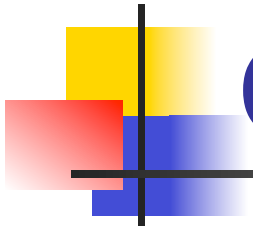


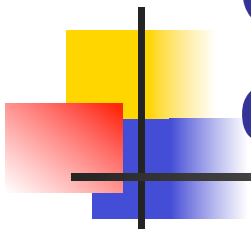
Organização de Arquivos

Thiago A. S. Pardo

Leandro C. Cintra

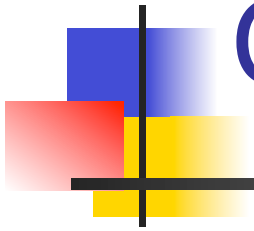
M.C.F. de Oliveira

Cristina D. A. Ciferri

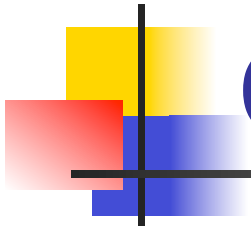


Organização de arquivos para desempenho

- Organização de arquivos visando desempenho
  - Complexidade de espaço
    - Compressão e compactação de dados
    - Reuso de espaço
  - Complexidade de tempo
    - Ordenação e busca de dados



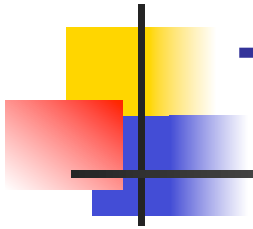
Compressão



Compressão de dados

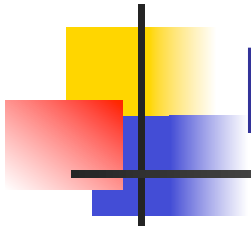
- A *compressão de dados* envolve a codificação da informação de modo que o arquivo ocupe menos espaço
  - Transmissão mais rápida
  - Processamento sequencial mais rápido
  - Menos espaço para armazenamento

- 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

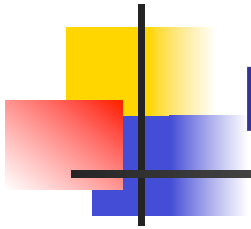
- Notação diferenciada
  - Redução de redundância
- Omissão de sequências repetidas
  - Redução de redundância
- Códigos de tamanho variável
  - Código de Huffman



Notação diferenciada

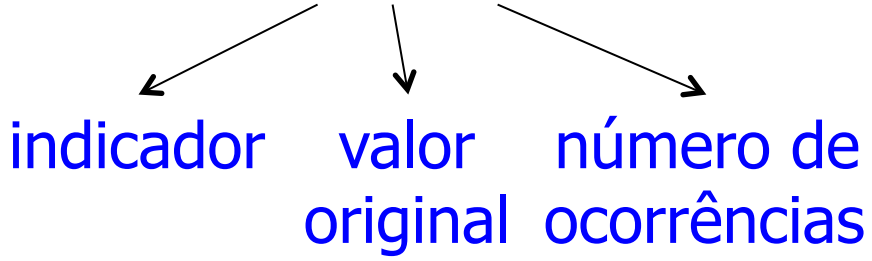
- Exemplo
  - Códigos de estado, armazenados na forma de texto: 2 bytes
    - 50 estados americanos
      - 2 bytes (para representação de 2 caracteres): NY, CA, etc.
    - Alternativa: com 50 opções, pode-se usar 6 bits
      - Por que?
        - É possível guardar a informação em 1 byte e economizar 50% do espaço
  - Desvantagens
    - Legibilidade
    - Necessidade de codificação e decodificação

Omissão de sequências repetidas

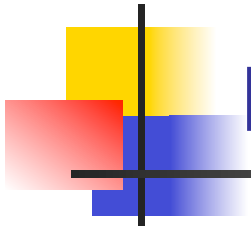


- Para a sequência hexadecimal
  - 22 23 24 24 24 24 24 24 24 25 26 26 26 26 26 26 26 25 24

