ICMSC-USP
PÓS-GRADUAÇÃO

I.C.M.S.C.

INSTITUTO DE CIÊNCIAS MATEMÁTICAS DE SÃO CARLOS

ESTRUTURAS DE DADOS PARA BUSCA POR INTERVALO

Heloisa de Arruda Camargo Scarpelli

Orientadora: *Profa. Dra. Maria Carolina Monard*

UNIVERSIDADE DE SÃO PAULO

SÃO CARLOS - SÃO PAULO
BRASIL

ICMSC-USP
PÓS-GRADUAÇÃO

ESTRUTURAS DE DADOS PARA BUSCA POR INTERVALO

Heloisa de Arruda Camargo Scarpelli

Orientadora: *Profa. Dra. Maria Carolina Monard*

Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos, da Universidade de São Paulo, para a obtenção do Título de Mestre em Ciências de Computação e Estatística.

São Carlos
-1984-

AGRADECIMENTOS

A todos que contribuiram para a concretização deste trabalho quero expressar meus agradecimentos, destacando em especial:

A dedicação, incentivo e orientação com que fui distinguida pela professora Dra. Maria Carolina Monard.

O apoio, interesse e amizade dos professores Dr. Odelar Leite Linhares e Doris Ferraz de Aragon, esta, também responsável pelos meus primeiros passos na pós-graduação.

Agradeço também:

Ao Departamento de Computação e Estatística da Universidade Federal de São Carlos.

Aos funcionários e amigos do Instituto de Ciências Matemáticas de São Carlos, da Universidade de São Paulo.

À Maria do Socorro Gonçalves de Souza e Sueli Paulina Sanchez pelo trabalho de datilografia.

Finalmente, aos meus pais que possibilitaram a formação acadêmica que me permitiu viabilizar este trabalho.

Este trabalho contou com o apoio financeiro das entidades: CAPES e FINEP

RESUMO

Neste trabalho são investigadas várias estruturas de dados para Busca por Intervalo em arquivos estáticos.

Cada estrutura é apresentada junto com a descrição dos procedimentos de pré-processamento e busca e da análise de seu comportamento, fornecendo os custos de pré-processamento, armazenamento e busca. As estruturas mais elaboradas são analisadas em detalhe.

São consideradas as situações nas quais uma estrutura é claramente superior à outra e finalmente são relecionados vários problemas em aberto que podem dar origem a trabalhos futuros.

ABSTRACT

In this work we investigate some data structures for range searching on static files.

Each structure is presented together with the description of the preprocessing and searching procedures and the analysis of its performance, giving the preprocessing, storage and query costs. The most elaborated structures are analysed in detail.

We consider the situations on wich one structure is clearly superior to the others and finally relate several open problemas for further work.

ÍNDICE

CAPÍTULO 1. INTRODUÇÃO 1

CAPÍTULO 2. BUSCA POR CHAVES MÚLTIPLAS 5

2.1. Tipos de Busca..... 6

CAPÍTULO 3. ESTRUTURAS BÁSICAS PARA BUSCA POR INTERVALO 11

3.1. Busca Sequencial 11

3.2. Projeção 13

3.2.1. Análise de Componentes Principais(ACP).20

3.3. Células 24

CAPÍTULO 4. ÁRVORES QUAD 35

4.1. Inserção 42

4.1.1. Algoritmos de Inserção 42

4.2. Busca 48

4.2.1. Algoritmo de Busca 53

4.2.2. Árvore Quad Ótima 61

4.3. Análise de Busca por Intervalo para Árvore Quad Perfeita 62

4.4. Análise do Pior Caso de Busca por intervalo para Árvores Quad Balanceadas.....76

4.5. Conclusões..... 91

CAPÍTULO 5. ÁRVORES K-D 93

5.1. Inserção 103

5.1.1. Algoritmo de Inserção 104

5.1.2. Propriedades 110
5.2. Busca 113
5.2.1. Algoritmo de Busca 120
5.2.2. Árvore k-d Ótima 122
5.3. Análise de Busca por Intervalo para Árvores k-d Balanceadas.....128
5.4. Análise do Pior cado de Busca por Intervalo para Árvores k-d Balanceadas 138
5.5. Árvore k-d não homogênea 159
5.6. Conclusões 162

