Algoritmos e Estruturas de Dados II

Profa. Debora Medeiros

Manutenção de Arquivos

Adaptado dos Originais de:

Ricardo Campello

Thiago Pardo

Leandro C. Cintra

Maria Cristina F. de Oliveira

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 seqüencial 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 seqüências 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...

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?

Notação diferenciada

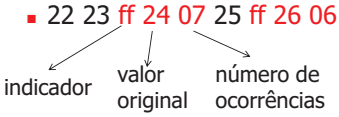
- Desvantagens?
  - Legibilidade
  - Codificação/decodificação
    - Complexidade dos softwares de processamento

Omissão de sequências repetidas

- Para a seqüência hexadecimal
  - 22 23 24 24 24 24 24 24 24 25 26 26 26 26 26 26 25 24
- Como melhorar isso?

Omissão de sequências repetidas

- Para a seqüência hexadecimal
  - 22 23 24 24 24 24 24 24 24 25 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

- Bom para dados esparsos ou com muita repetição
  - Imagens do céu, por exemplo
- Garante redução de espaço sempre?

Códigos de tamanho variável

- Representação binária de caracteres
- Comprimento variável

Códigos de tamanho variável

- Representação binária de caracteres
- Comprimento variável
  - Relação com a frequência dos caracteres
    - Código morse
  - Código livre de prefixo

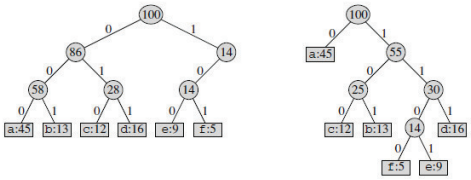
Código de Huffman

- Exemplo:
  - alfabeto de 6 letras: "a ", "b", "c", "d", "e" e "f"
  - adeba → 000011100001000 → 011111011010

Caracteres	a	b	c	d	e	f	# bits (x 1000)
Frequência (x 1000)	45	13	12	16	9	5	-
Código de comprimento fixo	000	001	010	011	100	101	300
Código de tamanho variável	0	101	100	111	1101	1100	224

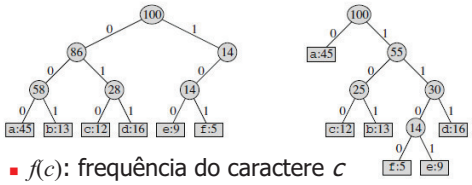
Código de Huffman

- Representação com árvore binária



Código de Huffman

- Representação com árvore binária



- $f(c)$: frequência do caractere c
- $d_T(c)$: profundidade da folha que representa c
  - Comprimento do código que representa c
- $B(T) = \sum_{c \in C} f(c) d_T(c)$: número de bits necessários
  - custo da árvore T

Código de Huffman

```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN(Q)
6          right[z] ← y ← EXTRACT-MIN(Q)
7          f[z] ← f[x] + f[y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)
```

- Q: heap binário mínimo

Código de Huffman

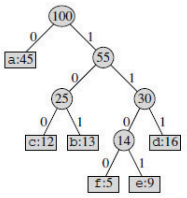
```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN(Q)
6          right[z] ← y ← EXTRACT-MIN(Q)
7          f[z] ← f[x] + f[y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)
```

a	b	c	d	e	f
45	13	12	16	9	5

Código de Huffman

```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN(Q)
6          right[z] ← y ← EXTRACT-MIN(Q)
7          f[z] ← f[x] + f[y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)
```

a	b	c	d	e	f
45	13	12	16	9	5



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
- Onde se usa isso?

Compactação e reuso do espaço

Manutenção de Arquivos

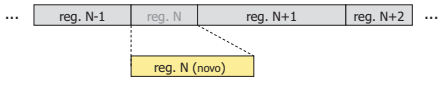
- Projetista deve considerar modificações no arquivo
  - **Adição**, **atualização** e **eliminação** de registros
- Problema é simples quando:

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...

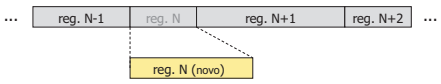
Manutenção de Arquivos

- P. ex., **atualizar** um registro que **aumente de tamanho**:
  - O que fazer com os dados adicionais?



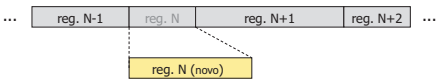
Manutenção de Arquivos

- 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



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

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|.....

