# IMPLEMENTAÇÃO DE UM MÉTODO DE ACESSO BASEADO EM ESTRUTURAS DE ÁRVORES MULTIDIMENSIONAIS

Marcos Roberto da Silva Borges

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovado por:

Ysmar Vianna e Silva Filho
(presidente)

Luiz Antonio C. C. Couceiro

Jayme Luiz Szwarcfiter

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 1981

BORGES, MARCOS ROBERTO DA SILVA

Implementação de um Método de Acesso Baseado em Estruturas de Árvores Multidimensionais (Rio de Janeiro) 1981.

VIII, 122 p. 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas, 1981).

Tese - Universidade Federal do Rio de Janeiro - Faculdade de Engenharia.

1. Recuperação de Informações. 2. Pesquisa por chave secundária. I. COPPE/UFRJ. II. Título (série).

Aos meus pais

## AGRADECIMENTOS

A minha esposa Maria Lucia Neves Borges pelo incentivo e apoio.

Ao Professor Ysmar Vianna e Silva Filho pelo incentivo, orientação e excelentes sugestões prestadas no decorrer deste trabalho.

Aos demais membros da banca que me honraram com sua participação.

E aos colegas do NCE pelo apoio prestado.

## RESUMO

O presente trabalho descreve a experiência da implementação de um método de acesso baseado em estruturas de árvores multidimensionais.

Esta implementação teve por objetivo demonstrar a viabilidade do uso da estrutura para solução de problemas de recuperação por chave secundária, bem como a construção de um método de acesso que possa ser usado generalizadamente para resolver este problema.

Discute-se a adaptação dos modelos matemáticos às condições reais de uso propondo-se soluções adequadas aos problemas de implementação. Efetua-se testes com a estrutura de dados já montada, comparando os resultados alcançados com o esperado, de modo a validar os algoritmos utilizados na construção do sistema.

Finalmente é descrita a utilização do método de acesso para solução de um problema real de recuperação de informações.

## ABSTRACT

This work describes an experience of implementing a file acess method based on multidimensional binary search tree.

The objective of the work was to show the feasibility of using this tree structure for secondary key search, and also to program an acess method that could be used to answer these search requisites in general.

The data structure was adapted to the real conditions of usage and implementation details were worked out. Tests with the method were performed and comparison with predicted behaviour was also performed.

At the end an application of the software for a real information retrieval problem was done.

# ÍNDICE

Pág.

# CAPÍTULO I

# I N T R O D U Ç Ã O

Este trabalho apresenta fundamentalmente a descrição do processo de implementação de um método de acesso baseado em estruturas de árvores multidimensionais. Até o momento muita pesquisa tem sido feita no comportamento desta estrutura, mas pouca coisa tem sido relatada no que se refere a implementação efetiva da estrutura como solução para problemas de pesquisa por chave secundária em arquivos.

O objetivo principal deste trabalho é o de demonstrar a viabilidade da estrutura como método de acesso e transportar os resultados alcançados em estudos teóricos para o problema prático.

No Capítulo II do trabalho é descrito o problema da pesquisa por chave secundária classificando-o segundo diversas formas de apresentação encontradas na prática. É feito também um resumo do que já surgiu de estruturas de dados, até o momento, como solução do problema.

No Capítulo III é feita a análise detalhada da estrutura de árvore multidimensionais descrevendo a sua evolução desde que foi apresentado por Bentley[3] em 1975.

O Capítulo IV representa a parte fundamental deste trabalho, pois é neste Capítulo onde são descritos os passos necessários a implementação e as soluções adotadas para transportar os

os algoritmos e as soluções propostas nos estudos teóricos para o contexto de um problema prático, qual seja a efetiva implementação da estrutura como método de acesso para recuperação de informações.

No Capítulo V é descrita a experiência de implantação de um sistema de recuperação de informações sobre marcas registradas no país utilizando o método de acesso como meio para manter e recuperar as informações.

Finalmente no Capítulo VI são apresentadas as conclusões resultantes do trabalho de implementação.

# CAPÍTULO II

# PESQUISA POR CHAVE SECUNDÁRIA

## II.1) CONSIDERAÇÕES INICIAIS

Em um sistema de informações o componente mais importante, sob o ponto de vista interno, é o arquivo de dados por ser o meio de armazenamento das informações deste sistema, e ainda, possibilitar o inter-relacionamento com outros sistemas.

Em geral grande parte da programação de um sistema de informação resume-se a manter e recuperar informações dos arquivos, a fim de atender as solicitações dos usuários deste sistema. Desta maneira, é de fundamental importância para o projeto do sistema, a definição sobre o modo de armazenamento das informações a fim de facilitar ao máximo a manutenção dos dados e sua recuperação.

Um arquivo é composto de registros que são ocorrências dos dados do arquivo, cada um representando as informações dos elementos deste arquivo. O registro, por sua vez, é dividido em atributos que representam a informação como ela é entendida e aceita pelo usuário, sendo portanto a menor sub-divisão que pode ser feita ao nível do usuário do sistema.

Quando se deseja acessar um determinado registro em um arquivo é indispensável identificá-lo por um ou mais de seus atributos. Desta maneira, em geral são escolhidos atributos que pos

sam identificar o registro de dados através de uma relação biunívoca. O atributo escolhido para executar esta função identificadora é então denominado chave primária do arquivo. Este processo é anterior a automatização que tornou apenas mais rigorosa a escolha e o tratamento das chaves primárias.

A chave primária em geral é numérica, de pequena quantidade de dígitos e normalmente é um atributo novo adicionado ao conjunto de dados do conhecimento do usuário.

É por esta razão, que surgiram os chamados números de registro, número do CIC, número da identidade que são imposições dos sistemas computacionais ao conjunto de dados aceitos pelo usuário. Praticamente em todos os sistemas automatizados, são criadas chaves primárias de identificação do registro e não causa estranheza o fato de se considerar o número do CPF ou da conta bancária mais importante que o nome da pessoa para o Sistema de Informação.

A importância desta chave é tão grande que em geral são utilizados artifícios, como é o caso do dígito verificador, para garantir a validade desta chave antes de qualquer operação nos arquivos nos quais ela é o identificador do registro, pois a partir dela é que serão recuperados e/ou atualizados os demais atributos do registro.

Adicionando este novo elemento ao arquivo podemos defi

nir agora que um registro é composto por uma chave de identificação (chave primária) e os demais atributos de modo que, para recuperar os atributos de um determinado registro basta conhecer o conteúdo da chave primária deste registro.

O método de acesso mais simples utilizado para recuperar as informações de um registro no arquivo é o método de acesso sequencial, assim denominado por possibilitar a recuperação dos registros somente em ordem sequencial, ou seja, para recuperar o registro de ordem 500 é necessário percorrer os 499 registros anteriores.

Este método é largamente empregado quando a recuperação desejada é em larga escala, todos ou praticamente todos os registros do arquivo são recuperados, mas é grande a sua ineficácia quando o desejado é recuperar determinados registros no arquivo em ordem aleatória.

Com o aperfeiçoamento dos discos magnéticos surgiram os chamados métodos de acesso direto, assim denominados porque possibilitam a recuperação de um registro diretamente a partir do seu identificador. Alguns desses métodos, inclusive, acessam índices e mesmo outros registros antes de atingir o registro desejado, mas todo o percurso é controlado pelo método de acesso sendo o processo transparente para o usuário. Com a utilização destes métodos foi então possível a construção dos sistemas em tempo real, que em geral, apresentam como principal característica a recuperação imediata dos registros especificados pelo usuário através do terminal ligado ao computador.

O aumento da complexidade dos sistemas de informação, a partir das exigências dos usuários, introduziu a necessidade de recuperação das informações de um arquivo não mais somente através da chave primária e sim através da seleção de registros que possuíssem determinados valores para os demais atributos. As perguntas do tipo: Quais os dados do funcionário cujo número de registro é X? Continuaram sendo necessárias mas foram ficando cada vez mais importante para o usuário que o sistema pudesse fornecer resposta às perguntas do tipo: Quais os funcionários que trabalham no departamento "Y"? ou, Quais os funcionários que pertencem ao projeto "P" e possuem tal especialização? Para estes tipos de perguntas uma recuperação mais elaborada pela chave primária é inteiramente inútil, uma vez que será necessário percorrer todo o arquivo e a cada registro testar os atributos especificados, selecionando aqueles que se encontram dentro dos valores desejados. Este tipo de solução é largamente empregada quando o sistema funciona em "batch" e portanto, o tempo de resposta não é a preocupação maior do sistema. A saída em geral, é em listagens que serão posteriormente consultadas pelos usuários.

O aumento dos tamanhos dos arquivos e da necessidade deste tipo de recuperação, sobretudo em sistemas integrados de informações, além da exigência de uma maior rapidez nas respostas, influenciou a construção e implementação de novos métodos de acesso capazes de atender com maior eficiência este tipo de recuperação. Com o advento da filosofia de Banco de Dados foram lançados como parte dos softwares de Banco de Dados vários métodos de acesso com

o objetivo de se criar facilidades a este tipo de recuperação.

Um dos métodos mais utilizados das LISTAS INVERTIDAS que são índices de acesso aos registros do arquivo através de outros atributos, que não a chave primária. O processo é simples; escolhido o atributo sobre o qual se deseja "inverter" o arquivo, para cada valor deste atributo é associada uma lista de registros que possuem este valor, permitindo que todo o conjunto de registros deste arquivo seja referenciado por esta lista.

Sendo ambos, arquivo e lista, de acesso direto, é óbvio que através deste processo o número de registros acessados corresponde somente ao conjunto desejado como resposta a este tipo de recuperação.

Um exemplo simples onde este método é empregado, é o das listas telefônicas onde é possível recuperar o número de telefones a partir de dois meios; através do endereço ou através do nome do assinante. Outro exemplo bastante interessante onde este tipo de solução é largamente empregada, é no funcionamento de uma biblioteca, onde o título da obra pode ser considerado a chave primária do arquivo de obras e os catálogos de autor e assunto são na verdade, listas invertidas deste arquivo.

É seguro afirmar que as LISTAS INVERTIDAS são o método mais utilizado nos sistemas atuais para recuperação por chave secundária, assim denominada a recuperação por atributos que não a

chave primária do arquivo. Este e outros métodos, são discutidos mais detalhadamente no Capítulo III deste trabalho.

O problema da recuperação por chave secundária, pode ser então definido como sendo o de recuperar os registros a partir da especificação de um ou mais atributos, objetivando a minimizar o número de registros lidos que não atendam às especificações. Há o interesse também que isto se processe de uma maneira eficiente, ou seja, no menor tempo possível e com o mínimo de gasto de memória e espaço em disco para índices e ponteiros.

## II.2) CLASSIFICAÇÃO DA PESQUISA

Para melhor entendermos como se processa a especificação de uma pesquisa por chave secundária passaremos agora a descrever as características e variáveis que a pesquisa pode ter:

### a) Quanto ao pré-conhecimento sobre a recuperação desejada

Se for possível definir a priori, através do levantamento das necessidades do usuário, qual o tipo de recuperação desejado no sistema e quais as chaves a serem referenciadas em consultas, além da freqüência destas consultas, é possível definir-se o arquivo com características mais apropriadas a fim de facilitar estas recuperações. Em sistemas que utilizam arquivos sequenciais, êste conhecimento proporcionará a confecção dos programas específicos para atender a estas consultas. Quando se projeta um Banco de Dados, êste conhecimento é utilizado para definição de sets, sub-sets ou relações residentes para os atributos mais usados nas consultas a fim de otimizar o tempo de resposta para a maior parte das consultas.

Quando o desconhecimento sobre o tipo de consultas é total ou a variedade é muito grande, a solução é projetar-se um sistema de arquivos que, mesmo não sendo o mais eficiente para um tipo de consulta em particular, pode ser o mais eficiente em média, considerando todas as consultas possíveis de serem solicitadas.

b) Quanto a rapidez desejada na resposta

Em sistemas em que se deseja tempos curtos de resposta, em virtude da utilização de terminais para consulta e recebimento das informações, será necessário prover-se o sistema de métodos de recuperação que não necessitem percorrer todo o arquivo, principalmente se estes forem grandes, sendo considerados ideais aqueles que só acessam os registros que atendam as especificações da consulta. Desta maneira, o usual é a utilização de índices ou funções que através da especificação dos atributos desejados recuperam os registros que atendem a essas condições.

Para maior rapidez de resposta, em geral, é necessário uma maior utilização de espaço em disco e um maior custo de atualização dos arquivos.

Neste caso ainda é bastante relevante o fato do processamento em tempo real permitir atualizações nos arquivos, pois deste modo, além de eficiente na recuperação, o método deverá ser eficiente também nas atualizações dos registros.

c) Quanto ao tipo de resposta

O mais comum em S.I. é desejar-se como resposta a uma consulta, o conteúdo (todo ou parte dele) dos registros que satisfaçam as condições das perguntas, mas existem casos em que só se deseja conhecer a quantidade de registros que atendam às condições. Há ainda, situações em que o usuário, em princípio, deseja saber quantos registros e depois quais os registros.

Um exemplo interessante desta última situação é o de uma pesquisa bibliográfica em que, dependendo do número de referências apontadas, o usuário solicita do computador uma lista de registros ou faz nova consulta restringindo um pouco mais a gama de variação do atributo, repetindo o processo até alcançar um número de referências consideradas por ele razoável.

d) Quanto ao tipo de pergunta

Considerando o conjunto de atributos de um arquivo classificaremos os tipos de pergunta com base na classificação adotada por Bentley[3] e Rivest[30]. A primeira distinção que é feita, é entre as perguntas de interseção e os denominados "melhor escore". No primeiro caso, deseja-se como resposta registros, cujos a

tributos encontram-se dentro dos valores especificados nas perguntas e no segundo caso, registros cujos atributos aproximam-se mais das condições especificadas. Este segundo tipo é considerado a mais complexa pesquisa por chave secundária em um arquivo.

As consultas mais comuns em um Sistema de Informações são as de interseção que passaremos a classificar. Antes disso descreveremos um arquivo que será utilizado para exemplificar o tipo de pesquisa.