CAPÍTULO 6. OUTRAS ESTRUTURAS 166

CAPÍTULO 7. CONCLUSÕES FINAIS E TRABALHO ADICIONAL 169

REFERÊNCIAS 173

1. INTRODUÇÃO

Um dos problemas fundamentais da ciência de computação é o problema de busca em arquivos e muitas estruturas de dados e algoritmos eficientes têm sido desenvolvidos para resolvê lo nas suas mais variadas formas. A maior parte dessas estruturas e algoritmos está voltada para o caso de busca por chaves primárias, isto é, busca sobre arquivos constituidos por registros identificados por um único campo de "chave". Para esta forma de busca muitas pesquisas já foram desenvolvidas e existe uma extensa bibliografia que trata do assunto.

A situação é diferente quando se trata de busca por chaves múltiplas, ou seja, a busca baseada nos valores dos outros campos do registro além da chave primária, que também é chamada de "busca multidimensional", "busca por chaves secundárias", "recuperação por atributos múltiplos" ou "busca associativa". Há algum tempo atrás pouco se conhecia sobre estruturas de dados para conjuntos com muitas chaves mas nos últimos anos este problema tem sido foco de um grande número de pesquisas e diversas estruturas começaram a ser projetadas.

Uma busca sobre um conjunto de informações é iniciada com uma requisição do usuário, chamada "consulta". A consulta coloca condições sobre as chaves dos registros e a busca é realizada para procurar registros no arquivo que satisfaçam essas condições. A busca por chaves múltiplas pode ser de vários tipos, dependendo da maneira com que a consulta estipula condições sobre as chaves. As estruturas de dados são, em geral, mais ou menos adequadas à um ou outro tipo de busca.

O objetivo deste trabalho é analisar métodos para

resolver um problema particular de busca por chaves múltiplas: a busca por intervalo. Colocando este problema em termos geométricos têm-se um conjunto F de n pontos no k-espaço para pré processar em uma estrutura de dados. Depois de pré-processar os pontos deve-se responder consultas que pedem todos os pontos x de F tal que a primeira coordenada x_1 de x está em algum intervalo $[\ell_1, u_1]$, a segunda coordenada $x_2 \in [\ell_2, u_2]$,... $x_k \in [\ell_k, u_k]$.

Um método para resolver o problema de busca por intervalo é caracterizado por um estrutura para armazenar os dados, pelo algoritmo para construção ou pré-processamento dessa estrutura e algoritmos para a busca. Cada estrutura é analisada usando-se três funções de custo que dependem de n, número de pontos e de k, número de dimensões. Logo, para uma estrutura de busca A, usa-se a seguinte notação [BENTLEY-79C]:

1. $P_A(n,k)$ representa o custo do pré-processamento de n pontos no k-espaço em uma estrutura de dados;
2. $S_A(n,k)$ representa o espaço de armazenamento exigido pela estrutura;
3. $Q_A(n,k)$ representa o tempo de busca, ou o custo da consulta.

Estes custos são em geral apresentados como análise do pior caso, podendo aparecer associados com o caso médio, quando mencionado explicitamente.

É importante observar que estão sendo considerados apenas conjuntos de dados estáticos. Apesar do procedimento de inserção ser descrito para algumas estruturas, não são tratados os casos em que um grande número de alterações, ou seja, inser

ções e eliminações, é feito alternadamente com a busca.

No capítulo 2 deste trabalho, são apresentados alguns tipos de busca por chaves múltiplas mais conhecidos.

No capítulo 3 são apresentadas estruturas chamadas básicas para a busca por intervalo: Sequencial, Projeção e Células. São apresentadas de maneira rápida por serem razoavelmente simples, juntamente com uma descrição dos procedimentos de pré-processamento e busca. São feitas as análises de seu comportamento e os custos são apresentados de acordo com as funções de custo já definidas.

No capítulo 4 é analisada a estrutura de árvores Quad, introduzida por Finkel e Bentley em [FINKEL-74]. Sua descrição e suas análises são feitas para o caso particular de registros com duas chaves. Apesar de se tratar de uma estrutura com características interessantes pode tornar-se muito custosa em termos de memória quando os registros tem mais que duas chaves. São apresentados algoritmos para inserção e busca e análises mais detalhadas da busca por intervalo em uma árvore Quad perfeita e do pior caso desta busca para árvores Quad balanceadas.

