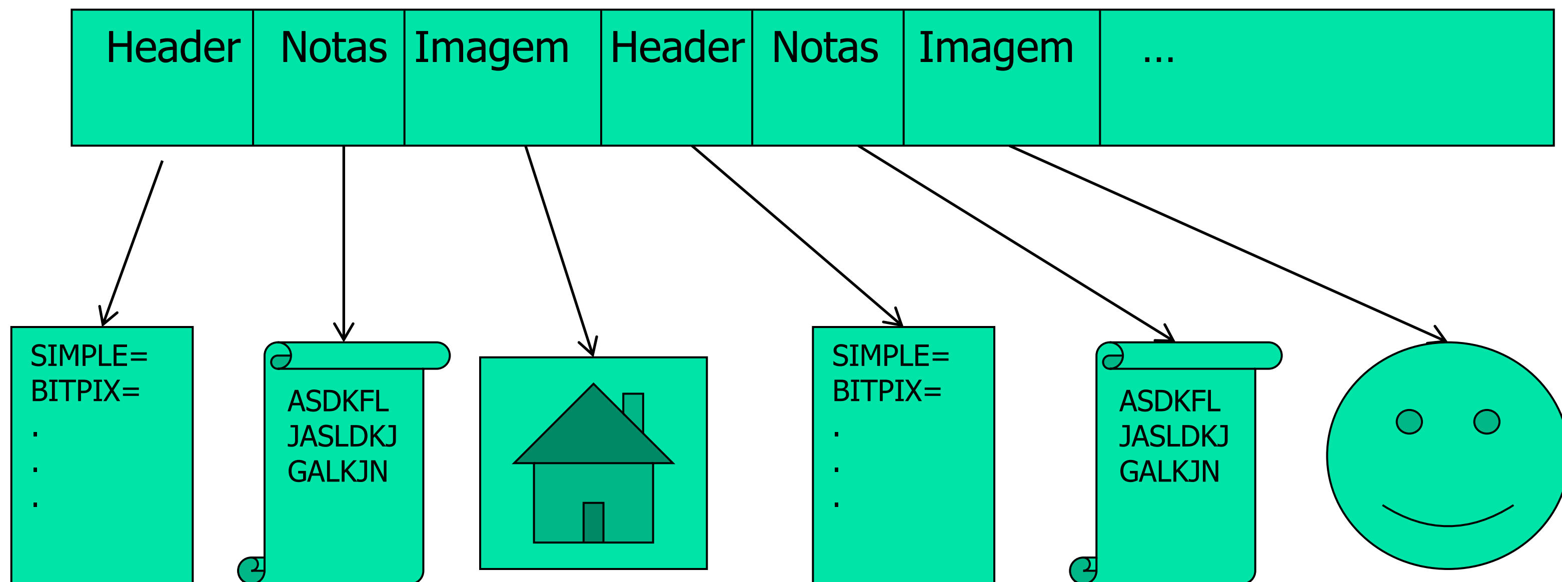
- O astrônomo resolvesse associar um documento (notas) com suas observações sobre cada imagem.
- Assim, haveria 3 registros de tamanhos variáveis (header, notas, imagem) associados ao objeto.
- Generaliza-se a noção de keywords para um arquivo de objetos mistos

Uso de Tags

- Cada registro de tamanho variável passa a ser indexado por uma *tag*
- No caso: header, notas, image
- Nos registros das tags, guardar o nome da keyword, o deslocamento (byte offset) no arquivo daquela informação, e a indicação do seu tamanho em bytes

Um arquivo “tagged”

Tabela de Índices com Tags



Arquivo com dados primários

➔ Métodos de leitura/escrita para cada tipo de tag

Exemplos de Formatos com estrutura Tag

- TIFF – Tagged Image File Format
- HDF – Hierarchical Data Format
- SGML – Standard General Markup Language
  - Linguagem para descrever estruturas de documentos e para definir tags usadas nas estruturas (HTML, XML, etc.)

