INFORMAZÔNIA/84

I SIMPÓSIO DE INFORMÁTICA NA AMAZÔNIA
I EXPOSIÇÃO DE INFORMÁTICA NA AMAZÔNIA

Campus da Universidade Federal do Pará
27 a 31 de Agosto de 1984

ANAIS
DO
I SIMPÓSIO DE INFORMÁTICA NA AMAZÔNIA

I SIMPÓSIO DE INFORMÁTICA NA AMAZÔNIA

CAMPUS DA UNIVERSIDADE FEDERAL DO PARÁ

27 a 31 de Agosto de 1984

OPERAÇÕES DE ABSTRAÇÃO PARA MANUTENÇÃO DE LISTAS INVERTIDAS EM MEMÓRIA SECUNDÁRIA

Vieira, Ewerton A. P.
Guimarães, Mário André N.

Palavras-Chave: Arquivo indexado, Árvores B, CP/M, Linguagem C, Método de Acesso, Lista Invertida.

Resumo

Em sistemas de informação, não são pouco frequentes os acessos baseados em chaves secundárias ou em chaves que não são únicas. Apresentamos um conjunto de primitivas implementadas sob forma de subprogramas funções que permitem a manipulação de listas invertidas em arquivos em disco de forma simples e compacta. Temos então as estruturas das listas, que são tão importantes na construção de bancos de dados, vistas como um tipo abstrato e as operações associadas sobre elas.

As primitivas de listas invertidas foram desenvolvidas utilizando dois outros níveis de operações de abstração:

1) Método de Acesso Paginafo (MAP), responsável pelo mapeamento em memória secundária das páginas lógicas.

2) Método de Acesso Indexado (MAI); responsável pela manipulação de registros de tamanho variável contendo chaves únicas também de tamanho variável.

Descrevemos a especificação e implementação destas operações e estruturas de dados utilizando a estrutura de arquivos de acesso direto e indexado, também construído por nós e funcionando sob o CP/M.

Discute-se no trabalho cada um dos três níveis da implementação, as opções e decisões tomadas, restrições e dificuldades encontradas ao longo do desenvolvimento, além das possíveis alterações e melhoramentos que poderiam ser realizados.

I INTRODUÇÃO

O objetivo do trabalho constou em fornecer mecanismos de recuperação de informação para trabalhar com um grande volume de dados. Sentimos a necessidade de utilizar um sistema que permitiria acessos a registros baseados em chaves que não são únicas, ou chaves secundárias. Optamos por um arquivo invertido, método muito conhecido para consultas baseadas em chaves secundárias.

Definimos então um conjunto de operações sobre listas invertidas que seria implementadas como funções de manipulação de arquivos invertidos. Essas operações foram construídas usando outras operações definidas no Método de Acesso Indexado (MAI) e no Método de Acesso Paginado (MAP). A este conjunto de funções denominadas de Método de Acesso Invertido (MAV).

No ítem dois apresentamos a descrição das operações de abstração definidas sobre os três níveis. Em seguida, descrevemos mais detalhadamente cada um dos níveis. Por fim, no ítem V, concluimos fazendo uma avaliação e alguns comentários sobre a implementação.

II ARQUITETURA

O ambiente onde os sistemas seriam inicialmente utilizados consistiria de um microcomputador baseado no microprocessador 8086 (1), construído no LESC (Laboratório de Engenharia e Sistemas de Computação) da PUC/RJ. Este computador está configurado com 128k bytes de memória e dois acionadoes de discos flexíveis de oito polegadas, densidade simples, face simples. O sistema operacional disponível é o CP/M-86 (2), que oferece algumas primitivas de acesso a disco bastante limitadas. Também disponíveis para o CPM-86 encontram-se o ASM86, montador de linguagem de máquina e a linguagem de alto nível C (3). Foi esta última a escolhida para a implementação efetiva dos sistemas de recuperação de informação devido às suas interessantes características de transportabilidade, estruturação, modularidade e eficiência.

Inicialmente seria produzida, então uma biblioteca de funções primitivas que estenderiam a linguagem C para que se pudesse manipular arquivos indexados (nível 1 e 2) e arquivos invertidos (nível 3) com o mínimo de dificuldade possível. Além disso, seria interessante que o método de acesso fosse o mais independente possível do sistema operacional hospedeiro, fator fundamental determinante de sua transportabilidade.