O capítulo 5 é dedicado à apresentação da estrutura de árvores k-d [BENTLEY-75], que caracteriza-se por ser de grande versatilidade e porisso de muita utilidade. Trata-se de uma generalização da árvore binária de busca unidimensional que tem as mesmas propriedades que esta. Também são vistos algoritmos de busca e inserção e análises mais detalhadas da busca por intervalo para árvores k-d balanceadas e análise do pior caso desta busca.

O capítulo 6 descreve brevemente outras estruturas para busca por intervalo que têm interesse fundamentalmente teórico.

O capítulo 7 apresenta as conclusões gerais com um resumo dos custos de cada uma das estruturas e relaciona problemas em aberto observados durante as pesquisas que dão origem a trabalhos futuros.

2. BUSCA POR CHAVES MÚLTIPLAS

Formalmente o problema de busca por chaves múltiplas relaciona-se com um arquivo F que é uma coleção de registros, onde cada registro R de F é uma m-upla ordenada $(v_1, v_2, \ldots v_m)$ de valores que são os campos ou atributos do registro. Uma requisição de busca é chamada uma "consulta" ao arquivo e especifica condições sobre alguns ou todos os atributos. Os registros cujos atributos satisfazem as condições especificadas formam a resposta à consulta e devem ser listados pelo algoritmo de busca. Dizemos que um campo ou atributo é uma chave se uma consulta pode se referir a ele.

Por exemplo, vamos supor que em uma universidade há um conjunto de informações sobre seus alunos, como: nome, número, período, número de créditos já cursados, média geral e curso. Para saber quais os alunos de um determinado curso que teriam chance de se candidatar a algum tipo de bolsa de estudos, pode-se fazer a seguinte requisição, estipulando-se condições sobre os campos: "quais os alunos do curso de engenharia que estão cursando o 7º ou o 8º período e tem média superior a 7.0".

Neste exemplo os campos: curso, período e média são, com certeza, chaves. Quanto aos outros campos: nome, número e número de créditos, serão também considerados chaves se existir outra consulta aplicável a esse arquivo que se refira a algum deles, caso contrário serão considerados apenas atributos.

Portanto, se um registro contém m valores, pode conter $k \leq m$ chaves. Para simplificar, vamos de agora em diante, nos referir a registros pelo número de chaves que ele possui,

sem mencionar explicitamente os outros m-k atributos que possam existir. Os registros serão tratados apenas como uma k-upla de valores ($k \leq m$) que são as suas chaves. Assumimos, de modo geral, que outras informações podem estar presentes no registro, mas como elas não tem influência nas estruturas e métodos analisados, não serão mais consideradas.

O problema de busca por chaves múltiplas pode também ser colocado em termos geométricos, considerando-se cada chave do registro como uma dimensão do espaço. Assim, os k valores (v_1, v_2,..., v_k) de cada registro representam um ponto no espaço de coordenadas k-dimensional. O arquivo de registros se torna um conjunto de pontos no k-espaço e uma consulta especificando critérios a serem satisfeitos consiste em uma região desse espaço. Os pontos que se localizam na região do espaço delimitada pela consulta constituem a resposta para a consulta.

A busca por chaves múltiplas engloba várias formas diferentes de busca, que variam de acordo com a maneira com que as condições sobre as chaves são especificadas. A análise de uma estrutura depende, em geral, do tipo de busca.

A seguir apresentamos uma classificação geral, que descreve alguns dos tipos mais conhecidos de busca, entre eles a busca por intervalo, que será o tipo enfocado neste trabalho.

2.1. *Tipos de Busca*

Presume-se nessa classificação que cada registro do arquivo tem k chaves K_1, K_2,...,K_k e denota-se a chave i do re

gistro R por $K_i(R)$, $1 \leq i \leq k$.

a) Busca Exata - é o tipo mais simples de consulta e consiste em perguntar se um registro específico está no arquivo. A consulta define valores $v_1, v_2, \ldots, v_k$ para as chaves $K_1, K_2, \ldots, K_k$ e então, procura-se no arquivo registros com exatamente esses valores para as chaves, ou seja,

$$\{R/K_i(R) = v_i \text{ para } 1 \leq i \leq k\}.$$

b) Busca Parcial - neste segundo tipo de consulta os valores são especificados para um subconjunto próprio das k-chaves. Em geral, especifica-se valores para t das k chaves e pede se todos os registros que tenham aqueles t valores, independentemente dos outros (k-t) valores. Neste caso a consulta chama-se consulta de busca parcial com t chaves especificadas. Chamando T(t) o conjunto destas chaves, então, os registros que estão sendo procurados são:

$$\{R/K_i(R) = v_i \text{ para } i \in T(t)\}.$$

Por exemplo, na base de dados da Universidade, requisitando-se uma listagem de todos os alunos que estejam no 4º período e tenham média geral 7.0, estão sendo especificados valores para t = 2 chaves.