Portabilidade e Padronização

- Formas de codificação de arquivos devem ser “estar de acordo” com a visão de outras pessoas, softwares e computadores
- Fatores que afetam portabilidade
  - Diferenças entre sistemas operacionais
  - Diferenças entre linguagens de programação
  - Diferenças entre arquiteturas de computadores
  - Etc.
- Muitas vezes são necessários conversores de formatos

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

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

Técnicas

- Notação diferenciada
  - Redução de redundância
- Omissão de sequências repetidas
  - Redução de redundância
- Códigos 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

Omissão de sequências repetidas

Ex.: representação de imagens
- O fundo a imagem resulta em códigos repetidos

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

Omissão de sequências repetidas

- 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

- Código ASCII: 1 byte por caracter (fixo)
  - 'A' = 65 (8 bits)
  - Cadeia 'ABC' ocupa 3 bytes
  - Ignora frequência dos caracteres

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

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

Código de Huffman

- Muito usado para codificar texto
- Como estabelecer a relação entre frequência e código?

Código de Huffman

- Exemplo

  Alfabeto: {A, B, C, D}

  Frequência: A > B > C = D

  Possível codificação:

  A=0, B=110, C=10, D=111

  Cadeia: ABACCDA

  Código: 0110010101110

  É possível decodificar?

  Codificação deve ser não ambígua

  Ex. A=0, B=01, C=1

  ACBA → 01010

  É possível decodificar?

Código de Huffman

- Cada "prefixo" de um código identifica as possibilidades de codificação
  - Se primeiro bit é 0, então A; se 1, então B, C ou D, dependendo do próximo bit
  - Se segundo bit é 0, então C; se 1, então B ou D, dependendo do próximo bit
  - Se terceiro bit é 0, então B; se 1, então D
  - Quando o símbolo é determinado, começa-se novamente a partir do próximo bit

Símbolo	Código
A	0
B	110
C	10
D	111

Código de Huffman

- Como representar todas as mensagens possíveis de serem expressas em língua portuguesa
  - Como determinar os códigos de cada letra/sílaba/palavra?
    - Qual a unidade do “alfabeto”?

- Código de Huffman
  - Calcula-se a frequência de cada símbolo do alfabeto
    1. Encontre os dois símbolos que têm menor frequência (B/1 e D/1)
    2. O último símbolo de seus códigos deve diferenciá-los (B=0 e D=1)
    3. Combinam-se esses símbolos e somam-se suas frequências (BD/2, indicando ocorrência de B ou D)
    4. Repete-se o processo até restar um símbolo
       1. C/2 e BD/2 → 0 para C e 1 para BD → CBD/4
       2. A/3 e CBD/4 → 0 para A e 1 para CBD → ACBD/7

