

Manutenção de Arquivos

Algoritmos e Estruturas de Dados II

Prof. Ricardo J. G. B. Campello

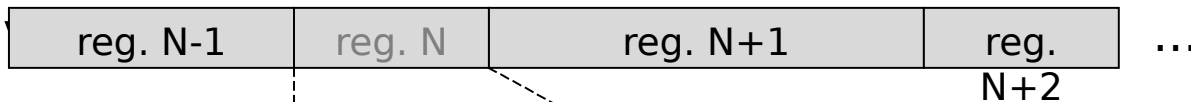


Manutenção de Arquivos

- Projetista deve considerar modificações no arquivo
 - **Adição, atualização e eliminação** de registros
- Problema é simples quando:
 - Registros são de tamanho fixo, **E**
 - Apenas adições e atualizações ocorrem
- Porém, em outras circunstâncias...

Manutenção de Arquivos

- P. ex., **atualizar** um registro que aumente de tamanho:
 - O que fazer com os dados adicionais?
 - Anexar ao final do arquivo e ligar as duas partes por “ponteiros” ?
 - Processamento de cada registro (logo do arq. todo) fica muito mais complexo
 - Apagar o registro original e reescrevê-lo todo no final do arquivo
 - Ok, mas temos que nos preocupar em reutilizar o espaço desocupado





Manutenção de Arquivos

- P. ex., **deletar** um registro (tamanho fixo ou variável):
 - O que fazer com o espaço remanescente?
 - Nesse caso também temos que nos preocupar em reutilizar o espaço vago
- Note que o foco de manutenção de arquivos pode se dar no problema de reaproveitamento de espaços vagos
- De fato, uma atualização sempre pode ser vista como:
 - Atualização = Eliminação + Adição



Manutenção de Arquivos

- Se o arquivo está off-line e sujeito a modificações esporádicas, e.g. lista de mala direta, espaços podem ser recuperados em **modo batelada (batch)**
 - Trata-se de apenas “marcar” os registros no momento da eliminação, e periodicamente eliminá-los todos de uma vez
 - Demanda um mecanismo que permita reconhecer quando uma área do arquivo corresponde a um registro que foi eliminado
 - Geralmente, isso é feito colocando um marcador especial no lugar do registro apagado (e.g. "*" nos primeiros 2 bytes do registro)
 - Após um certo no. de eliminações:
 - aciona-se um procedimento de **compactação**



Compactação

- 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 via **cópia seqüencial**:
 - novo arquivo é gerado copiando o original, porém ignorando os bytes correspondentes a registros eliminados
- Existem mecanismos de compactação **in-place**
 - mais complexos e computacionalmente pesados



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|.....  
*Morrison|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

- Procedimento de compactação é esporádico...
 - espaço não fica disponível imediatamente
- Em aplicações on-line, que acessam arquivos altamente voláteis, pode ser necessário um processo dinâmico de recuperação de espaço
 - marcar registros eliminados
 - localizar os espaços desocupados quando necessário
 - **sem buscas exaustivas !**



Recuperação Dinâmica

- Ao adicionar um novo registro, queremos:
 - Saber imediatamente se existem **slots**
 - slot = espaço disponível de um registro eliminado
 - Acessar **diretamente** um slot, se existir
 - diretamente = sem buscas exaustivas !

Registros de Tamanho Fixo

- Lista encadeada de registros eliminados disponíveis
- Cada elemento da lista armazena:
 - O RRN do próximo registro vago
- Cabeça da lista está no **header record** do arquivo:

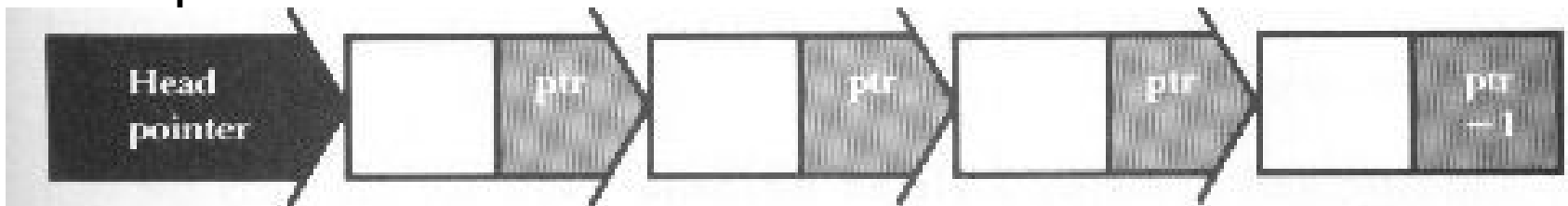
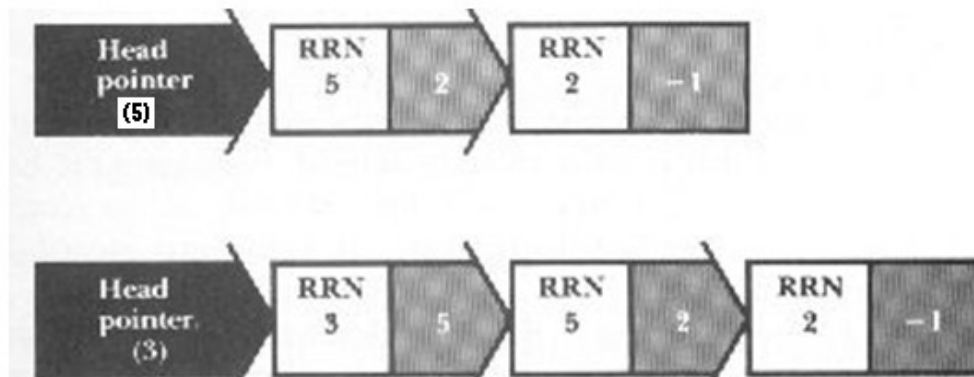


FIGURE 5.4 A linked list.

Registros de Tamanho Fixo

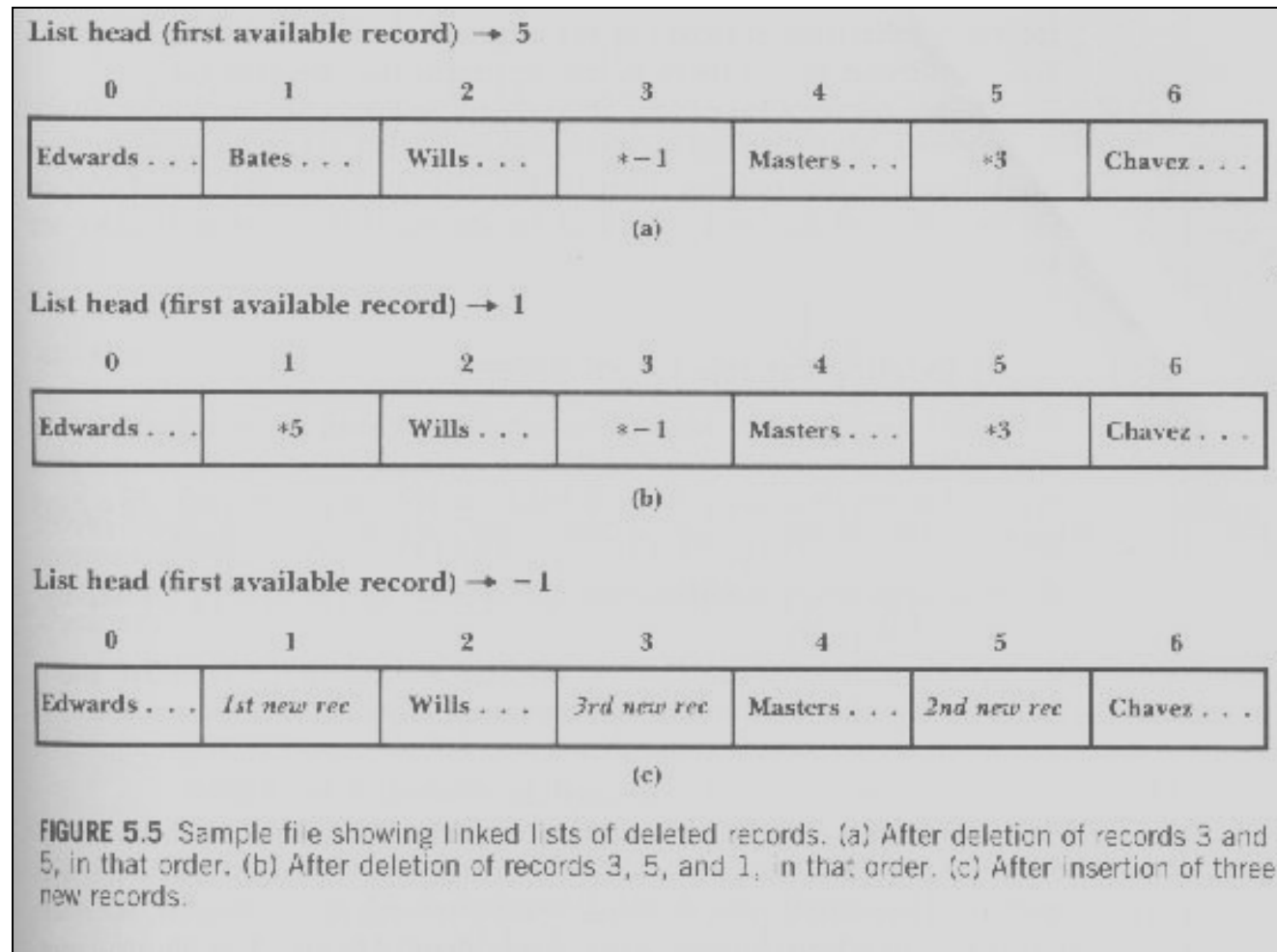
- Inserção e remoção ocorrem sempre no início da lista
 - Lista encadeada operada como Pilha !
 - Pilha pode ser mantida no próprio arquivo !