c) Busca por intervalo - é o tipo de consulta no qual é especificado um intervalo de valores para cada uma das k chaves, solicitando todos os registros que tenham valores nos intervalos

correspondentes. Por exemplo, em um arquivo com informações sobre usuários de um centro de processamento de dados, pode se querer saber quais os usuários que fizeram uma estimativa de uso de 1000 a 2000 cartões e de 500 a 800 formulários.

A sua definição pode ser colocada da seguinte maneira: especifica-se limites superiores c_i e limites inferiores b_i para os valores de cada uma das chaves K_i, $1 \leq i \leq k$. Os registros que estão sendo procurados são:

$$\{R/b_i \leq K_i(R) \leq c_i \text{ para } 1 \leq i \leq k\}.$$

Quando as consultas especificam intervalos para t das k chaves, $t < k$, determinam um tipo de busca chamada busca parcial por intervalo.

De acordo com a interpretação geométrica, a estipulação dos intervalos determina um hiper-retângulo no espaço k-dimensional, e a busca consiste em encontrar todos os pontos do espaço que se localizam dentro desse hiper-retângulo.

d) Busca por Região (Busca Booleana) - Trata-se de uma combinação dos tipos anteriores, usando-se os operadores booleanos AND, OR, NOT. Esse tipo de requisição é o mais geral e engloba todos os outros já definidos. Caracteriza-se por especificar uma região qualquer do k-espaço, onde devem localizar-se os pontos procurados.

e) Busca do Vizinho mais Próximo - As consultas definidas até agora, são usadas quando o que se quer saber é se a base de dados contém exatamente o que está sendo procurado, seja um,

nenhum ou vários registros. Em alguns casos porém, pode ser necessário procurar, quando a base não contém necessariamente o que foi especificado, o(s) registro(s) que mais se aproximem daquele que está sendo procurado. Por exemplo, se queremos um livro que trate de dez tópicos de um assunto e ele não se encontra no arquivo, podemos estar interessados em outro que trate da maioria desses tópicos. Ou ainda, se um arquivo contém informações sobre todas as cidades com agência de correio, e associada a cada cidade sua latitude e longitude e uma carta é endereçada a uma cidade sem agência, a cidade mais próxima que tenha uma agência pode ser selecionada como o destino.

Este problema, de encontrar um ou mais registros no arquivo que mais se aproxime de um registro dado é chamado problema do vizinho mais próximo, onde a proximidade é medida através de alguma função de dissimilaridade ou de distância, por exemplo distância euclidiana. Em geral o problema do vizinho mais próximo aplica-se a arquivos de dados que armazenam registros com valores reais, para que possa ser aplicada a medida de dissimilaridade.

Formalmente, dado um arquivo de n registros descritos por k valores cada, e uma medida de dissimilaridade D, o problema do vizinho mais próximo consiste em encontrar os m registros mais próximos de um registro dado (que possivelmente não está no arquivo).

Na colocação geométrica, dada uma função distância D e um ponto P, os m vizinhos mais próximos são os pontos

$$\{Q_i,\ i=1,\ldots,m \mid D(P,Q_i) < D(P,R), \forall\ R, R \neq Q_i,\ i=1,\ldots,m\}$$

Outras espécies de consultas poderiam ser citadas, originadas de necessidades práticas ou de outras combinações dos tipos já definidos. Uma modificação da consulta de vizinho mais próximo poderia permitir especificações gerais como encontrar o vizinho mais próximo de um intervalo, ou de uma região, em vez de um só registro. O uso de outros operadores lógicos além de AND, OR e NOT poderia incrementar o trabalho de busca booleana. Outras consultas podem ser ainda mais complicadas, como pedir todos os pares de registros que tenham o mesmo valor em cinco ou mais atributos, sem especificar quais, ou ainda permitir que as condições se refiram a valores que não estão armazenados como atributos explicitamente, mas tenham que ser deduzidos de outros atributos através de alguma função.