Compactação

- Software deve ser capaz de ignorar registro apagado
- Vantagens...
  - Recuperação de registros
    - Campo especial

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
    - Como?

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:
  - Registro cabeçalho armazena RRN do 1º registro vago

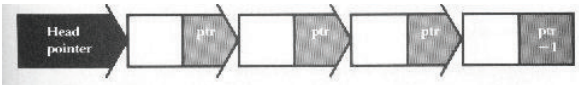
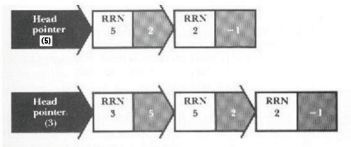


FIGURE 5.4 A linked list.

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 ⇔ registro eliminado do arquivo
  - remoção da pilha ⇔ registro adicionado ao arquivo

Registros de Tamanho Fixo

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

Registros de Tamanho Fixo

Exemplo (registros com 55 bytes)

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 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

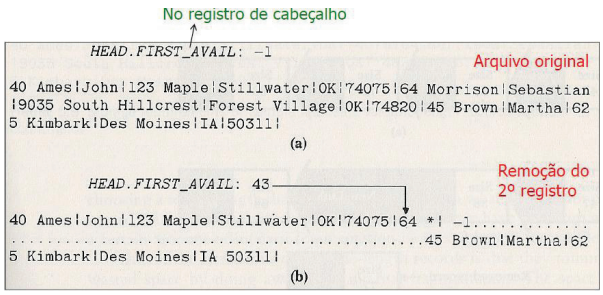
Registros de Tamanho Variável

- Registros não são acessíveis por RRN...
- **Solução**:
  - Armazenar os byte offsets na lista encadeada
    - ao invés dos RRNs

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



Registros de Tamanho Variável

- O problema está solucionado?

Registros de Tamanho Variável

- O problema 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 !

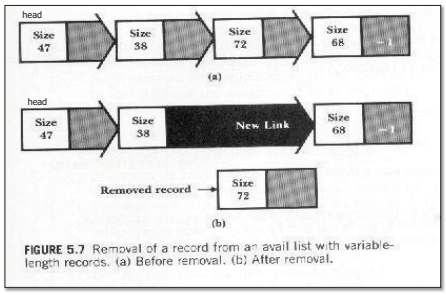
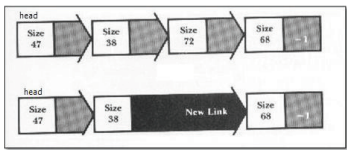


FIGURE 5.7 Removal of a record from an avail list with variable-length records. (a) Before removal. (b) After removal.

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|OK|70332|` (27 bytes)

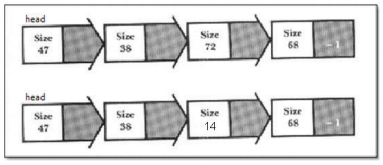
```
        HEAD.FIRST_AVAIL: 43                          Situação anterior
40 Ames¦John¦123 Maple¦Stillwater¦OK¦74075¦64 *¦ -1.............
.....................................................45 Brown¦Martha¦62
5 Kimbark¦Des Moines¦IA¦50311¦
                        (a)
                                                      Após adição de
        HEAD.FIRST_AVAIL: -1                          novo registro
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):

Antes

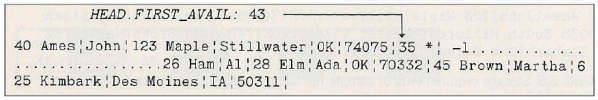
Depois



Fragmentação Interna

- No Exemplo 2:
  - adicionar `Ham|Al|28 Elm|Ada|OK|70332|` a um slot de 64 bytes

Adição do espaço restante à lista



Fragmentação Interna

- Os 35b restantes podem ser utilizados para outro registro
  - Por exemplo: `Lee|Ed|2 Rt|Ada|OK|74820|` (25 bytes)
- Após a inserção do novo registro:
  - Tem-se um registro menor ainda disponível
  - Probabilidade de utilização desse registro é quase nula
  - Problema é denominado **Fragmentação Externa**
    - Soluções?

Fragmentação Externa

- Formas de Combater a Fragmentação
  - **Compactação** (off-line)
    - Gerar novamente o arquivo de tempos em tempos
  - ***Coalescing***
    - Buscar por registros disponíveis 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
  - 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
    - Por que?

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: por que?
  - minimiza fragmentação ?

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 !

Bibliografia

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