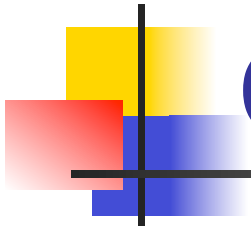
- Usando 0xff como código indicador de repetição (código de *run-length*)
  - 22 23 ff 24 07 25 ff 26 06 25 24



Omissão de sequências repetidas

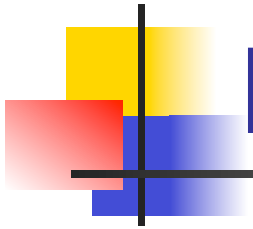


- Exemplo de uso
  - aplicações que possuem dados esparsos ou com muita repetição
    - imagens de céu
- Nem sempre garante redução de espaço
  - não pode ser usado em imagens com muita variação, por exemplo



Códigos de tamanho variável

- Código de Huffman
  - *Ideia*
    - valores mais frequentes são associados a códigos menores
    - 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



Exemplo: Código de Huffman

Alfabeto: {A, B, C, D}

Frequência: A > B > C = D

Possível codificação:

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

Cadeia: ABACCDA

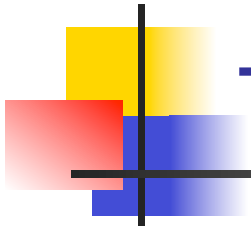
Código: 0110010101110

Codificação não deve ser ambígua

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

ACBA → 01010

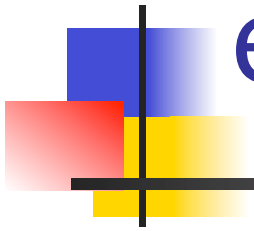
- É possível decodificar?

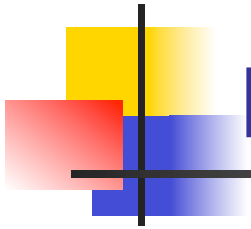


Técnicas de compressão irreversíveis

- 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
    - Trocam-se 16 pixels por 1
- Usado quando a informação perdida é de pouco ou nenhum valor

Compactação e reuso de espaço

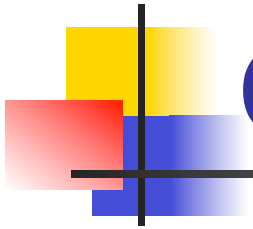




Manipulação de dados em arquivos

- Operações básicas
  - adição de registros
    - relativamente simples
  - remoção de registros
  - atualização de registros
    - eliminação e adição de um registro

Quando um registro é removido, deve-se posteriormente reutilizar o espaço do registro



Compactação

- Busca por regiões do arquivo que não contêm dados
- Recupera os espaços perdidos

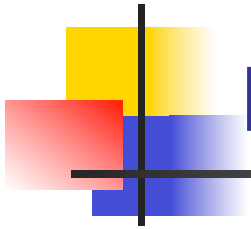
Abordagem Estática

programa "compactador"

remoção do registro → gera espaço sem dados úteis → reagrupa os registros e diminui o tamanho do arquivo

- Técnica para reconhecer registros removidos: remoção lógica
  - atribuir um valor para um campo do registro
  - usar um campo extra

Como reusar o espaço dos registros removidos



- Não faz nada em um intervalo de tempo Δt
- Durante Δt

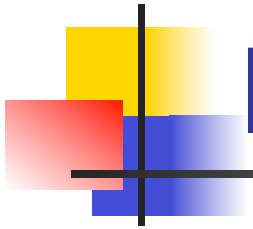
remoção lógica

  - registros removidos são marcados, porém não são reaproveitados
  - novas inserções são realizadas no final do arquivo
  - buscas desconsideram os registros marcados como removidos

- Após Δt

remoção física

  - programa é executado para reconstruir o arquivo
  - todos os registros removidos são descartados



Exemplo

Arquivo original

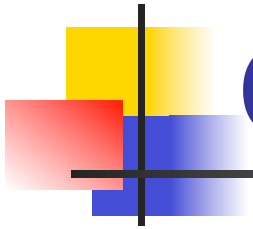
Maria|Rua 1|123|São Carlos|.....
João|Rua A|255|Rio Claro|.....
Pedro|Rua 10|56|Rib. Preto|.....