Essas questões não se enquadram no âmbito da análise desenvolvida neste trabalho e não serão consideradas, apesar de frequentemente poderem ser subdivididas em outras mais simples que se encaixem em um dos cinco tipos básicos definidos.

3. ESTRUTURAS BÁSICAS PARA BUSCA POR INTERVALO

De agora em diante, vamos nos concentrar em um tipo particular de busca por chaves múltiplas, que é a busca por intervalo. Neste capítulo serão apresentadas três estruturas básicas para resolver esse problema: Sequencial, Projeção e Células. Estas estruturas estão aqui agrupadas por serem razoavelmente simples e representarem as formas mais conhecidas de se tratar arquivos para busca por intervalo. Os algoritmos para busca, ou outras operações sobre essas estruturas não estão detalhados, pois a análise das estruturas em função dos parâmetros P(n,k), S(n,k) e Q(n,k) pode ser feita facilmente com base nos procedimentos gerais sem necessidade de nos prendermos aos passos mais detalhados de um algoritmo. Para simplificar, assumimos daqui prá frente que, as chaves de um registro $K_i(R)$ e os valores de uma chave v_i serão referenciados indistintamente por K_i, sendo seu significado interpretado de acordo com o contexto em que se apresenta.

3.1. *Busca Sequencial*

Entre as estruturas que podem ser utilizadas para resolver o problema de busca por intervalo, a mais simples é a lista sequencial. Para cada consulta essa lista seria percorrida sequencialmente e todos os registros satisfazendo as condições estipuladas seriam selecionados. Quando as perguntas não precisam ser respondidas imediatamente, elas podem ser acumula

das para que, depois de um certo período de tempo, várias sejam respondidas de uma só vez. Assim, com uma só passada pelo arquivo, procura-se registros que satisfaçam a cada uma das consultas colocadas.

Como uma das vantagens mais evidentes, aponta-se o fato de que essa estrutura é muito simples de ser usada e facilmente implementada em qualquer meio. Nos casos em que o arquivo é pequeno e o número de atributos é grande, e quando as consultas podem ser acumuladas, o que faz com que quase todos os registros satisfaçam pelo menos uma das consultas, esse método mostra-se conveniente e apresenta bons resultados.

A análise de custos pode ser feita facilmente: o trabalho de pré-processamento consiste no simples armazenamento de cada um dos n registros com k chaves.

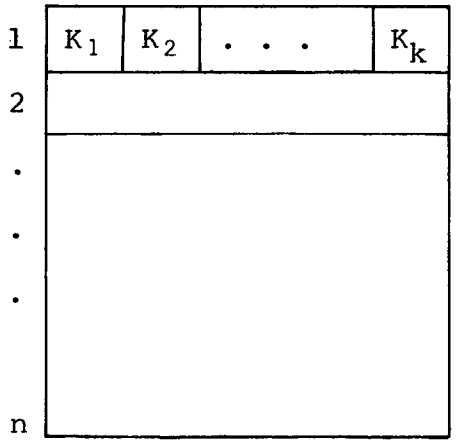


Fig. 3.1

Então o custo de pré-processamento será $P_{bs}(n,k) = O(nk)$. De maneira análoga observa-se que é necessário armaze

nar os n registros de k chaves e examinar cada um deles para realizar uma busca; com isto os custos de armazenamento e de busca são da mesma ordem que o custo de pré-processamento.

Em consequência, os custos da Busca Sequencial (bs) são:

$$P_{bs}(n,k) = O(nk)$$

$$S_{bs}(n,k) = O(nk)$$

$$Q_{bs}(n,k) = O(nk).$$

A estrutura de Lista Sequencial apresenta a mesma facilidade de uso e vantagens citadas acima, para outros tipos de busca além da busca por intervalo, especialmente quando se deve testar condições extremamente complicadas.