- Pilha antes e depois da inserção do registro com RRN 3
 - inserção na pilha \Leftrightarrow registro eliminado do arquivo
 - remoção da pilha \Leftrightarrow registro adicionado ao arquivo

Registros de Tamanho Fixo

- Exemplo



Registros de Tamanho Fixo

Exemplo (registros com 55 bytes)

Arquivo Original



Após remoção do

3º registro



Após remoção do

1º registro

```
head = -1
```

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

```
head = 2
```

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

```
head = 0
```

```
*|2|a|Rua 1|123|São Carlos|.....  
João|Rua A|255|Rio Claro|.....  
*|-1|Rua 10|56|Rib. Preto|.....
```



Registros de Tamanho Fixo

- Para fins de implementação prática, o cabeçalho pode ser implementado como uma **struct** em **C**:
 - um dos campos armazena o RRN do 1º reg. vago
 - p. ex. `int head.first_avail`
 - demais campos podem armazenar outras infos
- O arquivo em si começa após os bytes do cabeçalho



Registros de Tamanho Fixo

- Pseudo-Código (Eliminação Registro)

```
FUNCTION:  del_rec(RRN)
```

```
    move file pointer to RRN position in file  
    write delete flag field ('*!') in current position in file  
    write HEAD.FIRST_AVAIL in new current position in file  
    set HEAD.FIRST_AVAIL to RRN
```

```
end FUNCTION
```

FIGURE 5.4 ■ Function *del_rec(RRN)* deletes record number *RRN*. The space opened by the deletion is pushed onto the avail list.



Registros de Tamanho Fixo

- Pseudo-Código (Localização de Slot)

```
FUNCTION: pop_avail()  
  
    if HEAD.FIRST_AVAIL == -1 then /* avail list empty */  
        return RRN of next record to be appended  
  
    else /* pop avail list */  
        set RET_VAL to HEAD.FIRST_AVAIL  
        move file pointer to HEAD.FIRST_AVAIL position in file  
        skip over '*' field  
        read link field from file into RRN  
        set HEAD.FIRST_AVAIL to RRN  
        return RET_VAL  
  
end FUNCTION
```

FIGURE 5.5 • Function *pop_avail()* returns the RRN of the first available slot in the file. If the avail list of deleted records is empty, the function returns the RRN of the next record to be appended at the end of the file.