Após remover segundo registro

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

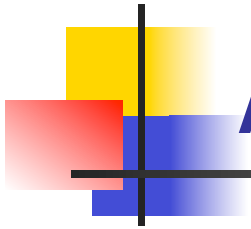
Após compactação do arquivo

Maria|Rua 1|123|São Carlos|.....
Pedro|Rua 10|56|Rib. Preto|.....



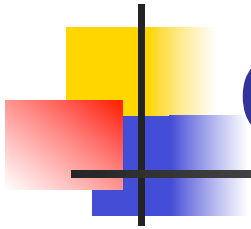
Observações

- Técnica pode ser aplicada a
  - registros de tamanho fixo
  - registros de tamanho variável
- Frequência para se aplicar a técnica
  - depende da aplicação
  - depende da procentagem de registros marcados como removidos
  - exemplos: toda noite, em certas datas, no final do ano, a cada semestre



Abordagem 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
1. identificar os registros marcados (existem registros marcados ≈ tem espaço?)
2. localizar os espaços antes ocupados por esses registros, sem buscas exaustivas



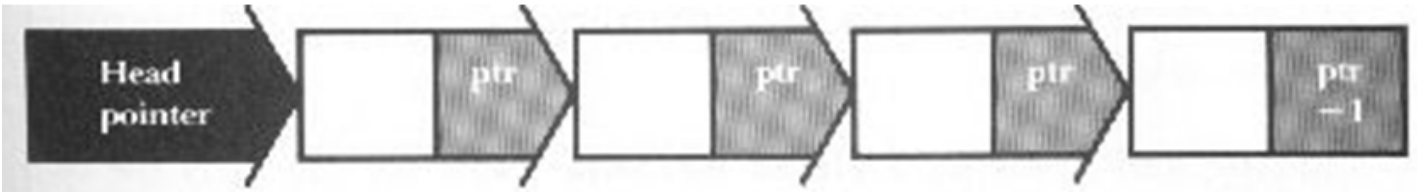
Como localizar os espaços vazios?

- 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)