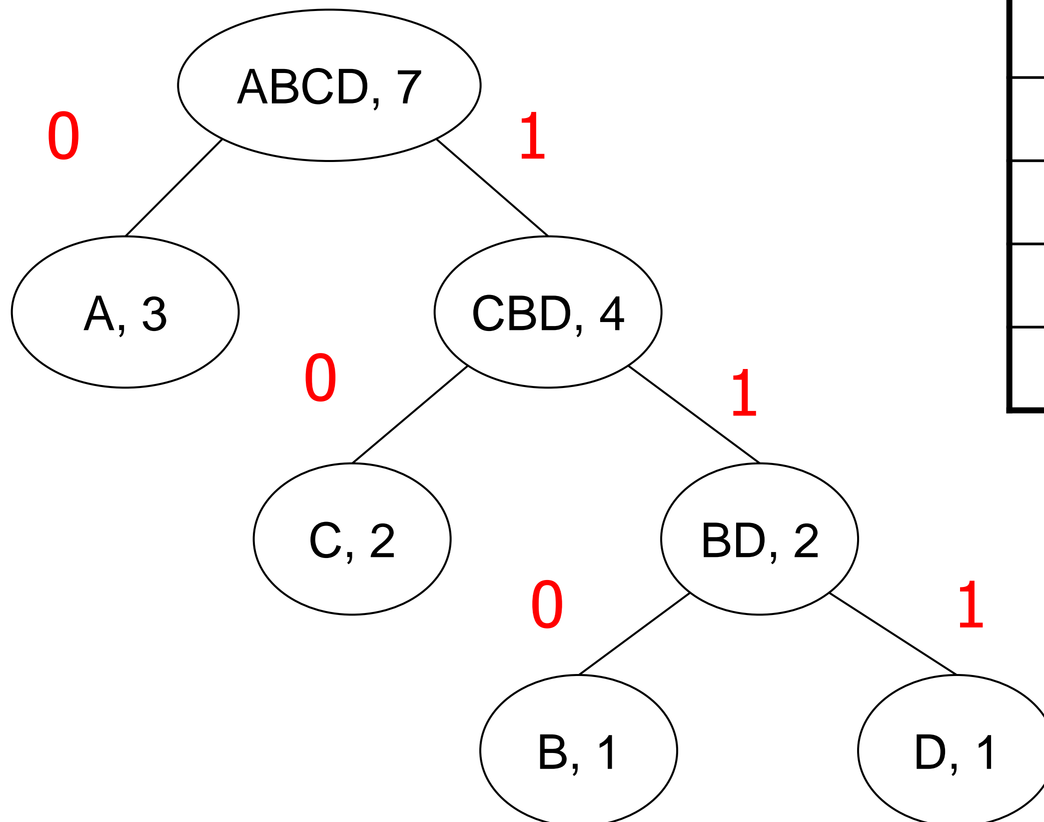
ABACCDA ⟶ Freq.: A/3, B/1, C/2 e D1

Código de Huffman

- Combinação de dois símbolos em 1
- Árvore de Huffman
  - Construída passo a passo após cada combinação de símbolos
  - Árvore binária
    - Cada nó da árvore representa um símbolo (e sua frequência)
    - Cada nó folha representa um símbolo do alfabeto original
  - Ao se percorrer a árvore da raiz até um símbolo (folha), o caminho fornece o código deste símbolo
    - Escalada por ramo esquerdo: 0 no início do código
    - Escalada por ramo direito: 1 no início do código

Código de Huffman

- Exemplo



Símbolo	Código
A	0
B	110
C	10
D	111

Símbolos mais frequentes mais à esquerda e mais próximos da raiz

Código de Huffman

- Exercício: construir a árvore para os dados abaixo

Símbolo	Freqüência
A	15
B	6
C	7
D	12
E	25
F	4
G	6
H	1
I	15

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
- Onde se usa isso?

Manipulação de dados em arquivos

- Operações básicas que podemos fazer com os dados nos arquivos?

Manipulação de dados em arquivos

- Operações básicas que 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?

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

Eliminação de registros

- Devem existir mecanismos que
  1. Permitam reconhecer áreas que foram apagadas
  2. Permitam recuperar e utilizar os espaços vazios

- Possibilidades?
  - Discussão em grupos de 2 alunos

Eliminação de registros

- Geralmente, áreas apagadas são marcadas com um marcador especial

- Quando o procedimento de compactação é ativado, 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)

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

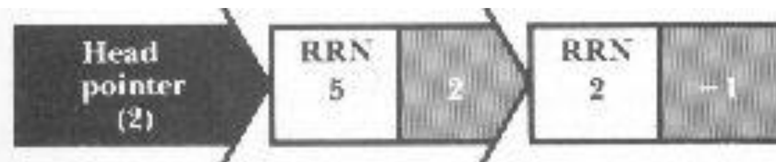
- Muitas vezes, o procedimento de compactação é esporádico
  - Um registro apagado não fica disponível para uso imediatamente
- Em aplicações interativas que acessam arquivos altamente voláteis, pode ser necessário um processo dinâmico de recuperação de espaços vazios
  - Marcar registros apagados
 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

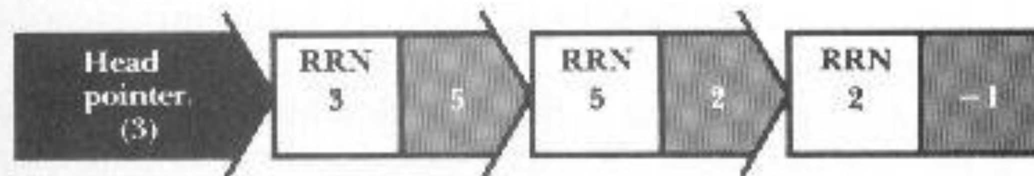
Como localizar os espaços vazios?

- Registros de tamanho fixo
  - Lista encadeada de registros eliminados no próprio arquivo
    - 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

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 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.