Registros de Tamanho Variável

- No caso de registros de tamanho variável, temos um problema adicional...
 - Registros não são acessíveis por RRN...
 - Não mais se recuperam os byte offsets pelos RRNs
 - Não adianta encadear os RRNs dos registros vagos



Registros de Tamanho Variável

- Registros não são acessíveis por RRN...
- **Solução:**
 - Armazenar os byte offsets na lista encadeada
 - ao invés dos RRNs
 - Utilizar registros com indicador de tamanho
 - permite saber o tamanho do slot a partir do byte offset




Exemplo

HEAD.FIRST_AVAIL: -1

```
40 Ames|John|123 Maple|Stillwater|OK|74075|64 Morrison|Sebastian  
19035 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

- Mas o problema ainda não está solucionado...
 - Como os registros são de tamanho variável, não é qualquer slot da lista que serve para acomodar um novo registro a ser adicionado
 - é preciso **encontrar um slot grande o suficiente**
 - se não for encontrado, adiciona-se ao final do arquivo
 - para isso, é preciso **percorrer seqüencialmente a lista**

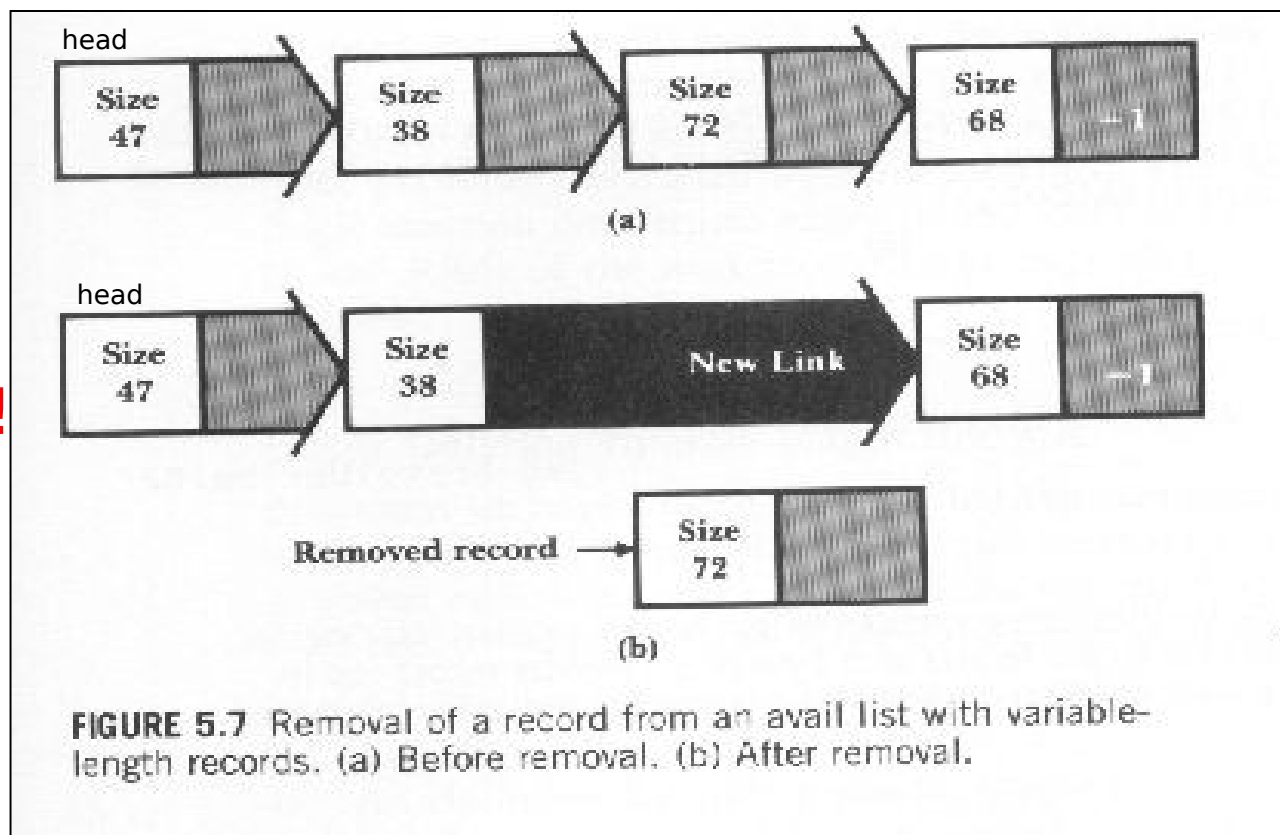
Registros de Tamanho Variável

- Exemplo 1: adicionar registro de 55 bytes

47 ? pequeno...

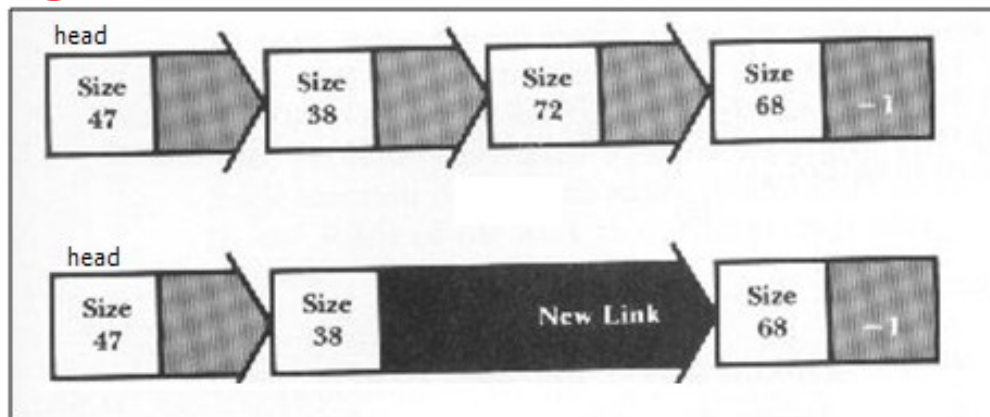