As árvores B+ trabalham com unidades de acesso à memória secundária chamadas páginas. Uma página pode conter índices ou, quando localizada em uma folha, registros. A página corresponde a uma área de tamanho fixo, um segmento do arquivo.

Tendo em vista as caracterísricas acima mencionadas do MAI, dividimo-lo em duas partes distintas, dois níveis de construção:

1. Método de Acesso Paginado, responsável pela manipulação de páginas pertencentes a um arquivo, em memória principal ou em disco.

2. Método de Acesso Indexado propriamente dito, utilizando-se das primitivas oferecidas pelo Método de Acesso Paginado e de árvores B+.

Uma vez implementado o Método de Acesso Indexado (MAI), constatamos uma restrição: as chaves são únicas. O gerenciamento de chaves duplicatas utilizando árvores B torna-se difícil devido a possibilidade de ocorrer uma quevra dividindo duas chaves idênticas, uma em cada página. Qual seria a chave separadora (Figura 1)?

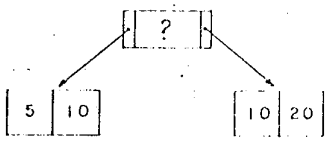

fig. 1

Assim sendo, ficamos impossibilitados de fazer pesquisas baseados em chaves secundárias, cuja utilidade em sistemas aplicativos é incontestável (5, 6). Resolvemos então implementar um Método de Acesso que permitisse a manipulação de listas invertidas, de forma mais simples e abstrata. Passamos então a ter mais uma abstração, com a seguinte configuração:

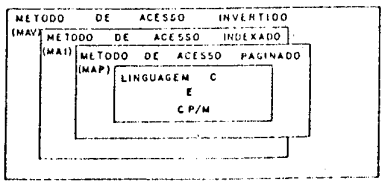

fig. 2

As primitivas do Método de Acesso Invertido podem ser divididas em três grupos:

1) **Iniciação/Encerramento**
   1.a) Abre um arquivo invertido;
   1.b) Cria um arquivo invertido;

1.c) Fecha um arquivo invertido;

2) **Manipulação de Listas**
   2.a) Criar uma lista invertida;
   2.b) Abrir uma lista invertida;
   2.c) Abrir a próxima lista na ordem;
   2.d) Posicionar o apontador de próxima lista em função de uma chave e de uma relação.
   2.e) Excluir uma lista;

3) **Manipulação dos Elementos de uma Lista**
   3.a) Lê o elemento corrente de uma lista;
   3.b) Lê o próximo elemento da lista na ordem;
   3.c) Inclui um elemento na lista;
   3.d) Exclui o elemento corrente de uma lista;
   3.e) Posiciona o primeiro elemento como a corrente de uma lista;

O Método de Acesso Indexado por sua vez, oferece o conjunto de primitivas abaixo:

a) Criar um arquivo indexado;
b) Abrir um arquivo indexado;
c) Fechar um arquivo indexado;
d) Incluir um registro com uma chave única, inexistente;
e) Ler um registro dada uma chave já existente;
f) Alterar um registro;
g) Excluir um registro;
h) Posicionar o apontador de próximo registro em função de uma chave e de uma relação.
i) Ler o próximo registro na ordem;
j) Ler o registro anterior ao corrente;
k) Posicionar o apontador de próximo registro no início do arquivo;
l) Posicionar o apontador de próximo registro no fim do arquivo.

De forma a atender aos requisitos do MAI, o Método de Acesso Paginado oferece as seguintes primitivas:

a) Criar um arquivo paginado;
b) Abrir um arquivo paginado;
c) Fechar um arquivo paginado;
d) Alocar uma página;
e) Liberar uma página;
f) Ler uma página;
g) Gravar uma página;
h) Fixar uma página;
i) Desfixar uma página.

Descrevemos abaixo a implementação dos conjuntos de primitivas na ordem de desenvolvimento e que caracteriza a técnica "bottom up".