No arquivo abaixo a CHV-P é a chave primária e CHV-1 , CHV-2 e CHV-3 são as chaves secundárias.

| CHV-P | CHV-1 | CHV-2 | CHV-3 |
|---|---|---|---|
| 1 | 7 | A | AB |
| 2 | 5 | Z | AA |
| 3 | 7 | A | MA |
| 4 | 3 | A | MP |
| 5 | 18 | A | MA |
| 6 | 21 | B | AB |
| 7 | 15 | C | CF |
| 8 | 18 | C | MP |
| 9 | 7 | B | MP |
| 10 | 3 | E | MP |
| 11 | 22 | A | MA |
| 12 | 7 | B | MA |
| 13 | 7 | E | MA |
| 14 | 7 | C | CF |
| 15 | 15 | A | AB |
| 16 | 18 | C | AB |
| 17 | 7 | A | AB |

d.1) Perguntas de escore exato (EXACT MATCH QUERY)

É um tipo bastante comum de consulta em que todas as chaves secundárias são especificadas por valores discretos. Portanto, todos os atributos são relevantes na pesquisa e devem ser comparados com os valores especificados.

Exemplo: Quais os registros cuja CHV-1 = 7, CHV-2 = A e CHV-3 = AB?

Resposta: CHV-P = 1 e 17

d.2) Perguntas de escore parcial (PARTIAL MATCH QUERY)

Neste tipo de pergunta são especificados somente um subconjunto de atributos, menor que o conjunto de chaves secundárias. As chaves que não foram especificadas poderão possuir qualquer valor e são portanto, desconsideradas para os testes.

Exemplo: Quais os registros cuja CHV-2 = A e CHV-3 = AB?

Resposta: CHV-P = 1, 15 e 17

Observação: Note-se que a pergunta 1 tem como resposta um sub-conjunto das respostas da pergunta 2.

d.3) Perguntas por Região (REGION QUERY ou RANGE SEARCH)

Neste caso as chaves não são mais especificadas somente por valores discretos e sim por um intervalo. Não há sub-divisão para o caso de especificar todas ou parte das chaves. Observe

também que as perguntas por região são o tipo mais geral de perguntas de interseção considerando-se os dois anteriores, casos particulares deste.

Exemplo: Quais os registros cuja CHV-1 =[1,9] e CHV-2 =[A,D] ?

Resposta: CHV-P = 1, 3, 4, 9, 12, 14 e 17.

d.4) Perguntas por melhor escores (BEST MATCH QUERY)

Este tipo de pergunta é mais complexo de todos os apresentados até agora e corresponde a pesquisa do chamado vizinho mais próximo (NEAREST NEIGHBOR). Dado um conjunto de pontos cada um deles correspondendo a 1 registro de um arquivo e o ponto de referência da pergunta, deseja-se conhecer o ponto que mais se aproxima deste ponto de referência segundo uma função distância conhecida. A mesma interpretação serve para quando se deseja saber os M pontos mais próximos.

Exemplo: Para maior facilidade de entendimento utilizaremos somente as duas primeiras chaves, cuja representação gráfica do arquivo se encontra na figura II.1.

Se considerarmos a mesma escala para distâncias entre letras e números, a pergunta: Qual o ponto mais próximo de P1 (10,T)? A resposta será CHV-P = 2(5,Z), pois é o registro que se apresenta mais próximo de P.

A pergunta: Quais os três pontos mais próximos de P2 (17,K) terá como resposta CHV-P = 7 (15,C), 8 (18,C e 16(18,C).

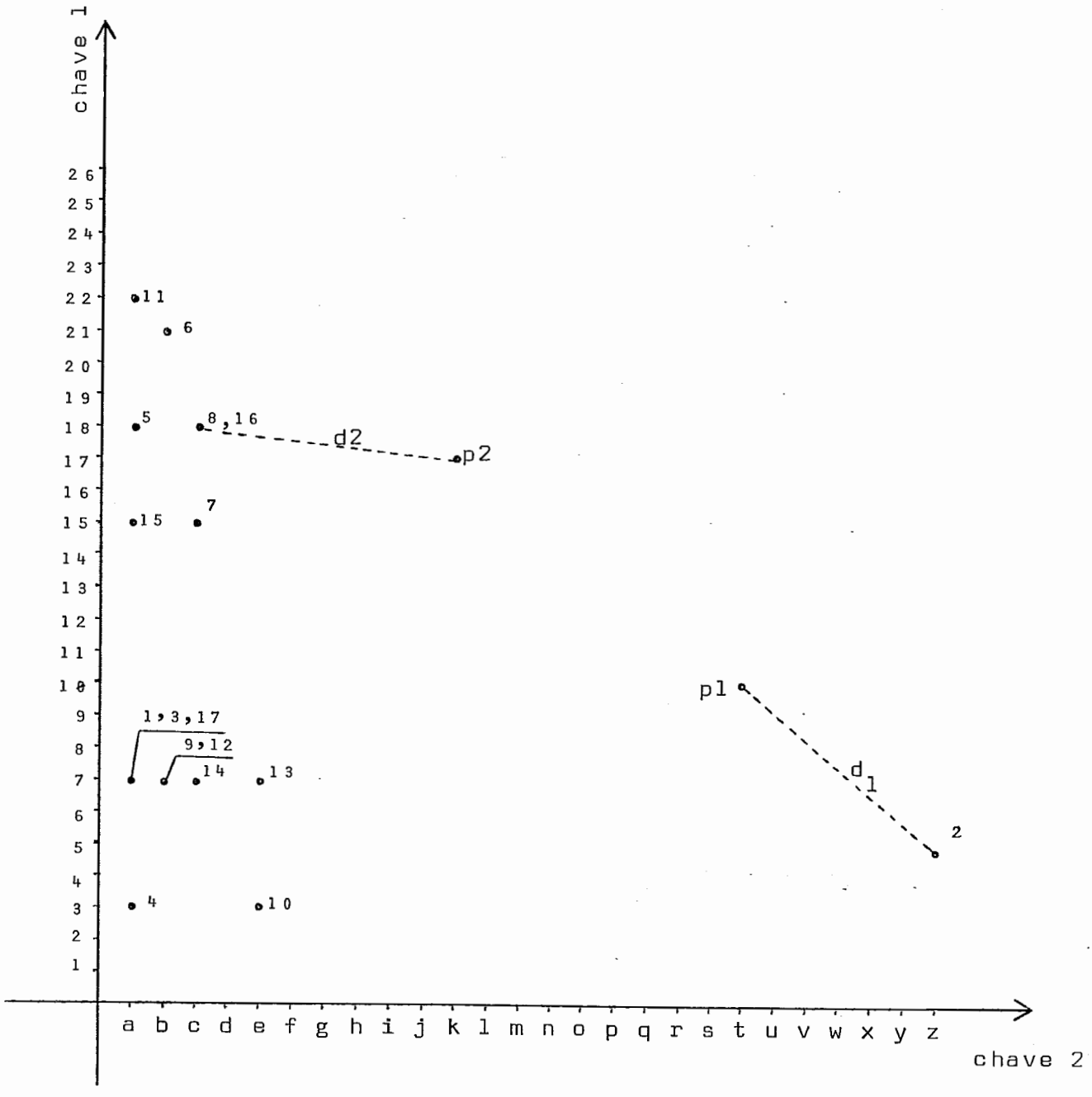



FIGURA II.1 - Pesquisa do melhor escore

e) Quanto ao conhecimento das características dos registros que possuam o arquivo.

Outro ponto de fundamental importância para a eficiência do método de acesso utilizado, é o conhecimento sobre o conteúdo das chaves secundárias pelas quais se deseja acessar o arquivo. Se não existe conhecimento prévio por parte do usuário quanto a este dado, é aconselhável fazer-se um levantamento a partir do arquivo de dados a fim de determinar os seguintes parâmetros:

a) VARIÂNCIA dos valores das chaves;

b) Grau de inter-relacionamento entre as chaves;

c) Distribuição dos valores das chaves pelos registros;

d) Freqüência com que as chaves são utilizadas nas perguntas.

Passaremos agora a detalhar o significado de cada parâmetro e sua influência na escolha do método de acesso.

e.1) VARIÂNCIA dos valores das chaves

Entende-se por VARIÂNCIA o conhecimento de quantos valores esta chave pode assumir para os registros do arquivo considerado. Este parâmetro pode assumir valores de 1 a N para os N registros do arquivo. Se a chave somente possui um valor, ela não deve ser considerada uma chave e sim uma constante. Este é o caso também de atributos a serem utilizados em futuras expansões do sistema ainda não implementados. Quando a Variância alcançar valores próximos de N, esta chave assemelha-se mais a chave primá

ria do que secundária. Neste último caso, o mais aconselhável é que esta chave não faça parte do índice e o teste da especificação da pergunta deverá ser direto contra o valor da chave nos registros selecionados por outras chaves, ou ainda utilizar outro índice semelhante ao utilizado para a chave primária quando esta chave for referenciada isoladamente.

e.2) Grau de Inter-Relacionamento entre as Chaves

Este parâmetro é bastante importante e representa a relação que pode existir entre os valores de duas chaves. Se esta relação for biunívoca, ou seja, para cada valor da primeira chave estão associados um ou mais valores da segunda chave e que estes valores da segunda chave só estão associados àquele da primeira chave, diz-se que existe uma relação hierárquica entre essas chaves e é possível associar-se as duas como sendo uma só, sem prejuízo da eficiência de recuperação.

e.3) Distribuição dos Valores das Chaves pelos Registros

O conhecimento sobre esta distribuição pode facilitar a busca e refere-se a distribuição dos registros pelos valores possíveis para esta chave. Por exemplo, se uma chave pode assumir dez valores diferentes, considere-se no primeiro caso 90% dos registros estão agrupados em um desses valores e os restantes 10% pelos outros valores e no segundo caso há uma distribuição dos valores uniformemente pelos registros (10% para cada valor). É bastante provável que as perguntas que referenciam esta chave recaiam

mais para especificar o valor que está associado a 90% dos registros no primeiro caso.

e.4) Freqüência com que as Chaves são Referenciadas nas Perguntas

Este parâmetro é semelhante ao pré-conhecimento sobre a recuperação desejada já referenciada na classificação das perguntas e representa o grau de importância de uma determinada chave para as consultas que serão feitas nos arquivos, ou seja, em que percentagem do total de perguntas esta chave é referenciada e como ela é referenciada.

Um último conhecimento desejável sobre o comportamento do arquivo é a sua dinâmica, pois o conhecimento deste parâmetro irá dirigir o estilo de recuperação a ser adotado. Se mais eficiente na recuperação em detrimento do tempo de atualização, para arquivos estáticos ou quase estáticos, ou se mais eficiente no processo de atualização para arquivos dinâmicos e com poucas consultas.

## II.3) ESTRUTURAS DE DADOS E ALGORITMOS DE BUSCA

Existe uma variedade tão grande de estruturas propostas para resolver o problema da pesquisa por chave secundária que uma tentativa de enumerá-las e descrevê-las uma a uma seria um problema de ordem tão grande quanto o da própria pesquisa. Por esta razão não é objetivo deste Capítulo apresentar uma análise exaustiva de todos os algoritmos de recuperação por chave secundária, pretende, isto sim, relatar de uma forma bastante suscinta somente aqueles considerados mais importantes para efeitos de comparação com a estrutura que serve de base para este trabalho. Desta forma, para melhor entendimento do comportamento das estruturas citadas é aconselhável a leitura das referências associadas.

Desde a análise de Knuth.[22] sobre recuperação por chave secundária, em que este considera o estudo do assunto bastante superficial, era de se esperar que uma grande quantidade de trabalho fosse dirigida para estudar o problema. Assim, em especial nos últimos anos, uma grande variedade de artigos relatando as idéias e os resultados alcançados na pesquisa do assunto tem surgido como conseqüência deste esforço.

Considerando-se o caso geral de recuperação, o algoritmo mais simples para recuperação de registros que satisfaçam as condições especificadas na consulta, é o da busca sequencial, ou seja, percorrer todos os registros do arquivo comparando os seus dados com os valores especificados na consulta. Este algoritmo tem a vantagem de, quando se trata de pesquisa por chave secundária, não introduzir nenhum custo adicional de espaço para armazenar o arquivo de dados. Entretanto o seu tempo de resposta é bastante elevado, especialmente quando a quantidade de registros no arquivo é grande.

Por outro lado existem estruturas que permitem recuperação bastante eficiente, mas introduzem um custo adicional de espaço e manutenção do arquivo que justificam o não aproveitamento da estrutura na maioria dos casos. Portanto é importante observar que a análise de uma estrutura de dados não deve ser feita somente pelo ângulo da eficiência da recuperação, mas também, do dusto adicional que ela provoca para armazenar e manter os índices auxiliares da estrutura.

Outro problema, este comum a todas as estruturas, é a escolha de quais das chaves secundárias do arquivo merecem ser referenciadas nos índices da estrutura para recuperação. Isto porque, podem existir chaves pouco frequentemente referenciadas não compensando o custo inerente a armazenagem e manutenção de índices para elas. Ou seja, quando a consulta referenciar uma destas chaves será necessário proceder a pesquisa sequencial dos registros.

A escolha destas chaves é bastante importante por influir fortemente na eficiência de recuperação das estruturas e, por esta razão, é grande a preocupação em procurar anternativas eficientes em função das particularidades de cada método de acesso. Bentley & Burkhard[5] propuseram uma heurística para escolha dos discriminantes na árvore multi-dimensional. Silva-Filho[34] apresentou um algoritmo que, baseado na probabilidade de ocorrência das chaves nas perguntas, determina os discriminantes de cada nível da árvore multi-dimensional. Anderson & Berra[2] fizeram uma análise detalhada da influência desta escolha na performance da estrutura seja ela qual for. Cárdenas[12] em sua análise sobre listas invertidas também propôs um método para escolha dos índices.

### II.3.1) Listas Invertidas

A estrutura denominada LISTA INVERTIDA é a mais difundida para utilização em sistemas de recuperação de informações devido a sua simplicidade e eficiência para determinados casos de recuperação por chave secundária. Além disso, esta estrutura proporciona outras vantagens ao sistema tais como a recuperação dos registros em ordem sequencial pela chave secundária que tem muita aplicação quando do tratamento dos arquivos em "batch".

A grande desvantagem desta estrutura se apresenta quando a pesquisa se dá por mais de uma chave e principalmente quando a consulta envolve combinações variadas das chaves secundárias do arquivo. No primeiro caso a desvantagem está na própria recupera

ção pois é necessário proceder à interseção das listas resultantes da recuperação dos registros em cada índice. No segundo a desvantagem está no fato de ser necessário se manter tantos índices quantos forem as chaves secundárias do arquivo possíveis de serem consultadas.

Cárdenas[12] apresenta uma análise bastante completa da estrutura demonstrando a sua aplicabilidade em Sistemas de Banco de Dados e explora bastante as características da estrutura em sua proposta de implementação. Hill[19] descreve mais detalhadamente a estrutura no que se refere a performance de recuperação e atualização. Croft[13] propõe uma solução para resolver um dos principais problemas da estrutura que se deve ao não grupamento, fisicamente no arquivo, dos registros que possuem as mesmas características para efeitos de recuperação por chave secundária. Desta forma, ele tenta diminuir a quantidade de acessos ao arquivo quando da recuperação efetiva dos registros.