Registros de tamanho fixo

- Por que se usa uma pilha e não uma fila ou outra estrutura de dados?

- A pilha poderia ser mantida na memória principal?
  - Vantagens?
  - Desvantagens?

Registros de tamanho variável

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

Eliminação de registros

```
        HEAD.FIRST_AVAIL: -1

40 Ames|John|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|
```

(a)

```
        HEAD.FIRST_AVAIL: 43

40 Ames|John|123 Maple|Stillwater|OK|74075|64 *| -1..............
.......................................................45 Brown|Martha|62
5 Kimbark|Des Moines|IA 50311|
```

(b)

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).

Registros de tamanho variável

- Para recuperar registros, não é possível usar uma pilha
  - É necessário uma busca seqüencial na lista para encontrar uma posição com espaço suficiente

Adição de um registro de 55 bytes: exemplo

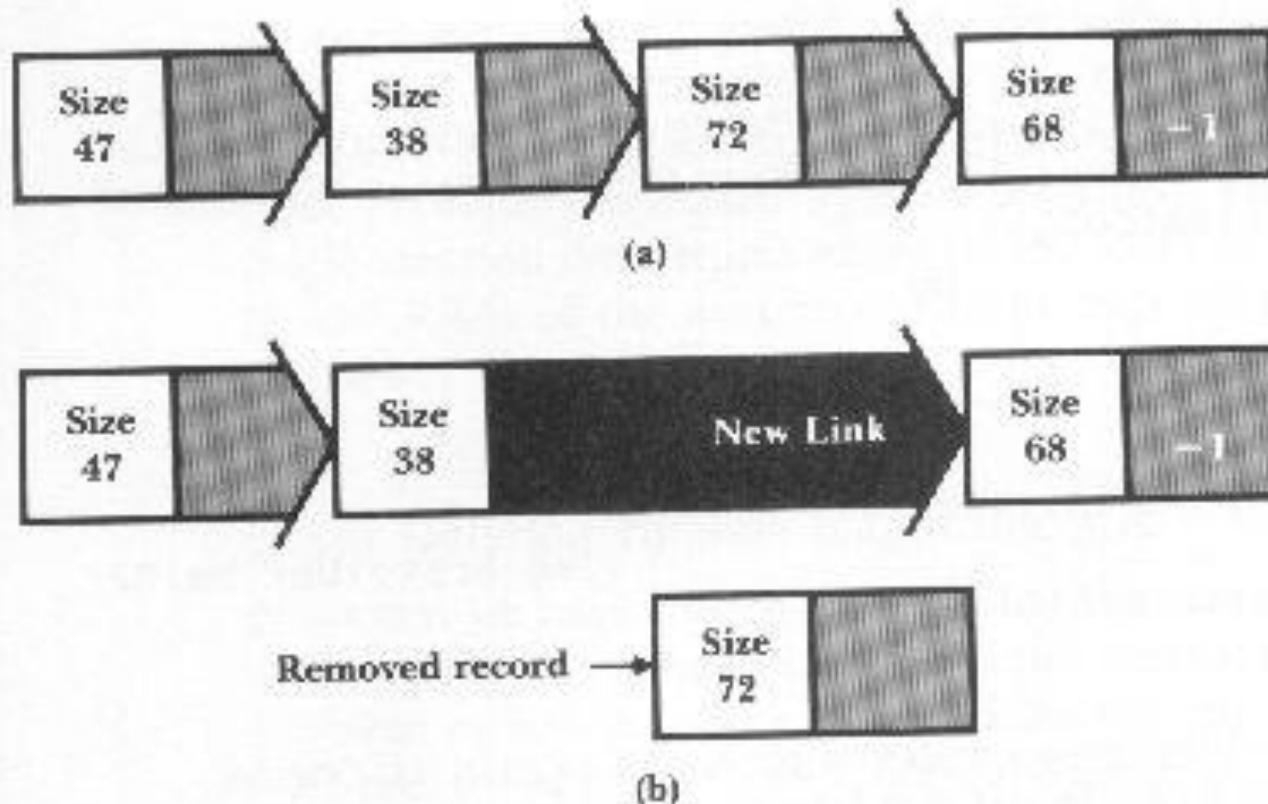


FIGURE 5.7 Removal of a record from an avail list with variable-length records. (a) Before removal. (b) After removal.