III MÉTODO DE ACESSO PAGINADO

O Método de Acesso Paginado tem como objetivo gerenciar a manutenção de uma quantidade de páginas de um arquivo em memória principal. Naturalmente, quanto maior o número de páginas em memória, maior a velocidade de acesso média alcançada pelo sistema. Visamos exatamente reduzir o tempo médio de acesso que depende diretamente do número de referências à memória secundária (um recurso caro).

Na criação ou abertura de um arquivo paginado se recebe um ponteiro, chamado identificador de arquivo paginado, que deverá ser usado para referenciar todo o acesso seguinte ao arquivo. Além disso, é alocada uma área de trabalho para conter as páginas que ficarão em memória e a tabela que controla estas páginas. Cada página da tabela tem associados a "hora" em que ocorreu o último acesso a ela, e o arquivo a que pertence. Além disso, como a página pode ser fixada na memória quando a aplicação assim o exigir, torna-se necessário também manter uma informação dizendo se a página é removível ou não.

A informação da "hora" do último acesso é usada na implementação do mecanismo de seleção para remoção da página menos recentemente utilizada (LRU em inglês). Isto permite que se libere a página que foi acessada pela última vez a mais tempo entre as removíveis. A liberação da página se dará quando for necessário espaço em memória para uma outra página lida ou criada (alocada). Existe um contador associado ao arquivo paginado que indica a "hora" atual do MAP e que é incrementado a cada acesso de leitura ou gravação.

As primitivas de fixação e desfixação de páginas na memória vem atender, basicamente, as necessidades de manipular várias páginas na memória desde após algum acesso a ela e até o próximo acesso à outra página qualquer, a menos que ela esteja explicitamente fixada.

Para contornar a incapacidade do CP/M de receber de volta as áreas em disco previamente alocadas porém não mais utilizadas o MAP deverá também, gerenciar uma pilha com as páginas liberadas, de onde serão retiradas quando se tornar necessário, isto é, na alocação das páginas.

A utilização do Método de Acesso Paginado se dá através de dez rotinas para o usuário e mais rotinas internas ao MAP. As primeiras nove rotinas correspondem exatamente às primitivas oferecidas e previstas no projeto do MAP, sendo que uma décima foi colocada posteriormente visando dar mais uma facilidade: ela retorna o número da página dado o seu endereço na memória. Além disso, existe uma função auxiliar que retorna o número de páginas possíveis de serem alocadas com o espaço de memória restante, e que permite a máxima utilização da memória disponível independente da configuração da máquina utilizada.

IV MÉTODO DE ACESSO INDEXADO

O objetivo do Método de Acesso Indexado é fornecer um conjunto de operações primitivas ao sistema de aplicação para a manipulação de registros de tamanho variável a serem recuperados por uma chave única, também de tamanho variável. Descrevemos abaixo as características da nossa implementação.

O MAI é constituído utilizando-se das primitivas do MAP. Sua única ligação com o meio físico é através do parâmetro que define o tamanho da página. Conseguiu-se, desta forma, também, uma independência neste nível em relação ao Sistema Operacional.

Basicamente, existem dois tipos de páginas no MAI: as páginas de índice e as páginas folha. As primeiras são responsáveis pelo armazenamento dos índices que apontam em última análise para as páginas do tipo folha. Um arquivo vazio não possui páginas índice. As páginas folha são as que contém os registros. Elas formam uma linha duplamente encadeada utilizada para a pesquisa sequencial.

Uma página do tipo índice tem a seguinte disposição interna:

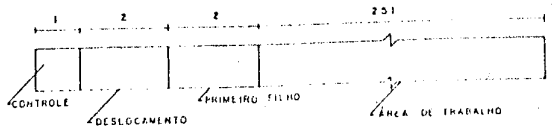

Fig. 3

e uma página do tipo folha:

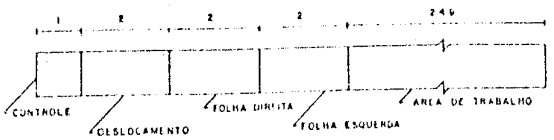

Fig. 4

As páginas podem, ainda, ser raiz ou não. Usa-se os 2 bits

menos significativos do byte de controle da página para identificá-las. O bit 0 indica se é índice (1) ou folha (0) e o bit 1 se a página é raiz (1) ou não, e RAIZ(11). A página única é raiz e folha ao mesmo tempo. A criação de uma árvore gera uma página única, vazia.