Diversos outros artigos tratam da estrutura por considerá-la uma base para comparação com outras estruturas.

### II.3.2) Estruturas de Árvores

A estrutura da árvore tem sua eficiência comprovada na utilização de busca por uma chave e é largamente utilizado nos métodos de acesso de um modo geral. Diversas variantes da estrutura surgiram desde que Sussenguth[35] propôs a estrutura para processamento de arquivos. A utilização de estruturas de árvore pa

ra recuperação por chave secundária é mais recente mas tem demonstrado a sua eficiência através dos resultados alcançados.

A mais importante proposta para esta utilização surgiu em 1975, quando Bentley[3] apresentou a estrutura denominada árvore multidimensional.

Depois disso vários outros autores além do próprio Bentley[4] apresentaram trabalhos analisando o comportamento desta estrutura sobre os vários aspectos da sua utilização. Lee & Wong[23] analisaram o pior caso para consultas do tipo "Region Search", Silva Filho[33] analisou a performance média para o mesmo tipo de consulta. Friedman, Bentley e Finkel[17] sugeriram uma variante do algoritmo de busca para solução do problema de "Best-match" em tempo proporcional a logN; Bentley & Maurer[7] e Bentley & Friedman[6] fizeram uma análise comparativa de diversas estruturas concluíndo pela vantagem para as árvores multidimensionais para as consultas por região.

Outro estudo sobre uma alteração na estrutura e nos algoritmos de consulta e atualização foi proposta por Willard[38] para tornar a estrutura mais eficiente em sua dinâmica.

Alternativas de utilização em Banco de Dados e Sistemas de recuperação de informações são sugeridas por Bentley[4] e Silva Filho[32].

Esta estrutura apresenta como ponto forte a eficiência de recuperação generalizada para os diversos tipos de consulta e

o baixo custo adicional de espaço para armazenar a árvore índice. A desvantagem está no custo de geração e atualização e na dificuldade que a arrumação dos registros provoca na recuperação pela chave primária. A análise mais detalhada do comportamento da estrutura é apresentada no Capítulo III.

As chamadas "QUAD TREES" e "QUINTARY TREES" propostas respectivamente por Finkel & Bentley[14] e Lee & Wong[24] são também estruturas que utilizam árvores com boa performance de recuperação.

### II.3.3) Multi-Hashing e Indexação de Descritores

As estruturas de arquivos baseada em uma função de "Multi-hashing" e na indexação de descritores ou códigos superpostos, possuem a mesma filosofia de recuperação, qual seja, através da codificação dos valores das chaves alcançar os endereços dos registros que possuem a mesma codificação ou o mesmo resultado na função de "Multi-hashing". Obtidos estes registros procede-se a comparação das chaves destes registros com os valores especificados na consulta. Esta é uma forma bastante simplista de descrever o "Multi-hashing" e os códigos superpostos pois os mesmos envolvem considerações importantes que estão relatados nos artigos que tratam especificamente sobre o assunto.

Os resultados obtidos até agora neste tipo de estrutura são bastante interessantes e demonstram a importância da continui

dade de sua pesquisa. A quantidade de chaves deixa de ser um grande problema e a eficiência para consultas do tipo "partial-match" é próxima do ótimo. Além disso, necessitam de muito pouco espaço adicional.

A grande dificuldade está na complexidade da estrutura e na escolha de algoritmo eficiente para gerar os códigos das chaves em cada situação. Outra desvantagem surge se a grande quantidade de consulta se dá por uma chave, quando a performance da lista invertida é bem superior.

Aho & Ullman[1] propoem uma estrutura de acesso através de multi-hashing quando as chaves são independentes e procedem a comparação com as estruturas de listas invertidas e códigos superpostos. Neste mesmo artigo é proposto um algoritmo para escolha dos discriminantes semelhantes ao descrito por Silva Filho[34] para as árvores multidimensionais.

A estrutura baseada em códigos superpostos é apresentada de diferentes maneiras em três artigos. Vallarino[36] propõe a utilização de mapa de "bits" nas listas invertidas. Roberts[31] propõe a estrutura denominada de códigos superpostos e além de analisar a sua performance oferece um estudo do que poderia ser o tratamento do algoritmo de busca por "hardware". Mais recentemente Pfaltz, Berman & Cagley[29] apresentaram uma estrutura baseada nos mesmos conceitos dos códigos superpostos analisando a sua utilização em recuperação do tipo escore-parcial ("Partial-Match").

II.3.4) Outras Estruturas

Diversas estruturas, algumas de uso geral outras de finalidade específica para determinados tipos de consultas tem surgido também, destacando-se a combinação de índices propostos por Lum[26] ; a MAT (Multiple-Attribute-Tree) apresentada por Kashyap, Subas e Yao[21]; o diretório multi-dimensional (MDD) analisado por Liou & Yao[25] e árvores de duplo encadeamento proposta por Cardenas & Sagamang[11].

# CAPÍTULO III

# ÁRVORES MULTIDIMENSIONAIS

## III.1) APRESENTAÇÃO DA ESTRUTURA

A estrutura de árvores multidimensionais (árvore Kd) foi apresentada inicialmente por Bentley[3] em 1975. De lá para os dias de hoje muito estudo tem sido feito para analisar as características e o comportamento desta estrutura como solução para recuperação de arquivos por chave secundária. Willard[38] apresentou uma variante da estrutura para melhorar a sua dinâmica. Silva Filho[32] e Bentley[4] apresentaram propostas para utilização desta estrutura em Sistemas de Banco de Dados; Lee e Wong[24] apresentaram uma análise do comportamento da estrutura em procuras por Região (Region Query); Bentley & Friedman[6] e Bentley & Maurer[7] apresentaram estudos comparativos do comportamento de diversas estruturas, incluindo árvores Kd, frente aos tipos de recuperação por chave secundária. Além destes, diversos outros artigos estão sendo publicados analisando os diversos aspectos da estrutura, inclusive tratando do problema de implementação.

As árvores multidimensionais são árvores binárias, com a diferença de que a cada nó da árvore está associado um discriminate cujo valor irá decidir se o registro se posiciona à esquerda ou à direita deste nó. Estes discriminantes são escolhidos entre

as chaves secundárias do registro do arquivo. Inicialmente Bentley[3] propôs que a escolha dos discriminantes, para efeitos de simplificação da análise da estrutura, se desse da seguinte maneira: todos os nós do mesmo nível da árvore devem possuir o mesmo discriminante e a associação do discriminante ao nível é feita de forma cíclica. Por exemplo, se escolhermos a primeira chave como inicial para um arquivo de três chaves então o discriminante da raíz será a CHAVE-1, o do segundo nível a CHAVE-2, do terceiro a CHAVE-3, do quarto a CHAVE-1 e assim por diante.

Para o arquivo com duas chaves descrito na figura III.1, a representação gráfica da árvore gerada é apresentada na figura III.2. Pelo fato dos registros estarem associados ao nó , diz-se que a árvore é homogênea,pois neste caso o nó tem duas funções distintas: armazenar o registro e servir de índice para o arquivo . A árvore é considerada não homogênea quando o armazenamento do registro é feito separadamente do nó que serve de índice ou seja os registros estão armazenados em folhas de árvore compondo na realidade outro arquivo. Note-se que a representação não homogênea é mais apropriada quando a quantidade de registros é muito grande, sendo necessário o seu armazenamento em unidades de disco magnético, enquanto que a árvore índice, em geral pode ficar residente na memória do computador.

ARQUIVO

1 - (5,F)

2 - (7,D)

3 - (7,A)

4 - (2,C)

5 - (3,Z)

6 - (8,E)

7 - (3,A)

8 - (9,B)

9 - (6,C)

10 - (1,A)

11 - (3,F)

12 - (4,F)

13 - (6,B)

14 - (4,G)

15 - (1,E)

16 - (2,G)

Fig. III.1

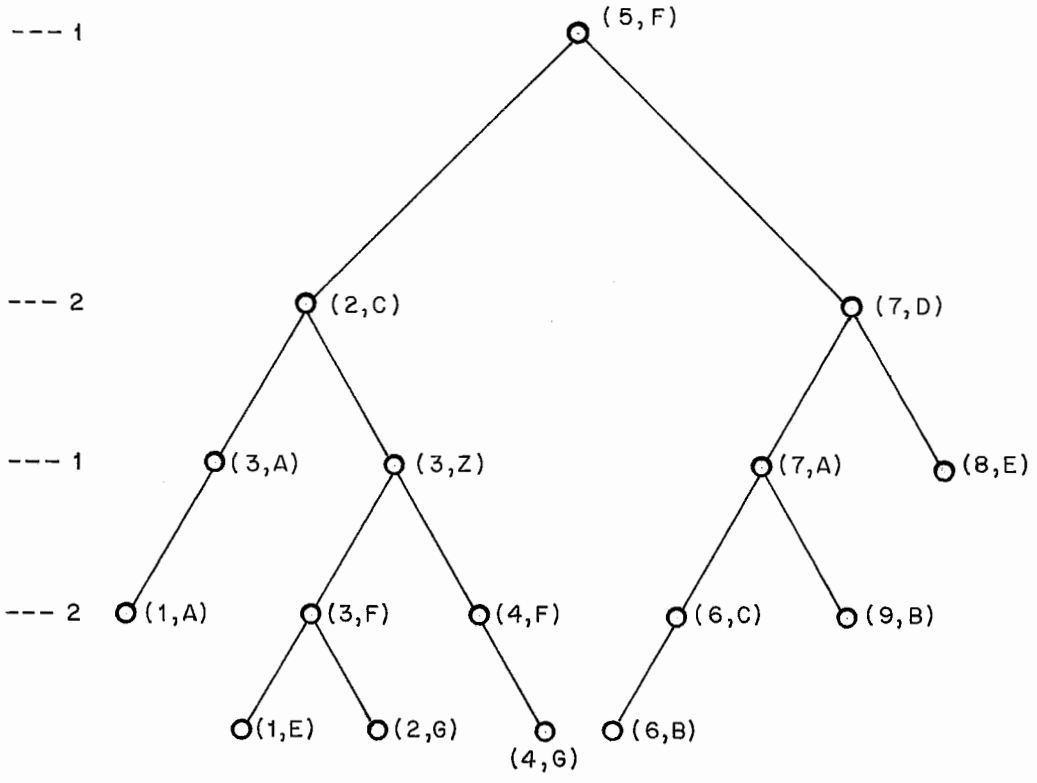



FIG. III-2 – ÁRVORE Kd HOMOGÊNEA PARA 2 CHAVES

Bentley[3] em seu artigo condicionou os resultados à restrição de que a árvore deva estar balanceada, ou seja, em cada nó a diferença entre a quantidade de níveis à esquerda e à direita é no máximo igual a 1. Vários artigos têm discutido o assunto e o que se pode concluir até agora é que: Em arquivos reais será muito difícil gerar uma árvore totalmente balanceada devido a quantidade de ocorrências de registros para um determinado valor da chave, ou seja, é provável que a mediana recaia sobre valores cuja ocorrência de registros provocará um desbalanceamento pois todos os registros com o valor da chave, igual ao da mediana, ficarão a esquerda do nó. Outra conclusão importante é que quando se trata de arquivos em disco, o que importa não é o balanceamento de registros lógicos e sim o de registros físicos (blocos) pois isto equaliza o número de acessos a disco para recuperação dos registros qualquer que seja a direção, mesmo que a quantidade de registros em cada bloco seja diferente. Isto deve-se ao fato de que o tempo gasto para tratar os registros é pequeno se comparado ao tempo de acesso e leitura dos registros no disco. A discussão deste fato é parte do próximo capítulo.

Para balancear-se a árvore será necessário que para cada nó seja calculada a mediana dos valores da chave discriminante deste nó. É possível adiantar que este cálculo irá ser o principal fator do custo de geração da árvore. Para exemplificar, é apresentado na figura III.4 o arquivo da figura III.1 em forma de árvore multidimensional não homogênea e balanceada. Na figura III.3 é demonstrado o processo de particionamento pela mediana.