resolve 1 e 2

Registros de tamanho fixo

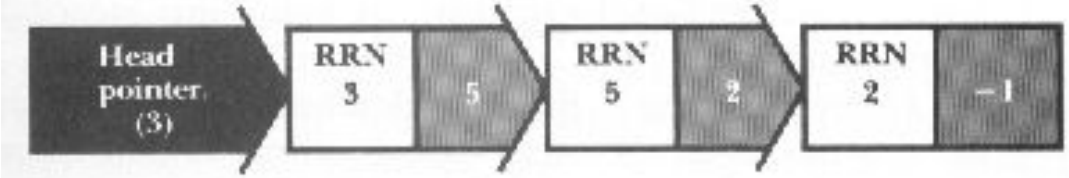
Lista encadeada: formato



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

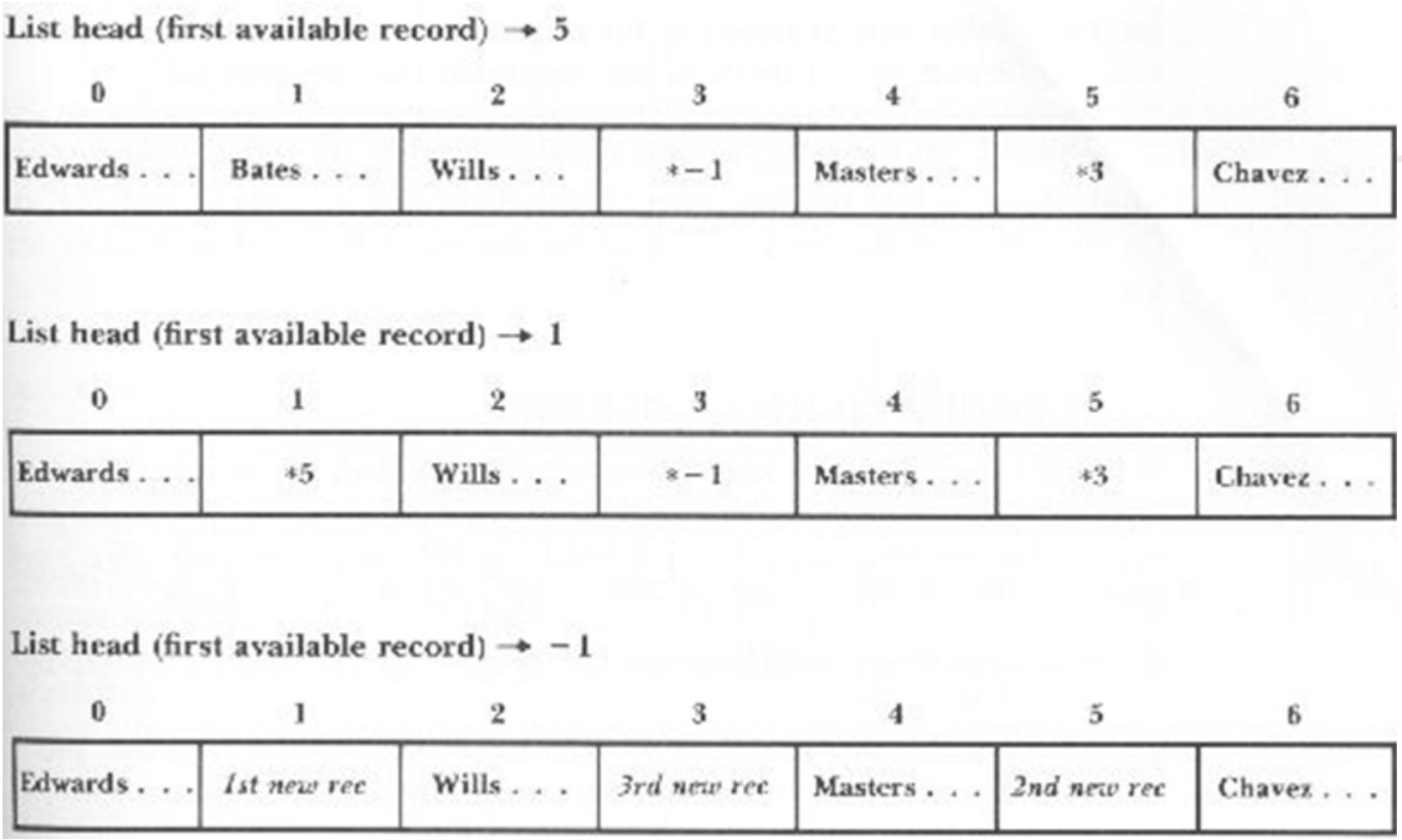
Número de registros na lista

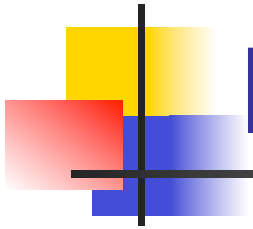
Remoção do registro de RRN 2 e depois o de RRN 5



Remove depois o de RRN 3

Exemplo

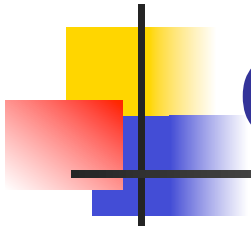

List head (first available record) → 5
0 1 2 3 4 5 6
Edwards . . . | Bates . . . | Wills . . . | *−1 | Masters . . . | *3 | Chavez . . .
Remoção de 3 e depois 5
List head (first available record) → 1
0 1 2 3 4 5 6
Edwards . . . | *5 | Wills . . . | *−1 | Masters . . . | *3 | Chavez . . .
Remoção de 1
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 . . .
Adição de 3 registros



Perguntas

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

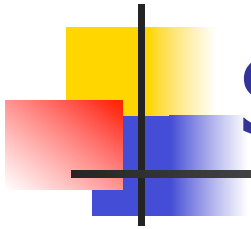
- A pilha poderia ser mantida na memória principal?
  - Vantagens?
  - Desvantagens?



Como localizar os espaços vazios?