Um registro de tamanho variável, como utilizado nesta implementação, tem no primeiro byte seu tamanho total. Imediatamente após este byte, segue-se o campo da chave de acesso e, em seguida, vem os demais campos, se houver. Observe que esta escolha limita o tamanho total de registro a 256 bytes, o que é mais do que suficiente para as aplicações pretendidas. A alteração deste parâmetro, contudo, é trivial.

A chave de acesso do registro, por sua vez, também é um campo de tamanho variável. Assim sendo, o seu primeiro byte é utilizado para armazenar o tamanho total do campo e é seguido da chave propriamente dita. Sempre que falamos da chave de acesso como parâmetro de entrada ou no interior de um registro, estamos nos referenciando ao campo de tamanho variável que possui a chave real e o tamanho do campo.

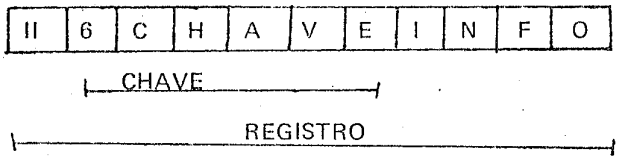

Fig. 5

Nesta implementação, escolheu-se por colocar os registros e os índices no mesmo arquivo para tornar a gerência das estruturas de dados e suas representações físicas mais simples. Por outro lado isto impacta na velocidade de acesso visto que a altura da árvore será em geral maior, pois mais folhas serão necessárias para armazenar os dados ao invés de armazenar só ponteiros em um arquivo e os dados em outro. Nada impede, porém, utilizarmo-nos do método de acesso atual para realizar esta modificação. Basta acrescentar-se mais uma "casca" externa ao mai, oferecendo-se assim estas mesmas operações com um único arquivo para os índices e outro para os dados.

Na criação e abertura, a raiz é fixada na memória com o objetivo de reduzir o número de acessos à memória secundária visto que todo acesso direto é feito através dela. Assim, se a árvore tiver altura quatro, por exemplo, serão necessários no máximo três acessos no caso de nenhuma página estar presente na memória.

O procedimento da exclusão garante a árvore tornar-se, no pior dos casos, uma árvore binária balanceada. Se a árvore tiver altura H teremos no mínimo, portanto, 2** (H-1) registros, podendo-se acessá-los em um máximo de (H-1) acessos. Na exclusão, a concatenação só será efetuada, quando a página estiver totalmente vazia. A sub-árvore que ficar pendente é reincluída em uma página "irmã" (de mesmo nível). Conseguimos, com isso, diminuir a complexidade da operação de exclusão, que prevê a redistribuição de índices ou registros nas baixas ocupações das páginas. Naturalmente, tal decisão implica em um comprometimento com a utilização de espaço em memória secundária, mas, por outro lado, diminui a frequência de alterações na árvore. No caso de nossas aplicações, onde a exclusão ocorrerá raramente não se perderá muito.

Na inclusão, a quebra será efetuada quando a página não puder suportar a inclusão, pois a página é de tamanho fixo. O conteúdo da página será dividido com uma nova página alocada e a inserção então realizada na página conveniente.

Existe um mecanismo que percebe se está fazendo inclusões sequenciais, possibilitando fazer com que, neste caso, a divisão dos registros não seja feita buscando obter níveis idênticos de ocupação, mas deixando a página cheia intacta e realizando a inclusão na nova página alocada. O efeito obtido é que arquivos criados pela inclusão sequencial e ordenada pelas chaves serão mais compactos que se fossem criados através de inclusões aleatórias.

O emprego de técnicas de compactação para o armazenamento dos mesmos por página é, por conseguinte, diminuor a altura da árvore de acesso.

V MÉTODO DE ACESSO INVERTIDO

Um arquivo invertido lógico consiste em 2 arquivos CP/M. Um, o arquivo indexado, gerenciado pelo Método de Acesso Indexado, contém as chaves das listas e ponteiros para seus respectivos elementos. O outro, o arquivo paginado, gerenciado pelo do Método de Acesso Paginado, contém os elementos das listas. Cada registro do arquivo indexado contém uma chave secundária que é o nome da lista e dois ponteiros. Os ponteiros apontam para as páginas no arquivo paginado que contém o primeiro e o último elemento respectivamente.