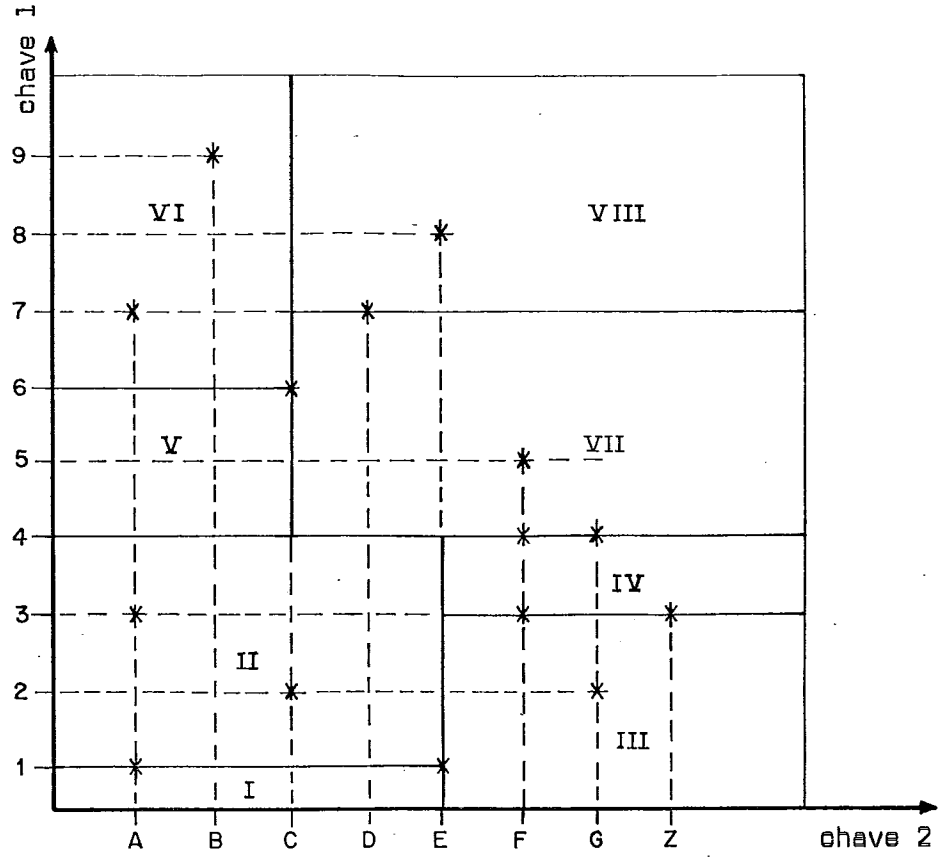



FIGURA III.3

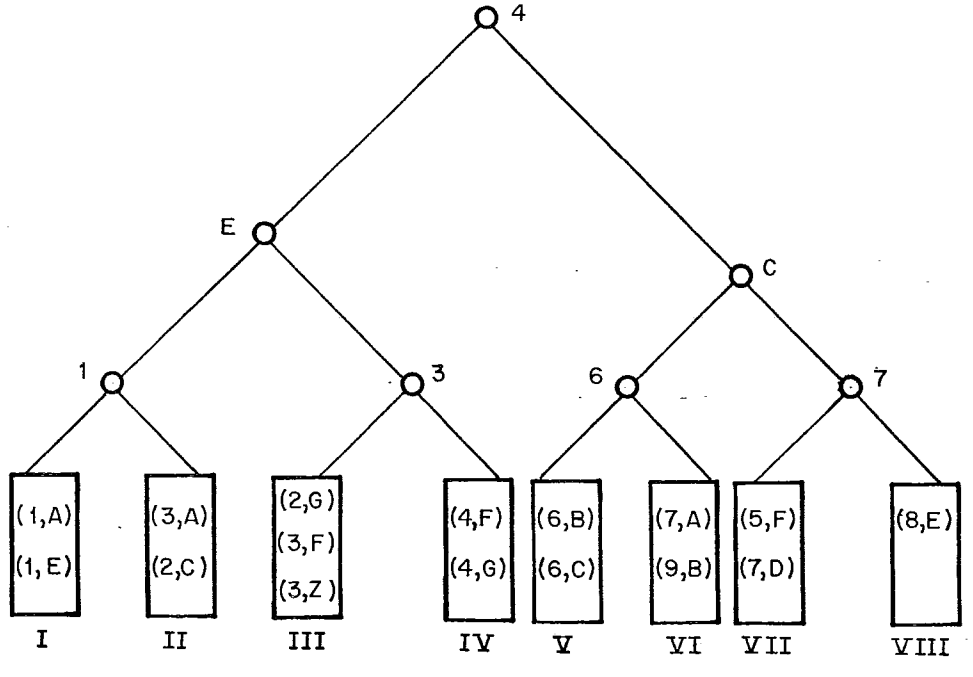



FIGURA III.4

## III.2) GERAÇÃO DA ÁRVORE ÍNDICE

A geração da árvore e a consequente arrumação dos registros de dados têm um custo muito alto e portanto deve ser minimizado o número de vezes em que será necessário gerar a árvore.

Esta geração se dá da seguinte forma:

1) Dado um conjunto de registros (arquivo), escolha uma das chaves como discriminante.

2) Ache o valor que corresponda a mediana dos valores da chave discriminante e considere este valor como discriminante da raíz.

3) Particione o arquivo considerando à esquerda os registros cujo valor da chave considerada é menor ou igual ao valor do discriminante da raíz, e à direita os registros com valores maiores.

4) Recursivamente construa a sub-árvore à esquerda e à direita usando os sub-conjuntos de registros resultantes da partição. Pare de repartir quando a quantidade de registros de um conjunto for menor ou igual ao tamanho do bloco especificado.

Ao final deste processo terá se obtido a árvore índice e os blocos apontados pelos nós desta árvore. A figura III.5 mostra a utilização deste processo no arquivo exemplo considerando o tamanho do bloco igual a três.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 - | (5,F) | (7,D) | (7,A) | (2,C) | (3,Z) | (8,E) | (3,A) | (9,B) | (6,C) | (1,A) | (3,F) | (4,F) | (6,B) | (4,G) | (1,E) | (2,G) |
| | | | | | | | | | 4 | | | | | | | |
| 1 - | (2,C) | (3,Z) | (3,A) | (1,A) | (3,F) | (4,F) | (4,G) | (1,E) | (2,G) | (6,B) | (6,C) | (9,B) | (8,E) | (7,A) | (7,D) | (5,F) |
| | | | | | E | | | | | | | | C | | | |
| 2 - | (2,C) | (3,A) | (1,A) | (1,E) | (2,G) | (4,G) | (4,F) | (3,F) | (3,Z) | (6,B) | (6,C) | (9,B) | (7,A) | (5,F) | (7,D) | (8,E) |
| | | | 1 | | | | 3 | | | | 6 | | | | 7 | |
| 3 - | (1,A) | (1,E) | (3,A) | (2,C) | (2,G) | (3,F) | (3,Z) | (4,F) | (4,G) | (6,B) | (6,C) | (7,A) | (9,B) | (5,F) | (7,D) | (8,E) |

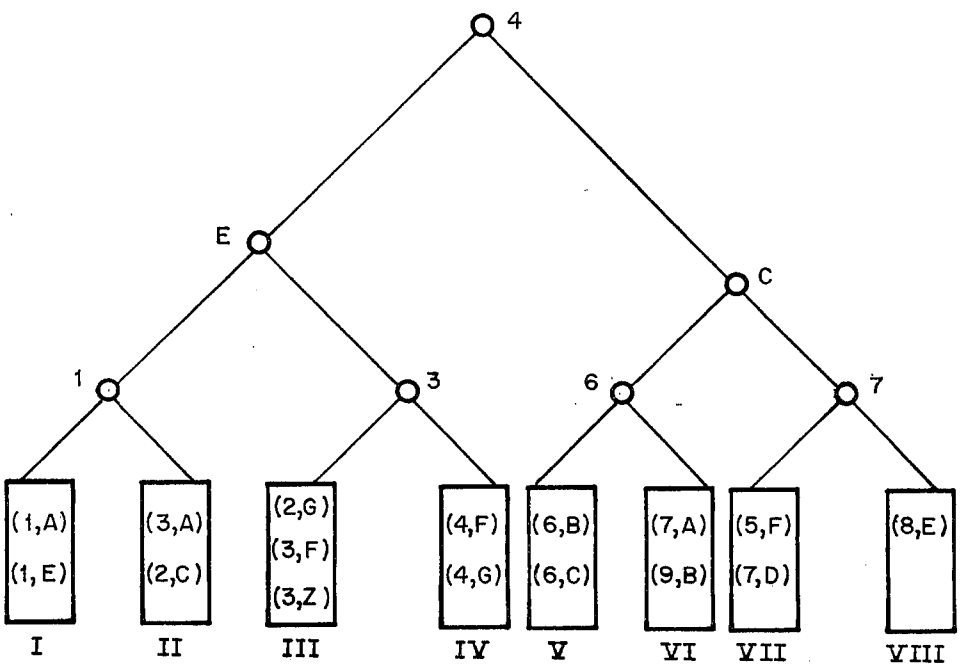



FIGURA III.5

O custo de geração da árvore é calculado levando-se em conta o número de registros de arquivo e o número de nós na árvore gerada. O número de nós é por sua vez, inversamente proporcional ao tamanho do bloco especificado, ou seja, quanto maior o tamanho do bloco menos nós terá a árvore índice e portanto menos partições será necessário efetuar no arquivo.

O custo de geração e o espaço necessário para a estrutura pode ser obtido através das fórmulas:

$$C(N, K) = O\left(N \log \frac{N}{b}\right) \qquad \text{III.1}$$

$$E(N, K) = O\left(K_1 N + K_2 \frac{N}{b}\right) \qquad \text{III.2}$$

Sendo N o número de registros do arquivo e b o número de registros no bloco. O espaço necessário para armazenar o índice é bem inferior ao do arquivo de dados e desta forma é possível até manter o índice na memória para arquivos pequenos ou parte do índice para arquivos grandes.

## III.3) PROCURA UTILIZANDO A ESTRUTURA

A procura de registros utilizando árvores multidimensionais pode ser resumida para os diversos tipos de perguntas (escore exato, escore parcial e região) em um só algoritmo geral: Começando na raíz, a árvore K-d é recursivamente percorrida da seguinte maneira:

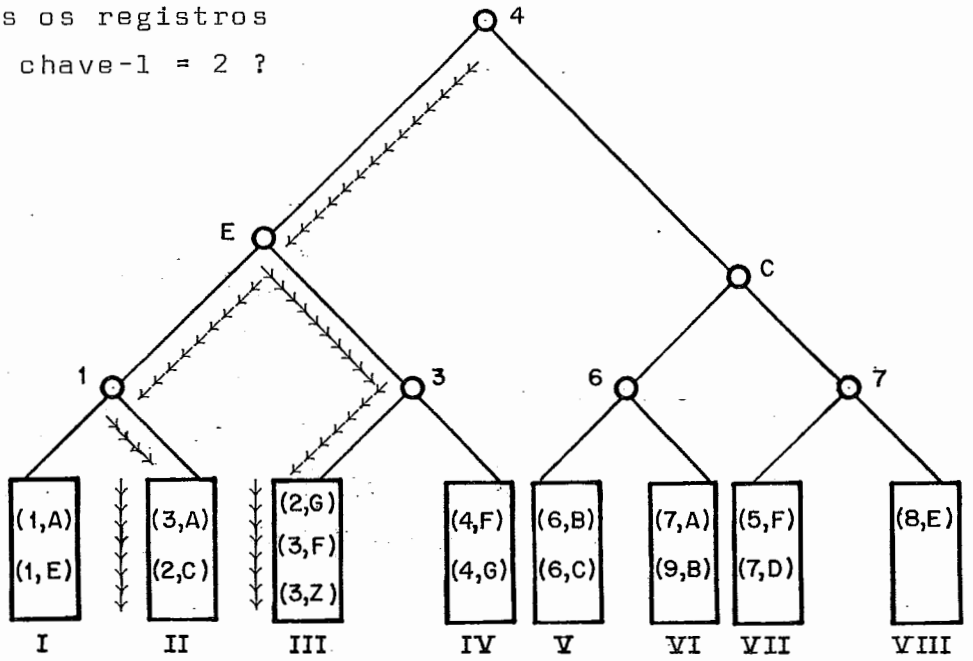
1) Quando em visita a um nó que discrimina a chave j, o algoritmo irá percorrer as duas sub-árvores, se a chave j não for especificada na pergunta ou se a região definida para esta chave (perguntas por região) sobrepôr-se ao valor do discriminante neste nó. A sub-árvore à esquerda será percorrida quando o valor ou o intervalo estiver abaixo do valor do discriminante neste nó, o inverso ocorrerá com a sub-árvore à direita.

2) Quando o nó considerado for um nó terminal (as sub-árvores apontadas são blocos de registros) uma busca sequencial será necessária comparando-se os valores das chaves especificadas nas perguntas com os registros dos blocos.

3) Os registros que satisfaçam as condições da pergunta são listados.

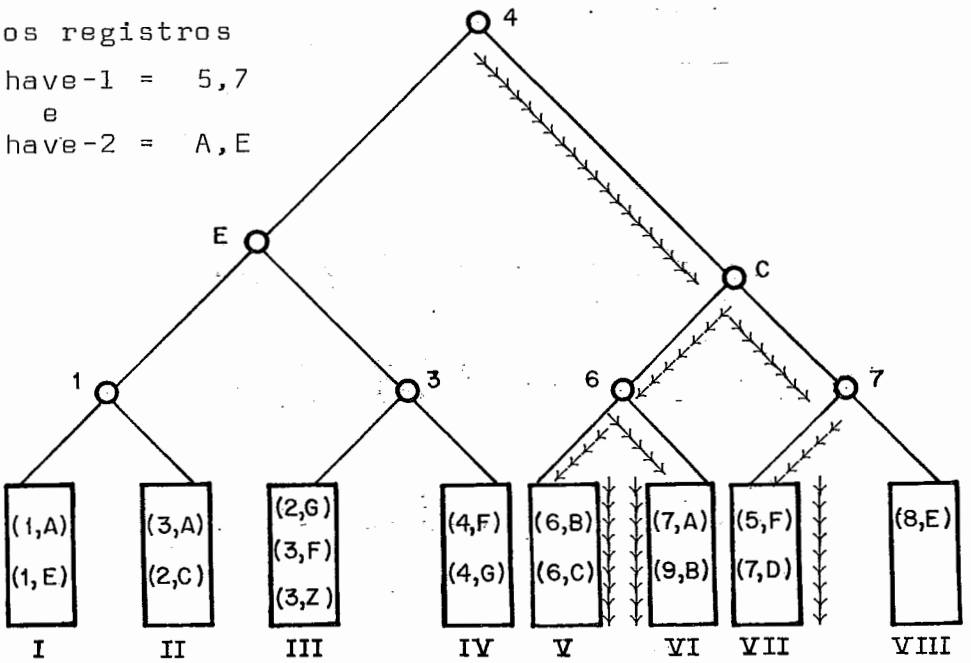
Para exemplificar o processo, são mostrados na figura III.6 dois exemplos de procura na árvore da fig. III.4.

P1) Quais os registros
cuja chave-1 = 2 ?




R1) (2,C) e (2,G)

P2) Quais os registros
cuja chave-1 = 5,7
e
chave-2 = A,E




R2) (6,B), (6,C), (7,A) e (7,D)

FIGURA III.6

O tempo médio gasto para perguntas do tipo escore parcial pode ser calculado utilizando-se a fórmula de Bentley adaptada as condições de geração da árvore.

$$Q\ (N,K) = O\ (N^{m/k}) \qquad \text{III.3}$$

sendo m o número de níveis cujos discriminantes não são especificados na pergunta e K o número total de níveis da estrutura. Já a quantidade de blocos de registros acessados é igual a $2^m$. Como exemplo, se 3 das 6 chaves de um arquivo de 1.000.000 registros for especificado somente 1.000 registros serão pesquisados.

Para as perguntas do tipo região Lee e Wong[24] mostraram que o pior caso pode ser calculado pela fórmula:

$$Q\ (N,K) = O\ (N^{1 - 1/k} + F) \qquad \text{III.4}$$

sendo F o número de registros que satisfazem a pergunta.

O valor médio foi calculado por Silva Filho[33] e mostra que a fórmula abaixo pode ser usada se o número de registros que satisfazem a pergunta é pequeno.

$$Q\ (N,K) = O\ (\log N + F) \qquad \text{III.5}$$

Para o caso em que uma grande parte do arquivo satisfaça a pergunta, a fórmula abaixo é aplicável:

$$Q\ (N,K) = O\ (F) \qquad \text{III.6}$$

O número de blocos acessados irá depender de como os intervalos especificados sobrepõem-se aos nós e será no mínimo igual a $2^m$ e no máximo igual $2^k$.