3.2. Projeção

O segundo método, a projeção, consiste em manter para cada chave uma lista dos registros ordenados por ela. Na representação geométrica do problema onde os n registros de k chaves são considerados como n pontos de k dimensões, vê-se cada uma das listas citadas acima como a projeção dos pontos em cada coordenada do espaço. A figura 3.2 ilustra esta representação para registros com duas chaves:

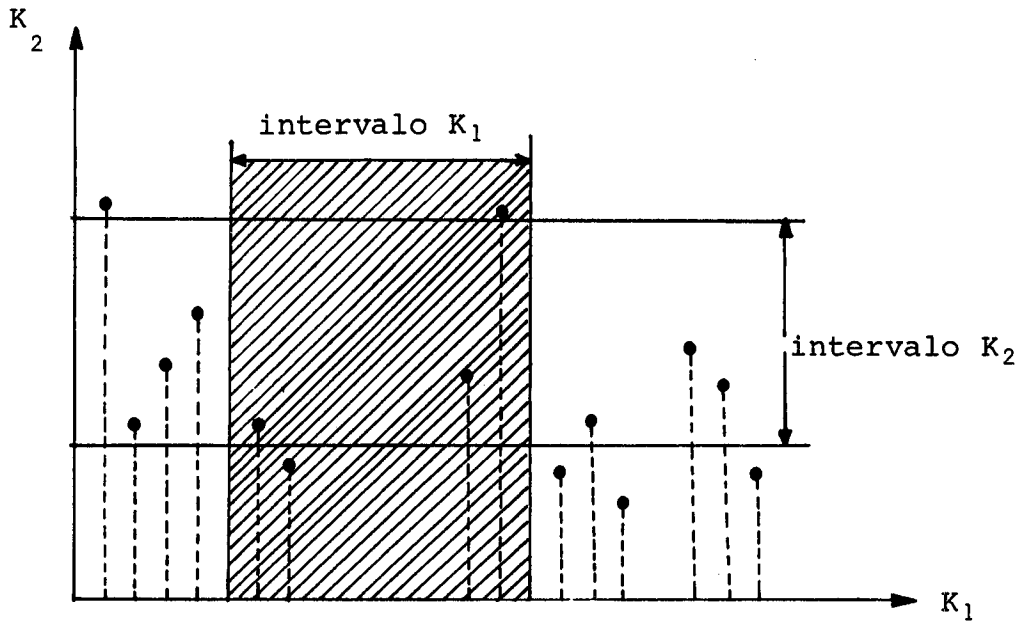


Fig. 3.2

Os pontos representam um conjunto de registros de 2 chaves cada, indicadas pelas coordenadas K_1 e K_2. As linhas pontilhadas representam a projeção dos registros na coordenada K_1, isto é, a lista dos registros ordenados pela primeira chave. A faixa vertical representa o intervalo especificado para a primeira chave, a horizontal representa o intervalo para a segunda chave, e o retângulo que fica na intersecção das duas contém os pontos que satisfazem a consulta. Para responder essa consulta teríamos que investigar somente os quatro pontos que se encontram na área hachurada, ou seja, no intervalo de K_1 para ver quais deles pertencem também ao intervalo K_2.

Essas k listas que representam as projeções ordenadas podem ser obtidas aplicando-se k vezes um algoritmo de classificação, ou seja, seu custo de pré-processamento é $P_p(n,k) =$

$= O(kn \log_2 n)$.

Neste método é de grande importância o pré-processamento, que em princípio envolve somente a ordenação das k listas. Existem porém, situações em que é vantajoso aplicar técnicas mais elaboradas para preparar a estrutura para as buscas, como será visto mais adiante.

Depois do pré-processamento, para responder consultas de busca por intervalo, o processo consiste em fazer a intersecção dos vários subconjuntos das listas, determinados pelos valores da consulta que definem os intervalos de busca.

O primeiro passo para a execução desta busca é a seleção de uma das chaves referenciadas na consulta e a procura, na sua lista ordenada, dos limitantes superior e inferior que determinam o intervalo para essa primeira chave. Nesse intervalo estão todos os pontos que satisfazem a consulta para essa chave, sendo que, provavelmente, uma parte do arquivo inicial terá sido excluida. A partir daí a busca continua, aplicando-se outras comparações sobre esse conjunto reduzido. Uma possível forma de comparação seria, por exemplo, selecionar uma próxima chave, digamos K_i e percorrer essa lista encontrada no passo anterior para comparar os valores de K_i e selecionar os registros que estão também dentro do intervalo especificado para K_i. Isto poderá eliminar mais alguns registros, resultando numa lista ainda menor. Assim o processo se repete sucessivamente, selecionando e comparando todas as demais chaves até obter a intersecção de todos os intervalos.