Ao criar uma lista invertida, o que significa criar uma chave secundária, os dois ponteiros assumem valor nulo. A figura (2.1) mostra a visão lógica de um arquivo invertido sobre o campo estado. Para cada estado (nome da lista), estão associados um conjunto de supridores (elementos de uma lista). A figura (2.2) apresenta a visão física correspondente. Quando um elemento é incluído na lista, uma página do arquivo paginado é alocada. A chave (RS) exemplifica como seria uma lista após ser criada. Como não há elementos associados a esta lista, seus dois ponteiros assumem o valor nulo.

As inserções dos próximos elementos serão realizadas sempre no fim da lista. As chaves (MG) e (RJ) mostram este caso. Se a página não suportar a inserção de um novo elemento, uma nova página é alocada e inserida na lista de páginas desta chave secundária. Com a inserção do elemento (S9) na lista referente a (SP), por exemplo, uma nova página teve que ser alocada e encadeada na lista.

(Vide Figuras 6.1 e 6.2)

As primitivas do Método de Acesso Invertido podem ser divididas em três grupos: Inicialização e Encerramento, Manipulação de listas e Manipulação dos elementos de uma lista. Para acessar uma lista (primitivas de manipulação de listas) basta ao usuário fornecer a cadeia que é a chave da lista. Isto difere do Método de Acesso Indexado, onde os parâmetros passados são o registro montado ou a chave montada. O resultado consiste num ganho de simplicidade para o usuário, acarretando porém a restrição de que a chave não poderá conter o caracter OOH, o qual é o delimitador de uma cadeia em C.

Da mesma forma que no Método de Acesso Paginado (MAP) e no Método de Acesso Indexado (MAI), na criação ou abertura de um arquivo invertido se recebe um ponteiro, chamado identificador de arquivo (no caso do Método de acesso Invertido, identificador de arquivo invertido), que deverá ser usado para referenciar todo o acesso subsequente ao arquivo.

Para acessar os elementos da lista (primitivas de manipulação de elementos de lista), o parâmetro passado é um registro do qual o primeiro Byte contém o tamanho do elemento e o resto do registro contém o elemento da lista.

Consideramos que, para a chave de uma lista, a restrição de não conter OOH não traz grandes problemas, enquanto que a vantagem em simplicidade é grande. Quanto aos elementos da lista, a restrição de não poder ter OOH tornaria muitas aplicações impraticáveis.

VI CONCLUSÕES

As operações primitivas especificadas e implementadas, conseguem ser tanto mais simples quanto poderosas, permitindo sua utilização em diversas aplicações de vários graus de complexidade. Elas preveem mecanismos bastante adequados à manipulação de dados em memória secundária utilizando conceitos que se sobrepõem em níveis de abstração distintos. Como exemplo de utilização do MAI podemos citar um sistema de informação desenvolvido na PUC para controlar os estoques de circuitos integrados do LESC (14). O MAV foi utilizado para a implementação de um Sistema de Pastas Eletrô-

nicas no projeto de automação de escritórios em redes locais, também desenvolvido na PUC/RJ (16).

O conjunto é modular, com rotinas totalmente parametrizadas e independentes umas das outras. Essa modularidade permite: boa documentação, modificação de uma rotina sem ter que alterar as demais, teste de cada rotina individualmente e a possibilidade de estender o sistema para outras aplicações.

Podemos definir um sistema de propósito geral (generalidade), como sendo um que pode ser utilizado para uma enorme variedade de aplicações. Este conjunto de primitivas satisfaz esta característica também. Praticamente qualquer sistema de informação pode ser implementado utilizando as primitivas do MAI e do MAV.

As rotinas foram escritas em C, o que nos garante uma boa confiabilidade contra falhas de software e também grande transportabilidade. Embora o projeto do sistema tenha sido "top-down", a implementação e os testes foram feitos "bottom-up".

Uma das limitações do nosso sistema é a ausência de mecanismo contra falhas físicas (p. ex., falta de energia). Poderíamos acrescentar tecnicas de recuperação dinâmica de falhas. Porém, estas técnicas se mostram fracas diante das limitações do CP/M com respeito a integridade dos arquivos em tal situação.