38 ? pequeno...

72 ? Suficiente !



Fragmentação Interna

- No Exemplo 1, usamos todos os 72 bytes de um slot para adicionar um registro de apenas 55 bytes
 - Os 17 bytes extras ficaram inutilizados
 - **fragmentação interna**



Fragmentação Interna

- Exemplo 2: adicionar Ham|A1|28 Elm|Ada|0K|70332| (27 bytes)

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)

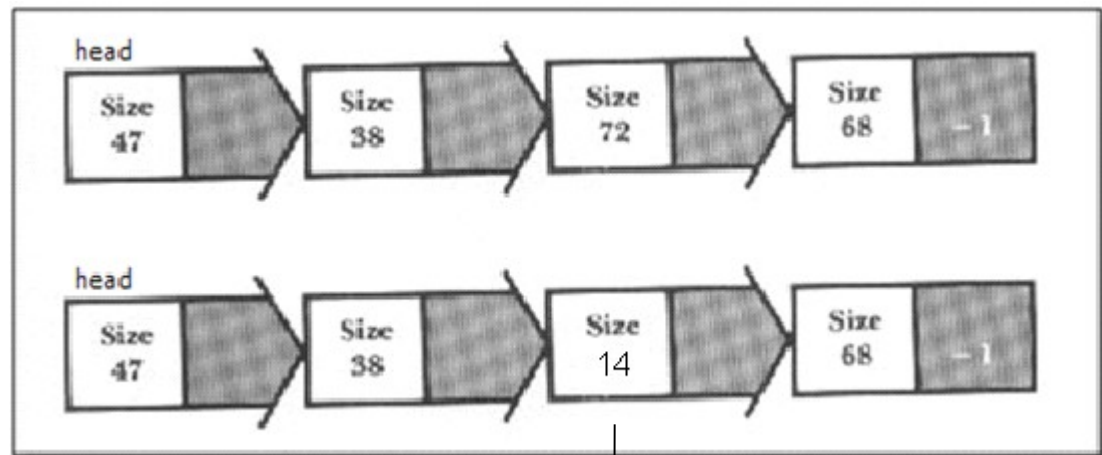
```
HEAD.FIRST_AVAIL: -1
40 Ames|John|123 Maple|Stillwater|OK|74075|64 Ham|A1|28 Elm|Ada|
OK|70332|.....45 Brown|Martha|62
5 Kimbark|Des Moines|IA|50311|
```

(b)

Fragmentação Interna

- Podemos combater a fragmentação interna mantendo os bytes não utilizados como um slot menor na lista
- No Exemplo 1 (slot de 72 bytes para um registro de 55):

Ante
s



Depois

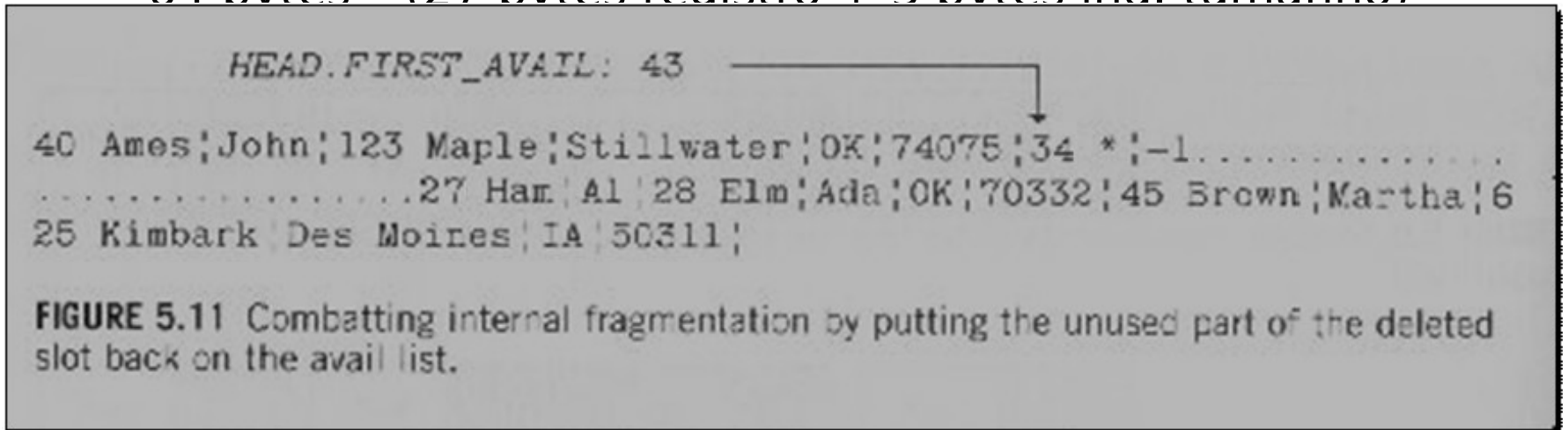


Porque 14 e não 17 ?

Fragmentação Interna

- No Exemplo 2:

- adicionar Ham|A1|28 Elm|Ada|OK|70332| a um slot de 64 bytes
- $64 \text{ bytes} - (27 \text{ bytes registro} + 3 \text{ bytes ind. tamanho}) =$



Fragmentação Interna

- Os 34b restantes podem ser utilizados para outro registro
 - Por exemplo: Lee|Ed|2 Rt|Ada|0K|74820| (25 bytes)
 - Exercício:** modifique o arquivo abaixo com essa adição