Uma outra forma de iniciar a busca seria encontrar, não só para uma chave, mas para todas, os limitantes inferiores

e superiores e entre todos os intervalos, tomar aquele com menor número de registros para continuar a busca, o que com certeza diminui ao máximo o primeiro subconjunto de registros que vai servir de base para as outras comparações. Apesar de eliminar-se inicialmente o maior número possível de registros na escolha desse intervalo, este processo envolve um trabalho maior para cada busca, já que todos os limitantes superiores e inferiores são procurados. No entanto, nos casos em que os intervalos tenham tamanhos muito diferentes, este método pode ser uma boa opção. Assim é possível calcular o número de elementos de cada intervalo e com base neste dado, selecionar as chaves a serem comparadas. Por exemplo, na figura 3.2, conhecendo os intervalos K_1 e K_2, podemos saber quantos elementos estão em cada um e escolher a chave K_1 para fazer a primeira eliminação.

O primeiro processo de busca descrito envolve acesso a apenas uma das listas de projeções para determinar um primeiro subconjunto de registros sobre o qual são realizadas as comparações. Uma variação dessa técnica seria manter uma lista de projeção de registros para apenas uma das chaves, reduzindo assim o custo de pré-processamento. Este recurso é eficiente quando se sabe que esta chave é sempre referenciada nas consultas, que a distribuição dos valores das várias chaves é uniforme e que os intervalos das consultas são semelhantes para todas as chaves. Quando isso não ocorre, a existência de várias listas de projeções oferece a vantagem de se poder usar técnicas para selecionar primeiro a chave cujo intervalo vai provavelmente permitir o maior número de eliminações, sem que seja necessário procurar os limitantes inferiores e superiores de todos os

intervalos. Portanto, um ponto a ser discutido neste método é: "que chave escolher primeiro?". Esta escolha é de fundamental importância, pois dela depende o número de registros que vai sobrar para se trabalhar daí para a frente. Estamos então interessados em encontrar meios de escolher a chave que permita fazer o maior número de eliminações nesta primeira fase, para depois aplicar o método sequencial nas demais comparações.

Vamos considerar que os intervalos da consulta tem praticamente o mesmo tamanho e analisar as figuras 3.3 e 3.4:

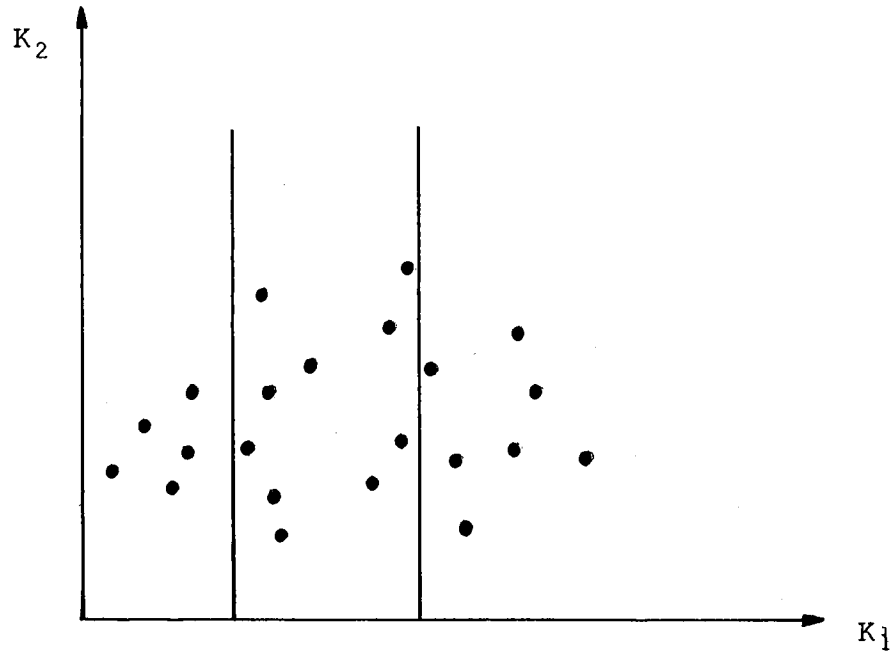


Fig. 3.3.