## III.4) ATUALIZAÇÃO DO ARQUIVO DE DADOS

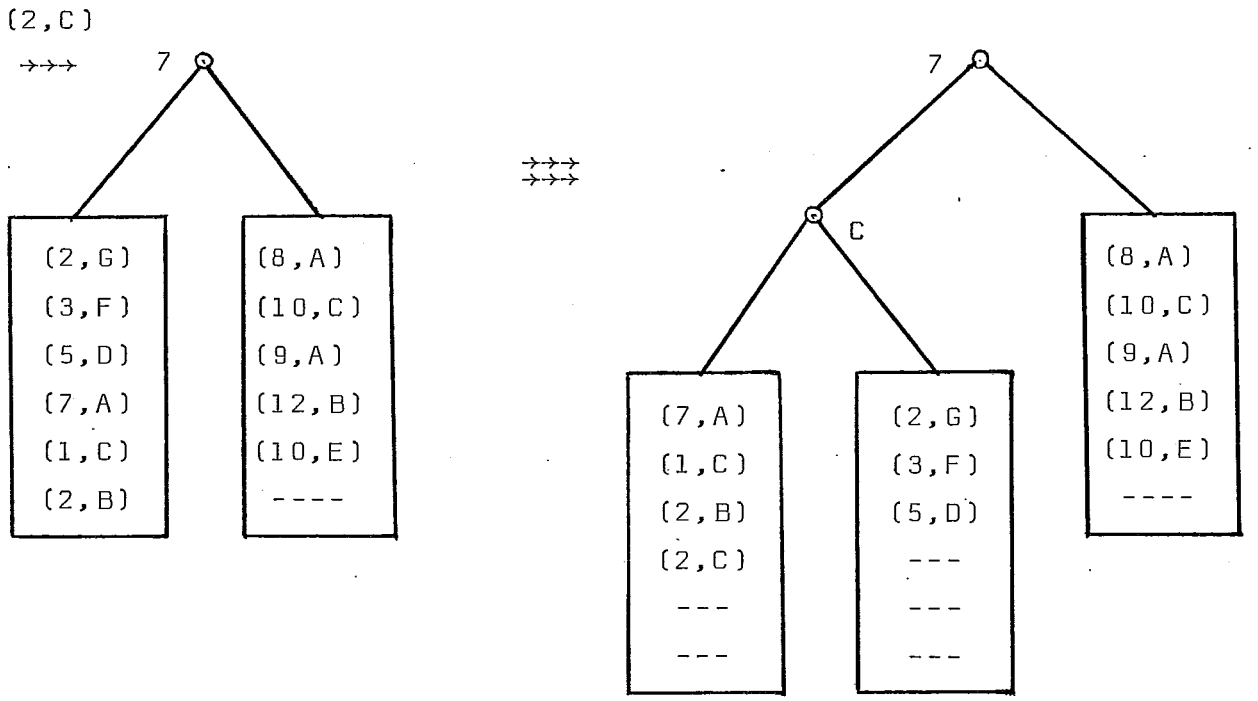
A atualização do arquivo é realizada de maneira extremamente simples bastando acrescentar mais um registro no bloco no caso de inclusão ou simplesmente excluir o registro no caso de exclusão. A dificuldade se apresenta quando o bloco está cheio ou fica completamente vazio após respectivamente, a inclusão ou exclusão de um registro.

No caso de bloco cheio a próxima inclusão no mesmo bloco irá ocasionar a geração de mais um bloco ("split") e ou a rearrumação dos registros nos blocos, se o bloco irmão, apontado pelo mesmo nó, contiver espaços vazios. No primeiro caso será necessário calcular a mediana dos registros do bloco com relação ao discriminante correspondente e utilizar este valor no novo nó gerado pelo particionamento. No segundo caso será necessário recalcular a mediana levando-se em conta os registros dos dois blocos tentando-se redividir os registros pelos dois blocos, neste caso, não será necessário a alteração da estrutura da árvore. A figura III.7 ilustra os dois procedimentos.

A exclusão de um registro pode se complicar quando esta torna um determinado bloco completamente vazio. Quando isto acontece, pode-se simplesmente eliminar o apontador para este bloco no nó correspondente ou então rearrumar os blocos em uma situação semelhante ao segundo caso da inclusão.

O processo de rearrumação da árvore pode, dependendo do caso, alcançar até a raíz quando então o custo de rearrumação da estrutura será idêntico ao da geração da árvore.

1º caso:




2º caso:

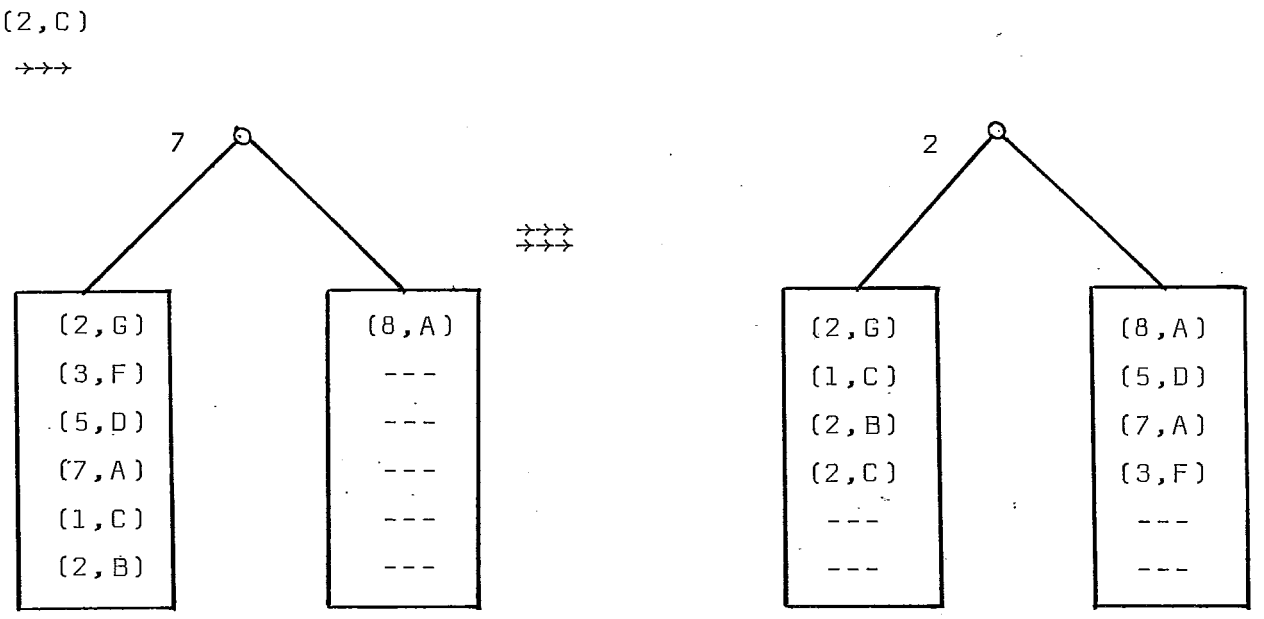



FIGURA III.7

## III.5) ANÁLISE DA ESTRUTURA

Como foi dito inicialmente, os estudos sobre a estrutura são muito recentes e muita coisa precisa ser feita a fim de aprimorar a estrutura com vistas a sua efetiva implementação para solução de problemas de recuperação por chave secundária.

Um destes aspectos é justamente a escolha dos discriminantes tendo em vista que a quantidade de blocos percorridos é diretamente proporcional a quantidade de níveis cujos discriminantes não foram especificados na pergunta. Desta forma, se houver algum conhecimento prévio sobre as probabilidades de ocorrências das chaves nas perguntas, isto deve ser usado para dirigir a escolha dos discriminantes em cada nível da árvore. Silva-Filho[34] recentemente publicou um estudo que apresenta um algoritmo para o cálculo do número de vezes que um discriminante deve ser utilizado na árvore. Isto visa minimizar o número de blocos acessados pela maioria das perguntas.

A ordem que estes discriminantes são usados é importante para minimizar a quantidade de nós percorridos, de forma que as chaves mais citadas nas perguntas sejam os discriminantes dos níveis iniciais, entretanto, este fatòr é menos importante do que a quantidade de vezes que a chave é usada, como veremos no próximo capítulo.

O conhecimento sobre os registros ajuda também a ajustar a estrutura de modo a capacitá-la a uma melhor performance. A Variância dos valores das chaves, e a distribuição dos registros

nestes valores citados no Capítulo I, pode alterar a escolha da ordem e do número de vezes, pois de nada adianta, por exemplo, utilizar um discriminante em dois níveis se este discriminante só possui três valores, ou seja, será impossível particionar os registros em quatro blocos. Outro problema acontece quando a distribuição dos registros pelos valores é muito desbalanceada provocando um desbalanceamento muito grande na árvore, o que pode não ser interessante.

Todos estes aspectos serão mais discutidos no próximo Capítulo que trata da experiência de implementação da estrutura.

## III.6) APLICAÇÕES

Esta estrutura apresenta soluções muito boas e até excelentes para os problemas relatados no Capítulo II como comprovam os resultados alcançados por Bentley[3] , Bentley & Maurer[7] , Bentley & Friedman[6] , Lee & Wong[24] e Silva Filho[33] . A aplicação desta estrutura em Sistemas de Banco de Dados é citada em Bentely[4] e Silva Filho[32] . Estas aplicações são na área de recuperação de informações que é uma área com um campo muito aberto à exploração, como afirmou Knuth[22] .

A utilização mais eficiente da estrutura se dá quando o problema apresenta uma variação grande de recuperação de dados, um grande volume de registros a serem tratados e uma necessidade

de resposta mais imediata através de terminais de vídeo. Um destes problemas citados por Knuth[22] e Bentley[4] é o de um sistema de informações geográficos onde se aplicam os problemas de escore exato, escore parcial, procuras por região e até o problema de melhor escore (Best Match).

Um problema bastante interessante onde a estrutura de dados pode ser usada com muito sucesso é o sistema de recuperação de informações sobre recursos geográficos, semelhante ao desenvolvido pelo projeto RADAM . Este sistema tem a vantagem de ser praticamente estático o que aumenta a eficiência da estrutura de árvores K-d.

O problema em que o método de acesso desenvolvido foi experimentado é tipicamente um problema de recuperação de informações por chave secundária, cujas características se ajustam perfeitamente ao que a estrutura proposta pode apresentar de eficiência no tratamento das informações. O problema da busca de colidências, relatado no Capítulo V poderia ser ampliado se o tratamento das colidências fosse por melhor escore ao invés de escore exato.

Estas e outras aplicações servem perfeitamente para justificar a experiência realizada por este trabalho em virtude dos resultados alcançados até agora.

# CAPÍTULO IV

# DESCRIÇÃO DA EXPERIÊNCIA DA IMPLEMENTAÇÃO

## IV.1) INTRODUÇÃO

Neste capítulo é descrito a experiência de implementação do método de acesso baseado na estrutura de árvores multidimensionais. São apresentados os programas e rotinas que compõem o método de acesso e a descrição detalhada das soluções adotadas para os problemas que surgiram no decorrer deste trabalho.

A maior dificuldade encontrada foi a adaptação dos algoritmos às condições reais, pois o fato dos dados estarem armazenados em disco transforma bastante o problema, provocando situações que não são previstas nos algoritmos que acompanham a proposta inicial da estrutura formulada por Bentley[3].

Para melhor compreensão o capítulo está dividido em três partes em que são relatados os programas de geração da estrutura de dados, consulta aos dados utilizando a estrutura gerada e atualização dos registros e dos índices que compõem o método de acesso.

## IV.2) GERAÇÃO DA ÁRVORE ÍNDICE

### IV.2.1) Descrição do Processo

A geração da árvore é a parte mais importante do método; pois as soluções e resultados desta fase irão influir enormemente no desempenho das funções de consulta e atualização.

A geração consiste em particionar o arquivo de dados de acordo com a mediana dos valores da chave discriminante e para cada uma destas células repetir o processo até que todos os blocos possuam uma quantidade de registros menor ou igual ao valor escolhido como tamanho do bloco do arquivo de dados. As medianas são armazenadas em registros do arquivo árvore que será o índice do arquivo de dados.

Por esta definição já é possível adiantar os dois grandes problemas que devem ser resolvidos para geração da estrutura. São eles: Como particionar eficientemente o arquivo de dados? Como escolher o tamanho de bloco adequado à taxa de ocupação desejada no arquivo de dados?

O problema de particionamento é descrito em detalhes mais adiante, mas é interessante relatar dois aspectos que diferem bastante nas soluções apresentadas nos algoritmos. O primeiro aspecto refere-se ao fato de que em arquivos reais dificilmente se conseguirá um particionamento que mantenha a árvore perfeitamente balanceada, que é o suposto da maioria dos algoritmos. Por

se tratar de chaves secundárias, é esperado que a freqüência de registros para cada valor da chave seja grande. Neste caso, é provável que ao se calcular a mediana e particionando os registros maiores e não maiores do que este valor, o resultado será uma quantidade diferente de registros à esquerda e à direita, isto porque o cálculo supõe que registros que possuam o valor da mediana como chave possam ficar parte à direita e parte à esquerda proporcionando o balanceamento, mas quando se particiona é necessário posicioná-los só à direita, ou só à esquerda. Isto irá ocasionar o desbalanceamento que, dependendo da situação, poderá alcançar índices altos. É interessante observar que podem haver situações em que seja mais vantajoso escolher um valor diferente da mediana a fim de provocar um desbalanceamento menor do que a mediana provocaria como no exemplo a seguir:

Suponha a seguinte distribuição de registros pelas chaves:

| VALOR | QUANT. DE REG. |
|---|---|
| 1 | 20 |
| 2 | 25 |
| 3 | 30 |
| 4 | 15 |
| 5 | 10 |

A mediana das chaves é o valor 3 e o particionamento neste valor representaria 75 registros a esquerda e 25 a direita se convencionarmos que os valores iguais se posicionam à esquerda.

Entretanto, se particionarmos no valor 2 o resultado será 45 registros à esquerda e 55 à direita, o que em termos de balanceamento, é melhor do que a situação anterior. Este tipo de procedimento é o adotado no algoritmo de particionamento do arquivo, mas mesmo assim não evitará o desbalanceamento, principalmente nós que usam como discriminante chaves com grande freqüência de registros. Entretanto, o balanceamento a nível dos blocos de dados e de nós da árvore poderá ser alcançado, o que é uma grande vantagem, pois equaliza a quantidade de acessos ao disco qualquer que seja o caminho percorrido até o registro de dados.