```
HEAD.FIRST_AVAIL: 43
40 Ames;John;123 Maple;Stillwater;OK;74075;34 *;-1.....
.....27 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.



Fragmentação Externa

- No exemplo anterior, após a inserção do novo registro:
 - Tem-se um registro disponível de $34 - (25 + 3) = 6$ bytes
 - Probabilidade de utilização desse registro é quase nula
 - Problema é denominado **Fragmentação Externa**



Fragmentação Externa

- Formas de Combater a Fragmentação
 - **Compactação** (off-line)
 - Gerar novamente o arquivo de tempos em tempos
 - **Coalescing**
 - Buscar por registros disponíveis que sejam logicamente adjacentes e uni-los em registros disponíveis maiores
 - **Prevenção**
 - Tentar evitar a fragmentação antes que ela ocorra através de estratégias de alocação de novos registros



Fragmentação Externa

- Estratégias de Alocação de Slots:
 - First-Fit
 - primeiro da lista que seja grande o suficiente
 - Best-Fit
 - aquele com tamanho mais parecido ao do registro
 - Worst-Fit
 - aquele com o maior tamanho de todos



Fragmentação Externa

- **First-Fit**

- estratégia mais simples de todas
 - requer apenas percorrer a lista
 - exatamente o que fizemos até agora
 - códigos 5.4, 5.5, 5.8 e 5.9 de (Folk & Zoellick, 1987)
- na verdade, não tenta prevenir fragmentação
 - responsabilidade da compactação e/ou *coalescing*



Fragmentação Externa

- **Best-Fit**

- estratégia mais intuitiva de todas
 - **requer manter a lista ordenada**
 - ordenação ascendente com o tamanho dos slots
 - demanda tempo computacional extra: não é mais possível sempre adicionar um slot ao início da lista
- **mas, na verdade, pode piorar fragmentação**
 - se slot não for perfeito, sobra é mínima !