- Registros de tamanho variável

- Lista encadeada modificada
  a) como ligar os registros removidos dentro da lista
  b) algoritmo para inserir novos registros na lista
  c) algoritmo para encontrar e remover registros da lista quando necessário

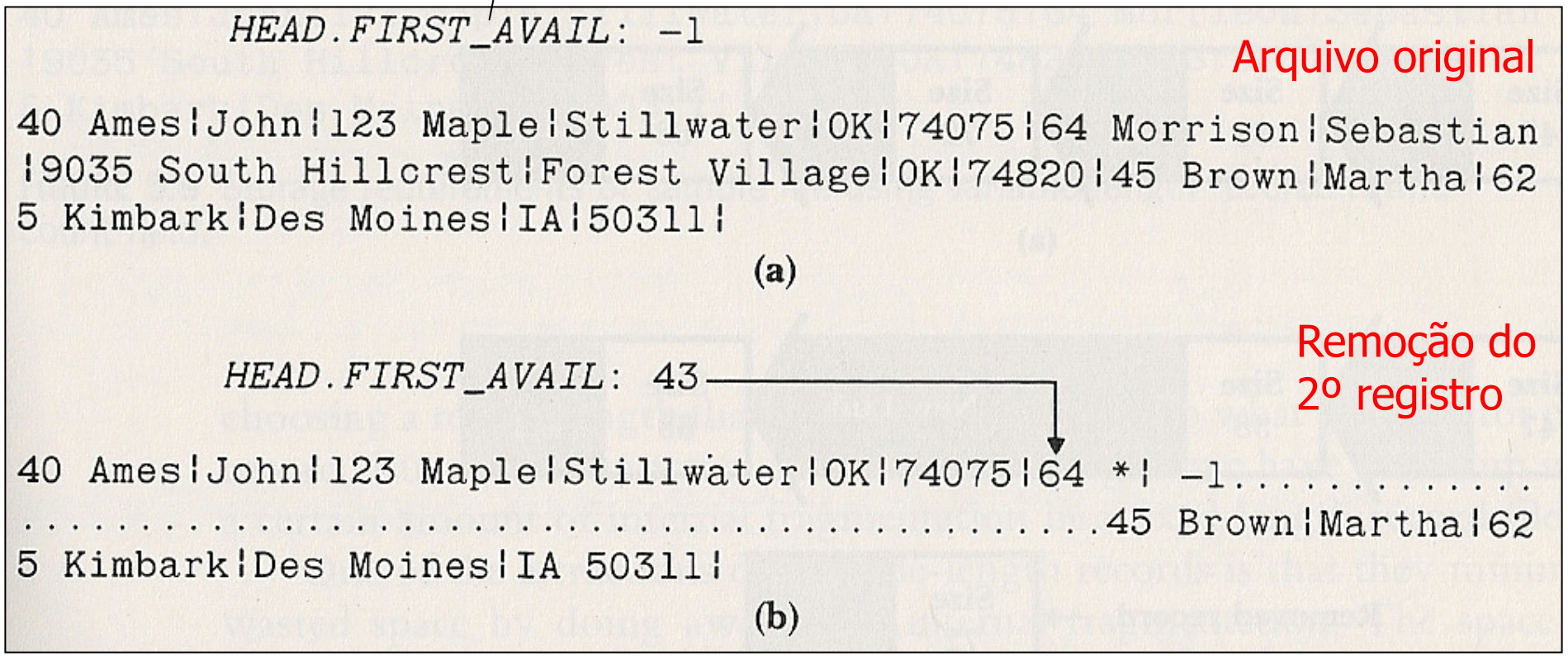


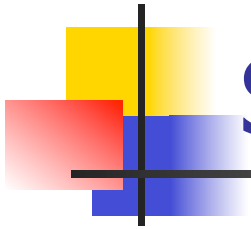
Solução

- Exemplo
  - método de indicador de tamanho para cada registro
- Para (a) *ligação dos registros*
  - marcação dos registros eliminados via um marcador especial
  - campo de ligação binário para o próximo registro removido
    - contém o byte offset a partir do início do arquivo
    - *não é possível usar RRNs (registros de tamanho variável)*