O segundo problema citado é quanto a escolha do discriminante em cada nível da árvore. Este problema na realidade deve ser sub-dividido em dois problemas que se referem a quantidade de níveis em que a chave é usada como discriminante e a ordem em que esses discriminantes são usados nos níveis da árvore. O primeiro destes problemas irá afetar somente a quantidade de blocos de dados acessados e o segundo irá influenciar o número de nós visitados na árvore, sem afetar a quantidade de blocos acessados. O algoritmo de geração não irá cuidar diretamente deste problema e apenas prevê que o usuário escolha livremente o discriminante em cada nível, além de oferecer como subsídio para análise da escolha o resultado do particionamento do arquivo segundo este discriminante.

Para determinação do número de vezes em que a chave deve ser usada como discriminante, deve-se levar em conta três fatores;

a) A probabilidade desta chave aparecer nas consultas;

b) A distribuição dos registros pelos valores das chaves;

c) A quantidade de ocorrências de valores para esta chave.

Para o primeiro fator é óbvia a constatação de que quanto maior a probabilidade da chave aparecer nas consultas, maior o número de vezes que ela deverá ser usada como discriminante.

O segundo fator refere-se ao resultado que o particionamento, segundo a mediana deste discriminante, irá causar no balanceamento da árvore. Como exemplo citamos a seguinte situação: Para uma determinada chave a ser utilizada como discriminante observa-se que após o particionamento segunda a mediana o desbalanceamento é de 80% porque a freqüência de registros no valor da mediana é muito alta. Neste caso, é preferível a não utilização desta chave como discriminante em níveis superiores da árvore.

O terceiro fator estabelece uma limitação para o particionamento por uma chave. É fácil constatar-se que o número de vezes em que o arquivo pode ser particionado por uma chave deve ser $\leq \log \lambda$, sendo $\lambda$ o número de valores diferentes que esta chave pode tomar, ou seja, será inutil particionar um arquivo duas vezes na mesma chave se esta chave só possui três valores diferentes. Este fato irá influenciar o primeiro fator de modo que após o cálculo, os valores deverão ser confrontados com o número de ocorrências e caso haja esta situação, o número de vezes deve ser recalculado.

A escolha do nível em que a chave deve ser utilizada como discriminante é função somente da probabilidade desta chave aparecer na pergunta, entretanto, da mesma forma que no primeiro fator o desbalanceamento proporcionado por esta chave pode alterar esta escolha, pois utilizando esta chave em níveis inferiores, o desbalanceamento ficará mais diluido do que na utilização em níveis superiores.

Silva-Filho[34] apresenta de forma bastante completa a solução deste problema da escolha do discriminante para árvores multidimensionais.

O terceiro grande problema citado, refere-se à escolha do tamanho do bloco do arquivo de dados. Como veremos mais adiante, este parâmetro tem grande influência no processo de atualização de modo a minimizar "splits" e rearrumações. Assim, este valor não deve ser escolhido de forma aleatória, mas mesmo que isso ocorra, não se irá alcançar taxas de ocupação média inferiores a 50%, pois o algoritmo utilizado não particiona mais uma célula que alcançou o tamanho do bloco especificado, independente do nível da árvore em que se encontre.

Se a árvore fosse perfeitamente balanceada a escolha do tamanho do bloco seria bastante simples, bastando usar a fórmula:

$$b = \frac{N}{2^n} * p$$

sendo N o número de registros, n o número de níveis da árvore desejados e p a taxa de ocupação desejada.

No caso real em que as células são desbalanceadas será impossível adotar-se um modelo matemático para determinar o tamanho do bloco. O mesmo deverá ser determinado de forma empírica e para auxiliar na escolha foi construido um programa auxiliar que percorre os registros da árvore e determina com um certo grau de aproximação qual a taxa de ocupação esperada. O algoritmo utilizado para este cálculo encontra-se no apêndice I.

IV.2.2) Descrição dos Programas

A geração da estrutura e arrumação do arquivo de dados é desempenhada por dois programas principais, um deles com duas opções de processamento. Para gerar dados que auxiliem na escolha dos parâmetros de geração existem ainda dois programas auxiliares. Todos os programas foram escritos em COBOL, pela facilidade que a linguagem proporciona no tratamento de arquivos além da grande vantagem da portabilidade para outros computadores.

O primeiro destes programas é o de geração propriamente dita, que a partir do arquivo de dados e baseado nos parâmetros de controle, procede o particionamento gerando a árvore índice e um outro arquivo de dados organizado em blocos de registros que são apontados pelos nós da árvore. O segundo programa tem a função de reorganizar a árvore e arrumar os blocos de registros de dados a partir do último particionamento executado pelo primeiro programa. Após este processo de arrumação, os dados estão prontos para serem acessados, via árvore índice, pelos programas de consulta e atualização.

Os programas auxiliares auxiliam o processo de geração na medida em que fornecem dados para uma melhor "performance" destes programas e do método como um todo. O primeiro deles percorre o arquivo de entrada e fornece dados sobre as chaves, tais como a freqüência de registros em cada valor da chave e a quantidade de valores diferentes que a chave possui. O segundo programa auxiliar determina a taxa de ocupação estimada para determinados valores de blocos e é usado no final do processo de geração da estrutura.

Para maior facilidade de compreensão de como funciona o programa de geração propriamente dito a explicação está dividida em quatro partes a saber:

a) O processo de particionamento;

b) O algoritmo de cálculo das medianas;

c) A leitura e gravação do arquivo de dados;

d) A leitura e gravação do arquivo índice (árvore).

a) O processo de particionamento é efetuado em n + 1 passos sendo n o número de níveis da árvore. No primeiro e no último passo o processo se dá de maneira diferente. Nos demais passos o processo é idêntico.

No primeiro passo, o processo consiste em ler o arquivo e calcular a mediana dos registros para o discriminante escolhido para a raíz da árvore. Nos passos intermediários o arquivo é particionado em sub-arquivos segundo as medianas calculadas no

passo anterior e ao mesmo tempo é calculada a mediana para cada um dos sub-arquivos resultantes do particionamento. Na última passada não se efetua qualquer cálculo da mediana, o programa limita-se a particionar o arquivo em blocos.

A única excessão ao processo de particionamento intermediário se dá quando o número de registros de um sub-arquivo atinge a quantidade especificada como sendo o tamanho do bloco. O algoritmo então não mais particiona este bloco, bem como não efetua os cálculos de medianas correspondentes, limitando-se a gravar os registros deste bloco no arquivo de saída. O fluxograma básico deste processo é apresentado na figura IV.1.

b) O algoritmo de cálculo das medianas adotado teve por objetivo não somente a otimização do cálculo propriamente dito mas principalmente a minimização da quantidade de leituras e gravações no arquivo de dados. Desta maneira, o algoritmo escolhido para o cálculo da mediana de Floyd & Rivest[15] foi combinado com o algoritmo proposto por Weide[37] para o cálculo aproximado da mediana. O algoritmo de Floyd & Rivest[15] tem como principal vantagem o menor número de comparações. Já o algoritmo de Weide[37] garante que o cálculo da mediana se dará em uma só passada no arquivo de dados.

O algoritmo de Floyd & Rivest[15] pode ser mais facilmente explicado se simplificarmos o processo da seguinte maneira:

1) Inicializa-se as variáveis POS-ESQ, POS-DIR e m no valor $\frac{N}{2}$ sendo N o total de registros.

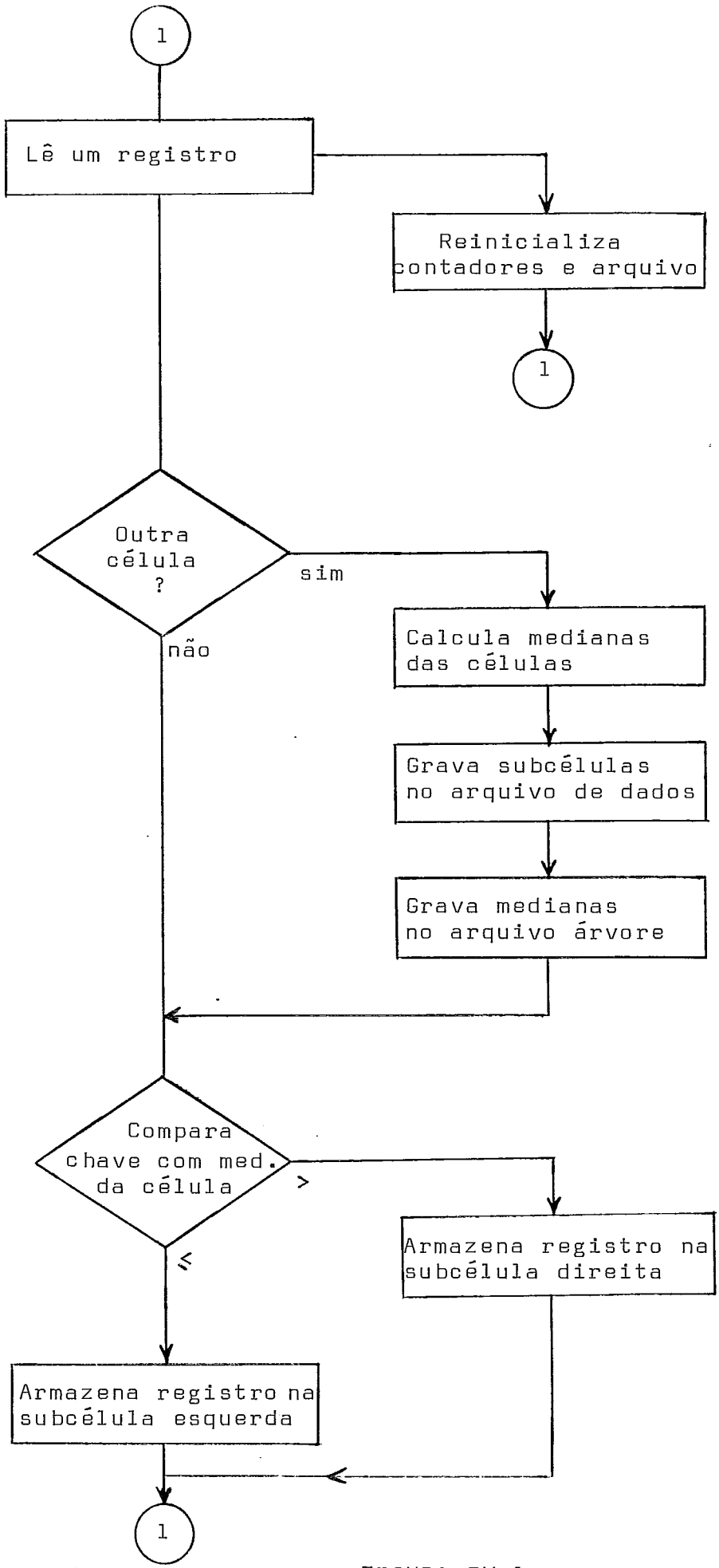

1
Lê um registro
Reinicializa contadores e arquivo
1
Outra célula ?
sim
não
Calcula medianas das células
Grava subcélulas no arquivo de dados
Grava medianas no arquivo árvore
Compara chave com med. da célula
>
<=
Armazena registro na subcélula direita
Armazena registro na subcélula esquerda
1

FIGURA IV.1

2) Seleciona-se do arquivo uma amostra aleatória de registros e para esta amostra calcula-se, utilizando o algoritmo, o registro que se encontra m-ésima posição.

3) Ler o arquivo separando os registros menores e iguais que o valor calculado no passo anterior dos registros maiores. Se a quantidade de registros de cada parte (maiores e não maiores) corresponderem aos valores de POS-DIR e POS-ESQ então o valor usado corresponde a mediana do arquivo. Em caso contrário subtrai se do valor esperado correspondente (POS-ESQ ou POS-DIR) a quantidade de registros que ficou abaixo do esperado.

4) Repetir o processo a partir do passo 2 considerando somente os registros que se posicionaram na parte cuja quantidade foi maior que a esperada. Usar para os próximos passos o valor de m igual a POS-ESQ.

O algoritmo consegue a cada ciclo diminuir o universo de registros a serem tratados de tal modo a aumentar a probabilidade de amostra refletir a situação dos registros no arquivo. Portanto, o algoritmo sempre converge para o valor da mediana.

Para diminuir o tamanho do universo de registros a ser tratado, o que o algoritmo faz é escolher um intervalo de modo que no passo seguinte três situações podem ocorrer: A mediana está na porção inferior, na porção superior ou no intervalo. Desta maneira duas porções do arquivo são eliminadas. O objetivo é que o valor procurado esteja no intervalo, pois desta maneira a quantidade de registros a serem tratados será minimizada. Os valores que

delimitam o intervalo são escolhidos de tal forma a maximizar a probabilidade disto acontecer.

A grande desvantagem deste algoritmo é no tratamento de grandes arquivos em que será necessário ler mais de uma vez uma grande parte do arquivo, apesar de quanto maior o número de registros maior a eficiência do algoritmo. Desta maneira, para garantir que o arquivo só será lido uma única vez a cada cálculo da mediana, foi adotado o algoritmo de Weide[37]. Este algoritmo pode ser descrito da seguinte maneira:

1) Escolher para o arquivo um tamanho de célula que deve ser aproximadamente $t = \sqrt[s]{N}$ sendo s um inteiro qualquer e N a quantidade de registros do arquivo. O valor escolhido para s em geral é 2 e pode chegar a 3 para arquivos muito grandes.

2) Ler no arquivo os próximos t registros de modo a encher a célula que corresponde a um vetor na memória do computador.

3) Determinar a mediana dos registros deste vetor e armazenar este valor em outro vetor denominado Vetor das Medianas.

4) Voltar ao passo 2 até que todo o arquivo tenha sido lido.

5) Determinar o valor correspondente à mediana a partir do cálculo da mediana dos valores armazenados no Vetor das Medianas.

O valor obtido é aproximado mas não chega a comprometer o cálculo pois, no caso de chaves secundárias, a freqüência dos registros em cada valor é muito grande, o que anula na maioria das vezes o erro de aproximação.

O algoritmo usado nos passos 3 e 5 é o de Floyd & Rivest[15] que pode ser processado inteiramente na memória, desde que o tamanho escolhido para a célula assim o permita.

c) A Leitura e gravação do arquivo de dados pode ser simplificada pelas facilidades oferecidas pelo sistema operacional do B-6700, cujo método de acesso sequencial em disco permite o comando "SEEK", que consiste em posicionar o arquivo em uma dada posição do arquivo e após este posicionamento a leitura ou gravação dos registros pode ser feita de forma sequencial. Isto facilitou bastante o algoritmo de gravação do arquivo de saída, pois possibilitou o uso de um método de acesso simples como é o sequencial, para gravação de registros em diferentes posições do arquivo.