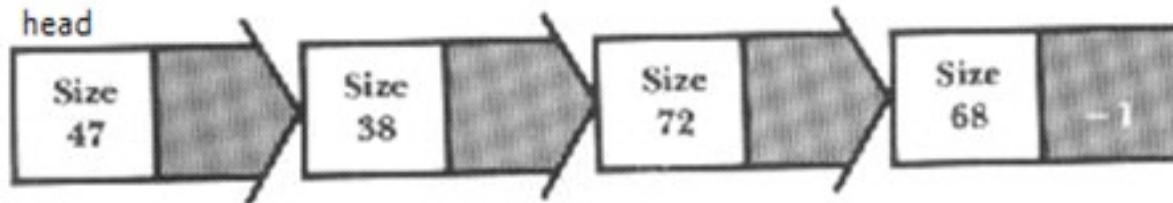
Fragmentação Externa

- **Worst-Fit**

- estratégia menos intuitiva de todas
 - requer manter a lista ordenada
 - ordenação descendente com o tamanho dos slots
 - mas tempo extra é compensado: se 1º slot não acomodar o registro, nenhum outro slot da lista acomodará !
- minimiza fragmentação
 - já que slot raramente é perfeito, sobra é máxima !

Exercícios

- Considere a seguinte lista de registros eliminados disponíveis (de tamanho variável) de um arquivo com estratégia de manutenção **First-Fit**:



- Ilustre como estaria esta lista caso:
 - A manutenção do arquivo fosse do tipo **Best-Fit**
 - A manutenção do arquivo fosse do tipo **Worst-Fit**



Exercícios

- No exercício anterior, considere que serão adicionados **ao arquivo** registros de tamanhos 75, 50 e 35, nesta ordem, e que então serão removidos **do arquivo** registros de tamanhos 10, 12, 83, 37 e 63. Ilustre a lista de registros vagos após cada uma dessas adições e eliminações.
 - Faça o exercício acima três vezes, uma para cada tipo de estratégia de manutenção (First, Best e Worst Fit).



Exercícios

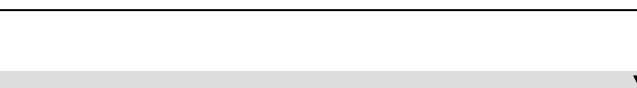
- Considere um arquivo com registros de tamanho variável (com indicador de tamanho) compostos de campos separados por delimitadores (“|”):
 - Adicione pelo menos 5 registros de tamanhos variados e ilustre o arquivo, incluindo o indicador de 1º reg. vago (`head.first_avail`)
 - Realize diversas adições e eliminações intercaladas de registros de tamanhos variados ilustrando o arquivo após cada operação, incluindo o indicador de 1º registro vago (`head.first_avail`)
 - Considere que a estratégia de manutenção **First-Fit** está em uso



Exercícios

- Considere o seguinte arquivo com registros de tamanho variável (com indicador de tamanho) compostos de campos separados por delimitadores (“|”):

head.first_avail = 42



```
40Ames|John|123 Maple|Stillwater|OK|74075|64*|-1|.....  
.....45Brown|Martha|625 Kimbark|  
Des Mcines|IA|50311|
```

- Mostre como fica o arquivo acima após a adição do registro:
 - `Lee|Ed|2 Rt|Ada|OK|74820|`
- Mostre então como fica o arq. resultante após eliminação do 1º reg.
- Aplique então *coalescing* e mostre como fica o arq. resultante.



Exercícios

- Tente implementar em linguagem C rotinas para adicionar e remover registros de **tamanho fixo** com campos de tamanho variável em um arquivo. Tome como ponto de partida os pseudo-códigos das rotinas `del_rec` e `pop_avail` para fazer a recuperação dinâmica de slots vagos.



Outros Exercícios

- Capítulo 5 (Folk & Zoellick, 1987)



Bibliografia

- **M. J. Folk and B. Zoellick, *File Structures: A Conceptual Toolkit*, Addison Wesley, 1987.**