Registros de tamanho variável

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

Registros de tamanho variável

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

Fragmentação interna

FIGURE 5.10 Illustration of fragmentation with variable-length records. (a) After deletion of the second record (unused characters in the deleted record are replaced by periods). (b) After the subsequent addition of the record for Al Ham.

```
        HEAD.FIRST_AVAIL: 43

40 Ames|John|123 Maple|Stillwater|OK|74075|64 *| -1.............
...................................................45 Brown|Martha|62 .
5 Kimbark|Des Moines|IA|50311|
```

(a)

```
        HEAD.FIRST_AVAIL: -1

40 Ames|John|123 Maple|Stillwater|OK|74075|64 Ham|Al|28 Elm|Ada|
OK|70332.......................................45 Brown|Martha|62
5 Kimbark|Des Moines|IA|50311|
```

(b)

Registros de tamanho variável

- Estratégias de alocação de espaço
  - *First-fit*: pega-se o primeiro que servir, como feito anteriormente
    - Soluções?

Registros de tamanho variável

- Estratégias de alocação de espaço
  - *First-fit*: pega-se o primeiro que servir, como feito anteriormente
    - Soluções?
      1. Colocar o espaço que sobrou na lista de espaços disponíveis
      2. Escolher o espaço mais justo possível

Combatendo a fragmentação

- Solução: colocar o espaço que sobrou na lista de espaços disponíveis
  - Parece uma boa estratégia, independentemente da forma que se escolhe o espaço

```
           HEAD.FIRST_AVAIL: 43

40 Ames|John|123 Maple|Stillwater|OK|74075|35 *   -1..............
......................26 Ham|Al|28 Elm|Ada|OK|70332|45 Brown|Martha|6
25 Kimbark|Des Moines|IA|50311|
```

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

Registros de tamanho variável

- Solução: escolher o espaço mais justo possível
  - *Best-fit*: pega-se o mais justo
    - Desvantagem?

Registros de tamanho variável

- Solução: 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

Registros de tamanho variável

- Solução: escolher o espaço mais justo possível
  - *Best-fit*: pega-se o mais justo
    - É conveniente organizar a lista de forma ascendente?

Registros de tamanho variável

- Solução: escolher o maior espaço possível

  - *Worst-fit*: pega-se o maior

    - Diminui a fragmentação externa

    - Lista organizada de forma descendente?
      - O processamento pode ser mais simples

Registros de tamanho variável

- Outra forma de combater fragmentação externa
  - Junção de espaços vazios adjacentes
    - Qual a dificuldade desta abordagem?

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*

Ordenação e busca em arquivos

- É relativamente fácil buscar elementos em conjuntos ordenados

- A ordenação pode ajudar a diminuir o número de acessos a disco

Ordenação e busca em arquivos

- Já vimos busca seqüencial
  - O(n) → Muito ruim para acesso a disco!

- E a busca binária?
  - Modo de funcionamento?
  - Complexidade de tempo?

Busca binária

- Dificuldade: ordenar os dados em arquivo para se fazer a busca binária

- Alternativa: ordenar os dados em RAM
  - Ainda é necessário: ler todo o arquivo e ter memória interna disponível

Busca binária

- Limitações
  - Registros de tamanho fixo
  - Manter um arquivo ordenado é muito caro
  - Requer mais do que 1 ou 2 acessos
    - Por exemplo, em um arquivo com 1.000 registros, são necessários aproximadamente 10 acessos em média → ainda é ruim!

Busca binária

- Exercício
  - Implementar em C uma sub-rotina de busca binária em um arquivo ordenado por número USP

```
struct aluno {
        char nome[50];
        int nro_USP;
}
```

Keysorting

- Ordenação por chaves

- Idéia 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 ou byte inicial)
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

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 seqüencial
    - Alterna-se leitura no arquivo antigo e escrita no arquivo novo

Keysorting

- Questões

  - Por que criar um novo arquivo?

  - Não vale a pena usar o vetor ordenado como um índice?

Questão delicada

- Independentemente do método de ordenação
  - O que fazer com os espaços vazios originados de registros eliminados?
    - E a estrutura de dados que os mantêm para que sejam reutilizados?
      - *Pinned records*