Este processo foi utilizado no programa de geração pois o particionamento prevê a gravação de dois sub-arquivos para cada conjunto particionado. O programa, entretanto, só utiliza um arquivo de saída sendo necessário então um esquema especial de gravação. Este esquema consiste em gravar registros do início para o meio do arquivo, se estes possuirem o valor da chave menor ou igual ao da mediana calculada, e do fim para o meio do arquivo no caso contrário. Isto se deve ao fato de que quando estamos particionando não sabemos a quantidade de registros que se posicionarão à direita e à esquerda do nó da árvore.

Uma outra alternativa seria a gravação de dois arquivos de saída que foi preterida por proporcionar maior gasto de memória para os "buffers" e não apresentar uma melhor eficiência.

Outra observação importante é que os registros são gravados em blocos de modo a minimizar o número de "SEEKS" e a quantidade de acessos ao disco. É devido a este fato que é sugerido que o tamanho de bloco do arquivo de dados seja escolhido entre os valores divisores do tamanho da célula do algoritmo de Weide[37].

A Figura IV.2 ilustra uma analogia do processo de leitura e gravação de registros no arquivo de dados. Os blocos de armazenamento representam na figura as células do algoritmo de Weide[37].

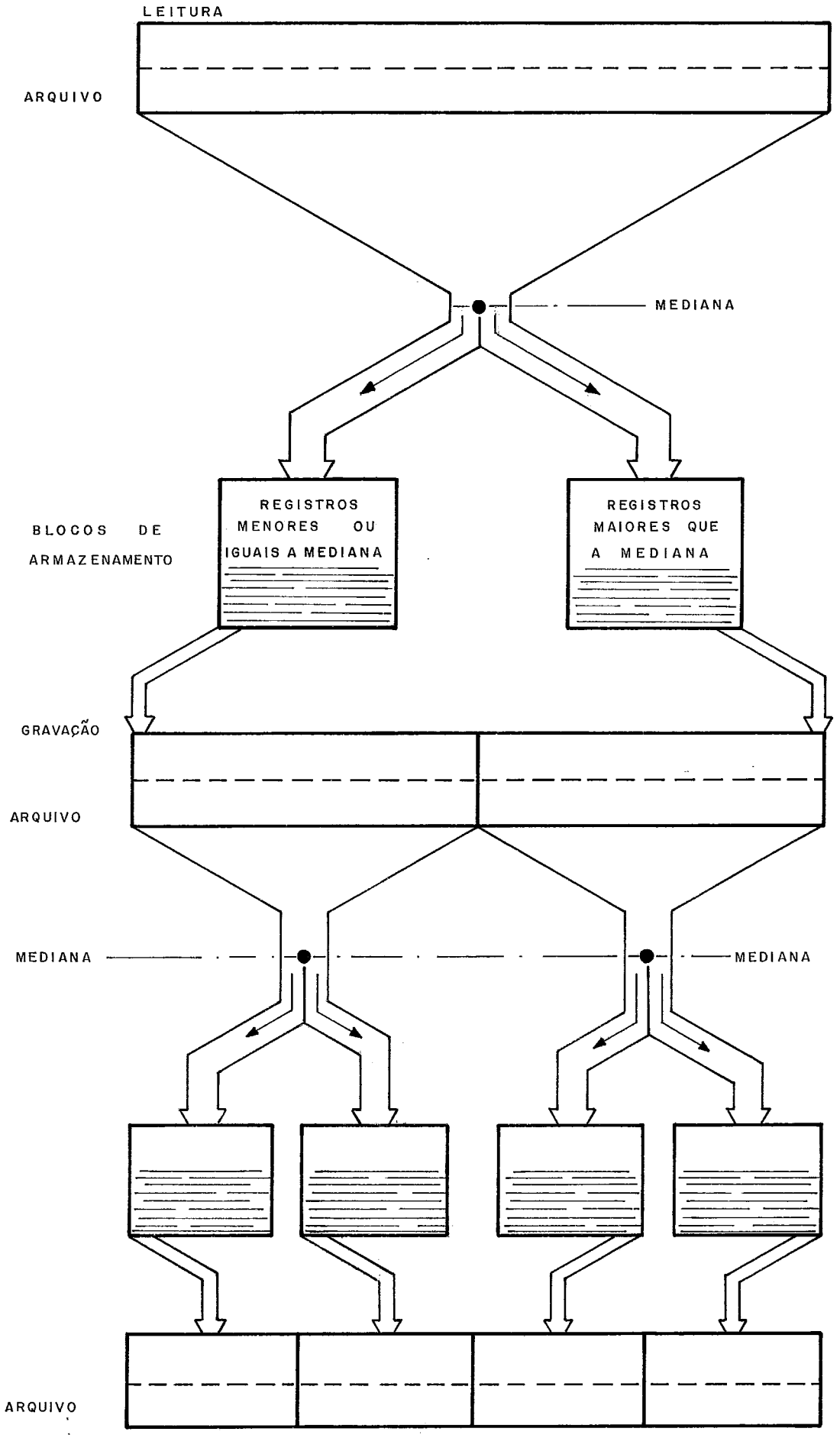



FIG. IV-2- ANALOGIA AO PROCESSO DE GRAVAÇÃO DO ARQUIVO DE DADOS

d) A leitura e gravação no arquivo árvore são feitas utilizando o método de acesso relativo do B-6700, que permite o acesso direto aos registros. O arquivo árvore executa duas funções diferentes no processo. A primeira refere-se a sua função principal, qual seja armazenar os nós da árvore e a segunda função é a de armazenar as informações de controle que serão utilizadas no próximo passo.

O processo de leitura e gravação no arquivo árvore é geral para todos os nós e pode ser explicado pelo seguinte ciclo:

1) Quando do particionamento do arquivo de dados segundo a mediana calculada, é feito o cálculo da mediana para cada um dos sub-arquivos e êstes valores das medianas são armazenados em registros do arquivo árvore, juntamente com os ponteiros que delimitam cada sub-arquivo.

2) Na próxima passada no arquivo de dados, após ter lido o primeiro registro de um sub-arquivo, é lido também o registro correspondente no arquivo árvore que contém a mediana que irá particionar em dois sub-arquivos estes registros de dados. Os ponteiros armazenados são utilizados neste momento para permitirà rotina de gravação dos registros de dados posicionar no arquivo de saída o início da gravação das células de registros maiores e não maiores que a mediana.

3) Após a leitura completa dos registros de um sub-arquivo de dados e a subsequente gravação dos dois sub-arquivos gera

dos no arquivo de saída é então regravado o registro no arquivo árvore, já agora com informações completas sobre os dois grupos de registros que ele aponta.

Durante o processo não é necessário armazenar ponteiros do arquivo árvore pois a gravação dos nós obedece a alocação proposta por Muntz & Uzgalis[27].

Entretanto, esta ainda não é a versão final do arquivo árvore, que será modificado pelo programa de arrumação para ajustar-se às condições ideais para a consulta.

Após o processo de geração descrito teremos como produto um arquivo de dados cujos registros estão ordenados pelos blocos apontados pela árvore ocupando N * T "bytes", sendo N o número de registros do arquivo de dados e T o tamanho do registro acrescido de seis bytes que são utilizados para controle durante o processo de geração.

O arquivo árvore terá $2^n - 1$ registros sendo n o número de níveis máximo da árvore e seu registro terá um tamanho variável em função do maior tamanho da chave que foi utilizada como discriminante. As demais informações constantes do registro somam nove bytes e contém o discriminante e os ponteiros dos blocos de dados à esquerda e à direita do nó representado pelo registro.

Como ilustração podemos citar o exemplo de um arquivo de dados com 1.000.000 de registros com 100 bytes de informação e cujo fator de bloco (número de registros no bloco) escolhido

seja 70 com taxa de ocupação média de 90% ocupará um espaço de 111 Mbytes divididos em aproximadamente 16.000 blocos. A árvore terá 14 níveis com 16.383 registros dos quais aproximadamente 8.000 apontarão blocos de dados (cada nó aponta dois blocos). Desta forma para um tamanho máximo da chave de cinco bytes, esta árvore ocupará 229 Kbytes, o que é praticamente irrelevante se comparado ao arquivo de dados.

O programa de arrumação consiste de duas partes: A primeira efetua a arrumação dos registros no arquivo de dados retirando do registro as informações de controle e abrindo os espaços nos blocos para os registros vazios, ou seja, transforma a separação lógica em separação física. A segunda parte procede a arrumação do arquivo árvore em blocos conforme descreve a figura IV.4 além de atualizar os ponteiros para os blocos de dados agora já rearrumados.

O processo é simples e poderia estar embutido no programa de geração. Isto não foi feito porque o programa de geração é em geral processado várias vezes até a versão final da estrutura e seria disperdiçado o tempo gasto com a arrumação quando não se tratasse de versão final. Além disso, para este processamento poderá ser especificado, de acordo com a conveniência, um valor de bloco diferente do adotado pelo programa de geração. Isto então pode ser obtido sem necessidade de regenerar a estrutura, desde que ele seja superior ao especificado durante a geração.

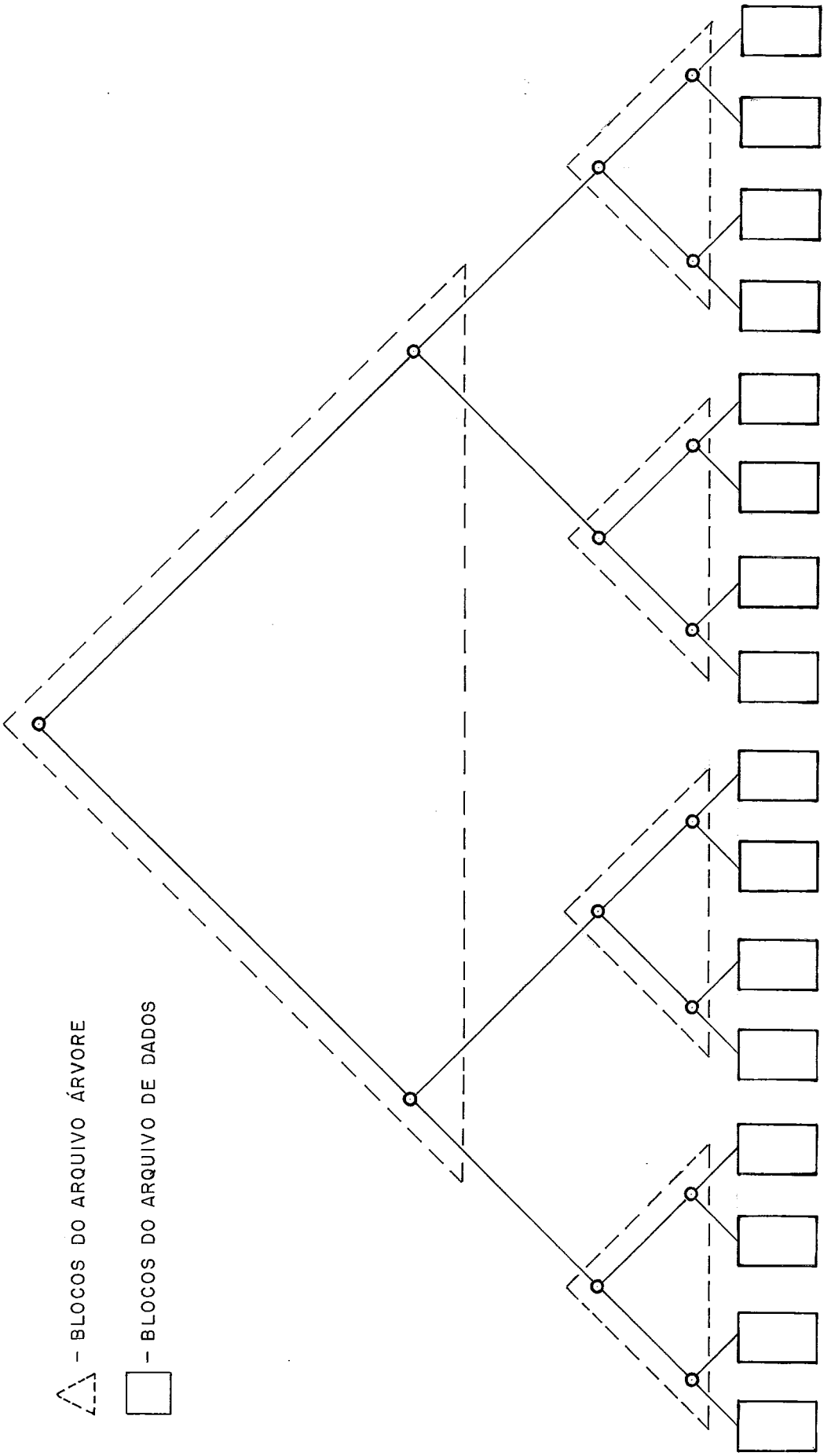



FIG. IV-3 – ARRUMAÇÃO DOS ARQUIVOS

IV.2.3) Utilização dos Programas

O processo de geração é bastante interativo, sendo necessário às vezes, várias tentativas até obter-se uma geração considerada aceitável. Para orientar o trabalho de geração da estrutura, a escolha dos parâmetros e a análise dos resultados, é que esta seção foi escrita. Estão descritos nas folhas seguintes os vários passos do processo de geração e uma análise da influência de cada parâmetro de controle neste processo.

Dois procedimentos são necessários antes do efetivo processamento dos programas. O primeiro deles é a geração do arquivo de dados a ser usado como entrada para o primeiro passo da geração. Este arquivo deverá ser sequencial, com registros de tamanho fixo e a descrição do registro deverá estar na biblioteca do sistema em formato COBOL.

Para maior eficiência é aconselhável só permanecer no arquivo os dados que serão usados como discriminantes (chaves) e um identificador único para cada registro que poderá ser gerado, caso não exista entre os demais dados do arquivo. Os dados restantes deverão estar associados ao mesmo identificador para serem reagrupados novamente no final do processo. Isto aumentará bastante a eficiência do processo pois a transferência de dados entre a memória e o disco e vice-versa será tão menor quanto menor for a quantidade de informação em cada registro. O processo de separação e reagrupamento será amplamente compensado por esta economia.