Remoção de registros

No registro de cabeçalho



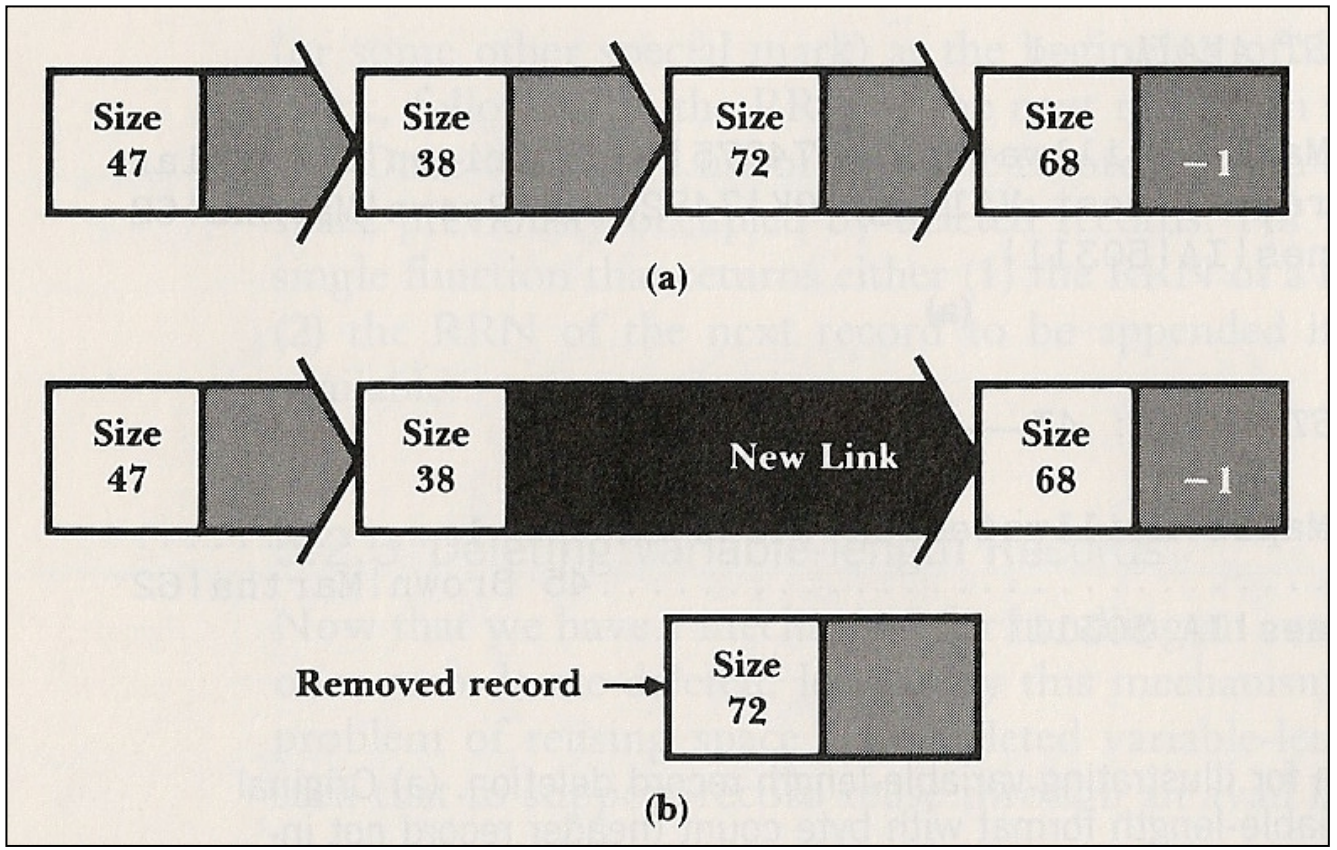


Solução

- Para (b) *inserir novos registros*
  - coloca-se o novo registro na frente da lista (como uma pilha)
- Para (c) *encontrar e usar o espaço do registro*
  - realiza uma busca sequencial na lista
    - se encontrou espaço do tamanho certo, então reutiliza o espaço usando uma estratégia de alocação
    - caso contrário, armazena o registro no final do arquivo

O tamanho do registro que foi removido deve ser do tamanho certo, ou seja, "grande o suficiente" para que os dados do novo registro usem aquele espaço

Adição de um registro de 55 bytes: exemplo



Antes da escolha

Depois da escolha