Finalmente, podemos dizer que o sistema é muito simples para o usuário ou o programador do nível de aplicação utilizar. Basta possuir a biblioteca contendo as rotinas primitivas do MAV, MAI E MAP, e conhecer suas definições (14).

ESTADO	PONTEIRO
MG	↑S5
RJ	↑S1 ↑S2 ↑S4
RS	
SP	↑S3 ↑S7 ↑S8 ↑S9

Fig. 6.1
Chave secundária = Estado

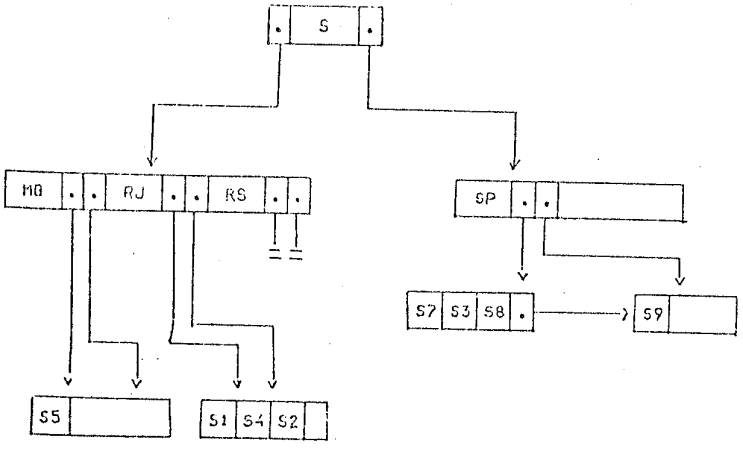

Fig. 6.2
Estrutura Física do MAV

BIBLIOGRAFIA

(1) — Vieira, Ewerton A. P. e Guimarães Mário A. M. "Implementação de Um Método de Acesso Indexado com utilização de árvores B+," anais da X Conferência Latino-Americana de Informática Valparaíso, Chile, 4/1984.

(2) — Vieira, Ewerton A. P. e Guimarães, Mario A.M. "MAV, MAI e MAP, Manual do Usuário V1.1" Depto. de Informática PUC/RJ.

(3) — Kernigham, B.W. e Ritchie, D., "The C Programming language" Prentice-Hall, 1978.

(4) — Date, C.J. "An Introduction to Data Base Systems". Addison-Wesley — Publishing Co. Mass., 1977, Second Edition.

(5) — Furtado, A. L. e Santos, C.S. "Organização de Banco de Dados", Editora Campus Ltda. RJ, 1982.

(6) — Melo, Rubens N. "Estruturas de Armazenamento de dados". CCE, Depto. de Informática PUC/RJ, 1981.

(7) — Wedekind, H. "On the selection of access paths in a data base system", Proc. IFIP Working Conference on Base Management, North Holland, 1974.

(8) — Comer, D., "The ubiquitous B-tree", ACM Computing Surveys, 11(2), 6/79.

(9) — Bayer, R. e McCreight, C., "Organization and maintenance of large ordered indexes", Acta Informatica, 1, 3/72, pp 173-189.

(10) — Menascé, D. e Landes)., "On the design of a reliable storage component for distributed data base management systems", Proc. Very Large Data Base Conference, Montreal, 1980.

(11) — Kruglinski, David "Data Base Management Systems" Osborne, McGraw-Hill 1983.

(12) — Stonebraker, Michael. "Retrospection on a Data-Base System". ACM transaction on Database Systems, 5/(2), 6/80.

(13) — Bayer, R., Unterauer, K., "Prefix B-trees" ACM transactions on Data Base Systems, 2(1), 3/77.

(14) — Vieira, Ewerton A.P., "Sistema de Controle de Estoque para Circuitos Integrados", Trabalho de Final de Curso em Engenharia Elétrica/Sistemas, PUC/RJ, 1983.

(15) — Menascê, D. e Landes, O., "Dynamic crash recovery of balanced trees", IEEE Computer Society Press, Symposium on reliability in distributed software, Pittsburgh, 7/81.

(16) — Guimarães, Mário A.M. "Especificação e Implementação de um Sistema de Pastas Eletrônicas", Dissertação de Mestrado, PUC/RJ, 04/84