O segundo procedimento se refere a inicialização das variáveis que compõem o registro de controle. Estas informações serão utilizados por todos os programas do sistema e poderão ser modificados no decorrer do processo de geração a fim de que a estrutura atinja as condições desejadas. Uma descrição de cada campo é apresentada a seguir:

a) Tamanho do bloco - Corresponde ao tamanho do "buffer" de armazenamento dos registros após o particionamento e também o tamanho da célula para o algoritmo de Weide[37] . A unidade é registros e o valor se situa entre 100 e 1.000. Além disso, é importante que este valor seja múltiplo do fator de bloco especificado para o arquivo de trabalho, pois desta maneira estaremos minimizando o número de "SEEKS" no arquivo. Este valor é dividido automaticamente por $\sqrt{2}$ a cada passada para adequar-se a fórmula de Weide.[37]

b) Tamanho da amostra - Corresponde ao percentual do tamanho do bloco, que será usado como amostra no algoritmo de Floyd & Rivest[15] . O valor é recalculado a cada passada em função da alteração do valor do tamanho do bloco, estabelecido o limite inferior igual a 5. Teoricamente quanto menor o tamanho da amostra menor será o custo do cálculo da mediana, entretanto, se escolhermos um valor muito pequeno isto poderá comprometer a amostra fazendo com que mais insucessos se deem no algoritmo de cálculo. Tamanhos de amostra correspondentes a 5% do tamanho do bloco proporcionam bons resultados, entretanto, isto irá depender de quão aleatória é a amostra e a distribuição dos valores das chaves pelos registros.

c) Limite de aproximação - Este parâmetro foi criado em função da modificação introduzida no algoritmo de Floyd e Rivest[15] de modo a considerar como mediana, os valores que se encontram dentro de certos limites de aproximação estabelecidos por este parâmetro. Desta forma, aumentaremos a eficiência do algoritmo comprometendo um pouco a exatidão de cálculo da mesma forma como propõe Weide[37] . Esta inexatidão será anulada quando o valor da mediana recair sobre um valor com grande freqüência de registros. Esta variante do algoritmo não é adotada quando do cálculo da mediana das medianas.

d) Limite de particionamento - Este parâmetro indica que as células, cuja quantidade de registros for menor ou igual a este valor, não serão mais particionadas. O tamanho do bloco final adotado para o arquivo de dados deverá ser superior ao valor estabelecido para este parâmetro. Como foi citado anteriormente, a escolha deste parâmetro é bastante importante para o processo pois em função dele poderá se alcançar melhores taxas de ocupação dos blocos.

e) Número de níveis - Êste parâmetro corresponde ao número máximo de níveis na árvore índice. Desta forma, o particionamento será interrompido quando êste valor for alcançado. Êste parâmetro serve para delimitar o particionamento quando este se efetua passo a passo como é sugerido mais adiante.

f) Número de chaves - Corresponde ao número de chaves do arquivo de dados mesmo que nem todas sejam utilizados como discri

minantes na árvore. Este parâmetro auxilia o programa na identificação e no tratamento das chaves.

g) Vetor discriminante - Em cada posição do vetor está associada uma chave que será usada como discriminante no nível correspondente a sua posição no vetor. A chave é identificada pela ordem em que ela aparece no arquivo de dados e o fato de usarmos um só dígito para esta identificação limita em 10 a quantidade de chaves possíveis. Para inicializar este vetor sugere-se a adoção do algoritmo proposto por Silva-Filho[34] .

As informações a seguir fazem parte do registro de controle mas são de uso exclusivo dos programas não sendo possível a alteração pelo usuário sem comprometer a integridade do sistema.

h) Situação dos arquivos - Indica se houve término anormal quando da última atualização do arquivo e se o mesmo pode ser considerado íntegro. Este parâmetro é descrito mais adiante no programa de atualização. Deverá ser inicializado com Ø.

i) Próximo bloco vazio - Corresponde ao próximo bloco vazio no arquivo de dados para quando for necessário proceder um "split" de um bloco. Este parâmetro é inicializado pelo programa de arrumação e alterado pelo programa de atualização.

j) Data da última atualização - Corresponde a data no formato DD/MM/AA que indica a data da última atualização processada no arquivo.

A descrição do registro de controle com a posição de cada parâmetro consta do apêndice II.

Procedida a inicialização do registro de controle pode-se então iniciar o processo de geração propriamente dito. Este processo é feito em 2 fases de processamento idêntico no que se refere ao tratamento dos arquivos, mas com procedimentos diferentes no que se refere a interpretação dos resultados obtidos. A primeira fase comporta a geração do primeiro bloco da árvore, ou seja, os n/2 primeiros níveis do índice. Neste primeiro bloco é muito importante se obter o melhor balanceamento possível, pois um desbalanceamento nos primeiros níveis irá refletir enormemente sobre os últimos níveis, comprometendo a eficiência da estrutura. Desta forma, é preferível sacrificar a ordem dos discriminantes em função de um melhor balanceamento.

Para considerar este balanceamento satisfatório, as seguintes condições devem ser atingidas:

a) Nenhuma célula deverá alcançar o limite estabelecido para o não particionamento;

b) O tamanho da menor célula deverá ser superior a metade do tamanho da maior célula.

A primeira condição irá garantir que o primeiro bloco do arquivo árvore não será alterado a curto prazo pelo processo de atualização garantindo assim uma vida mais longa a estrutura.

A segunda condição garante neste momento, que a diferença entre os blocos apontados pelo índice é de um nível.

Se estas condições não forem satisfeitas, o problema pode ter sido causado por uma má escolha do discriminante, seja em função da distribuição dos registros pelos valores das chaves, seja por uma dependência com algum discriminante usado em um nível anterior. A informação de como está se dando o desbalanceamento consta do relatório produzido pelo programa de geração.

Para o segundo caso deve-se tentar inverter a ordem de utilização das discriminantes ou afastá-los o máximo de modo que a dependência possa ser anulada pelo particionamento por outras chaves. No primeiro caso deve-se afastar o discriminante da raíz de modo a tentar espalhar a sua distribuição. Várias combinações devem ser tentadas e se não for obtido o resultado desejado deve-se repetir o processo de escolha do vetor discriminante impondo como restrição o observado durante as tentativas sem êxito.

Nesta primeira fase aconselha-se a utilização do programa gerando a estrutura de uma vez só, limitando o particionamento através do campo número de níveis.

Conseguida a geração do primeiro bloco de registros da árvore passa-se para a próxima fase de geração que se refere aos demais particionamentos. Nesta fase sugere-se que se utilize a opção da geração nível a nível de modo a poder constatar as eventuais anormalidades antes de prosseguir o particionamento. Estas

anomalias são observadas em um particionamento excessivamente desbalanceado, (mais de 30%) quando deve ser verificado a sua ocasionalidade. Em se tratando de um problema generalizado para esta chave há de se estudar a variância da chave a fim de decidir sobre a exclusão desta chave como discriminante ou se o problema pode ser diminuído abaixando-se o nível em que a chave está sendo usado como discriminante. No caso de alteração será necessário repetir as duas últimas passadas no arquivo (o cálculo e o particionamento).

É nesta fase que se faz a escolha final do tamanho do bloco do arquivo de dados. Quando um particionamento acusar um tamanho de célula inferior ao dobro do tamanho de bloco desejado deve-se processar o programa que calcula a porcentagem de ocupação estimada para um determinado tamanho de bloco. Este programa calcula a taxa de ocupação para até 10 valores diferentes de modo que pode-se especificar diversos valores e verificar aquele que melhor se adaptar as condições desejadas. Esta flexibilidade para escolha foi preferida pois há outros fatores que influem na escolha do tamanho do bloco além da taxa de ocupação, como tamanho dos "buffer" de memória e o tamanho do segmento no disco em que será armazenado o arquivo. Escolhido o valor altera-se esta informação no arquivo controle e repete-se, se for o caso, todas as passadas em que o tamanho de qualquer das células for inferior ao valor escolhido para ser o tamanho do bloco. Na última passada o programa informará qual a taxa de ocupação média que o valor escolhido irá acarretar no arquivo de dados.

Outro conselho importante é fazer com que nesta segunda fase o tamanho do bloco seja superior ao tamanho da maior célula. Isto acarretará a suspensão automática do método aproximado de Weide[37] substituindo-o unicamente pelo método de Floyd & Rivest[15], isto porque cada célula ocupará um único bloco, e ao se calcular a mediana do bloco estará se calculando a mediana da célula. Isto, em geral, não é alcançado na primeira fase da geração porque os tamanhos de células são muito elevados impossibilitando o uso desta alternativa por limitações de memória. Entretanto, a alternativa pode e deve ser usada a partir do momento em que esta limitação não mais ocorra.

O programa de rearrumação permite somente a possibilidade de se aumentar o tamanho final do bloco aumentando-se o espaço vazio em todos os blocos. É importante observar que não será possível diminuir este tamanho, pois podem haver células cuja quantidade de registros seja igual ao valor inicialmente escolhido, o que iria acarretar uma sobra de registros no bloco.

O programa de arrumação insere também no início de cada bloco um registro de controle que facilitará o processo de consulta e aumentará a eficiência da atualização.

## IV.3) CONSULTA AOS DADOS

### IV.3.1) Considerações Iniciais

A consulta aos dados é o objetivo principal do método de acesso por ser o produto final esperado pelo usuário do sistema. Com o objetivo principal de facilitar e minimizar os tempos de resposta é que foi projetada e construída a estrutura. Desta forma, é de se esperar que o resultado conseguido pelo método de acesso para recuperação de informações seja a medida da eficiência do método de acesso no que diz respeito ao usuário final.

Precisando alcançar a satisfação plena do usuário, o programa de consulta foi construído para além de recuperar os registros utilizando o algoritmo de procura, oferecer ao usuário opções que vão desde a obtenção de resultados parciais mais rápidos de modo a dirigir a consulta, até a possibilidade de saída dos dados em forma de arquivo para serem listados ou para que outro programa possa tratá-los separadamente.

Os resultados alcançados nas experiências comprovaram inteiramente os resultados dos modelos matemáticos citados por Bentley[3] e demonstram a viabilidade de uso da estrutura como uma solução para o problema de recuperação de informações.

IV.3.2) Descrição do Programa

O programa está dividido em seis partes que estão listados abaixo e a mesma divisão será usada para descrever o programa.

1) Identificação do usuário e inicialização;

2) Recebimento das consultas;

3) Procura no bloco residente da árvore;

4) Procura nos demais blocos da árvore;

5) Procura nos blocos de dados;

6) Resposta final à consulta.

Após os passos 3 e 4 são oferecidos resultados parciais e a possibilidade de interromper o processamento. Após o passo 5 é oferecido o resultado numérico da consulta e o usuário especifica então como deseja a saída dos dados.

1) Identificação do Usuário e Inicialização

Nesta fase é alocado o terminal e é pedido ao usuário que se identifique através do nome e uma senha que será testada pelo programa. Desta forma, é possível prover mais segurança aos dados quando a informação tratada for confidencial. É nesta fase que é carregado na memória o primeiro bloco da árvore. Êste procedimento tem por objetivo principal minimizar os acessos a disco no arquivo árvore, pois o primeiro bloco será sempre consultado. Este bloco é armazenado na memória como um vetor e o acesso aos registros se dá exatamente igual ao do arquivo, ou seja, po

sicionando o índice do vetor na chave do registro desejado e transferindo o conteúdo para a área de trabalho do arquivo.

2) Recebimento da Consulta

Nesta fase é apresentado ao usuário uma tela que possibilite-o especificar a consulta desejada.

O programa recebe os dados e efetua uma crítica para verificar se não houve erro na especificação da pergunta. O único erro detectável é se, no caso da especificação de uma região, o valor superior é menor que o valor inferior. Em caso de erro, este é acusado e o usuário deverá repetir a consulta consertando os erros acusados.

3) Procura no Bloco Residente da Árvore

Nesta fase são pesquisados os registros do bloco residente da árvore e o resultado será uma lista de blocos que deverão ser visitados no próximo passo.

O processo é idêntico ao apresentado em III.3 e o algoritmo construído é apresentado no apêndice III.

4) Procura nos demais Blocos da Árvore

Nesta fase são pesquisados cada um dos blocos da árvore relacionados no passo anterior, seguindo o mesmo algoritmo de pesquisa. A diferença é que agora os blocos apontados pelos nós da árvore são os do arquivo de dados.

Antes de iniciar esta fase é, entretanto, fornecido o resultado da fase anterior e o usuário decidirá se a pesquisa deve ou não continuar.

5) Procura nos Blocos de Dados

O resultado da fase anterior é apresentado ao usuário sob a forma de número de blocos do arquivo de dados que deverão ser acessados. Caso o usuário decida prosseguir a procura, serão lidos os registros destes blocos comparando-se os valores das chaves de cada registro com os valores especificados na consulta e selecionando-se aqueles que satisfaçam as condições.

Como foi visto no Capítulo III, se a árvore for totalmente balanceada, a quantidade de blocos acessados será igual a $2^t$ sendo t o número de níveis em que o discriminante não é especificado, para o caso de uma pesquisa por escore parcial. Uma grande quantidade de blocos significará que, ou a pergunta foi demais abrangente (poucas chaves foram especificadas), ou as chaves especificadas não são discriminantes poderosas.

6) Resposta FINAL a Consulta

O usuário após o passo 5 recebe as informações de quantos registros foram percorridos e quantos satisfizeram realmente as condições da pergunta. O usuário decidirá então se os registros deverão ser listados no terminal, em listagem ou armazenados em um arquivo à parte.

## IV.4) ATUALIZAÇÃO DE DADOS

### IV.4.1) Considerações Iniciais

A atualização dos registros de dados é a terceira parte do método de acesso e é processada pelo programa de atualização. Este se incumbe de incluir e excluir os registros dos arquivos de dados e proceder alterações na estrutura quando as inclusões ocasionarem o enchimento ou esvaziamento completo do bloco.

Quando da descrição do programa de geração, foi sugerido que o tamanho do bloco fosse escolhido de modo a deixar espaços nos blocos para inclusões de novos registros. Este número de espaços será diferente para cada um dos blocos, pois irá depender da quantidade de registros na célula. Se as inclusões de novos registros se derem proporcionalmente à quantidade de espaços em cada bloco será possível efetuar as inclusões, sem que seja necessário criar novos blocos de dados e reestruturar a árvore índice. Esta proporcionalidade ocorrerá se os valores das chaves nos registros incluídos mantiverem a mesma distribuição observada na fase de geração da estrutura.

### IV.4.2) Descrição do Programa

O programa de atualização utiliza o algoritmo de procura um pouco simplificado, por não ser necessário prever o caso em que a INTERSEÇÃO com o nó da árvore se dê à direita e à esquerda, ou seja, é o algoritmo de consulta para o caso particular de