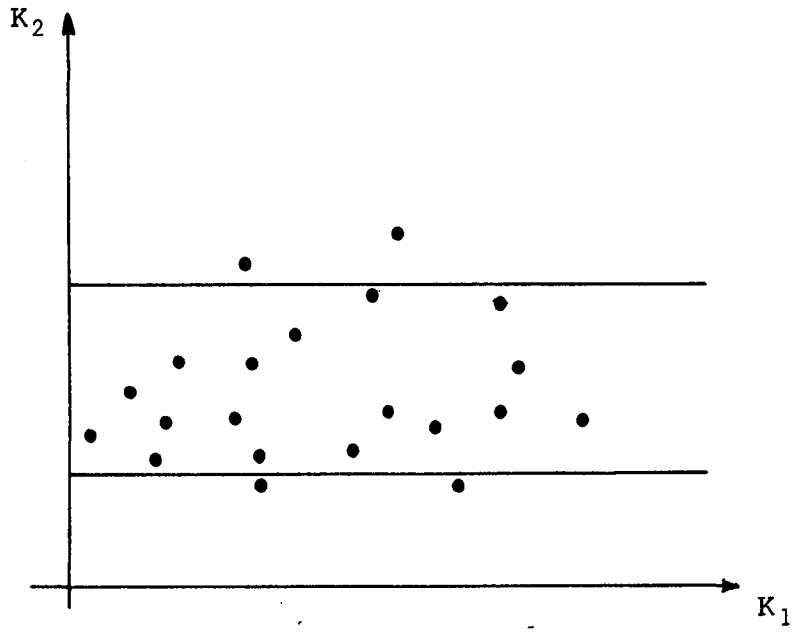


Fig. 3.4

Na figura 3.3, a seleção da chave K_1 permitiu que vários registros (pontos) fossem eliminados. Na figura 3.4, que representa o mesmo conjunto de dados, vê-se que a escolha de K_2 não foi boa, pois muito poucos registros foram eliminados. Logicamente K_1 é a melhor escolha para se fazer a projeção pois restam menos elementos para continuar as comparações. Pode-se observar que os valores de K_1, estão mais espalhados do que os de K_2. Isto sugere que a variância das chaves pode ser uma boa medida (heurística) para se basear o critério de escolha de chaves. Na figura 3.3, a variância de K_1 é bem maior que a de K_2, logo K_1 seria escolhida. Além de escolher a chave de maior variância para fazer a projeção, pode-se ordenar as outras chaves por ordem decrescente de variância para fazer as com

parações sequenciais.

Vamos considerar agora a figura 3.5:

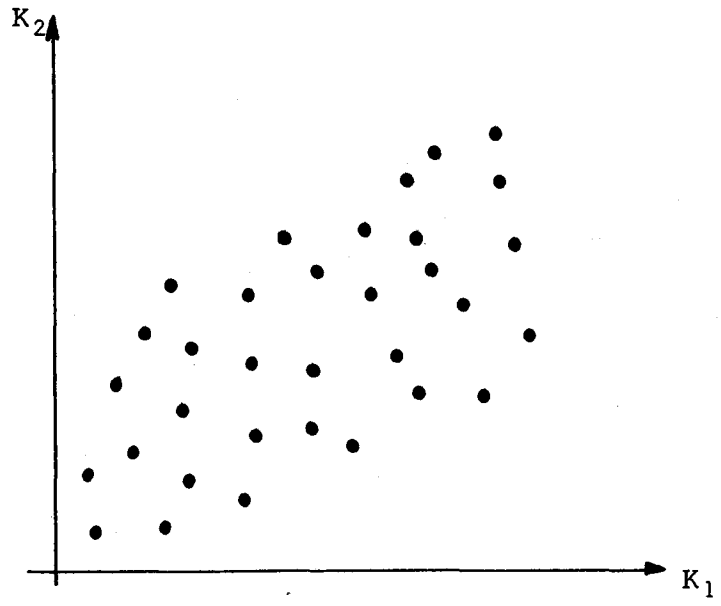


Fig 3.5

Para esse conjunto de dados o método de escolha de chaves baseado na variância não funcionaria bem, pois estas são praticamente a mesma para K_1 e K_2.

Portanto conclui-se que, para o bom funcionamento do método, seria conveniente ter uma ou duas chaves com variância bem maior que as outras. Infelizmente, nem sempre o conjunto de dados possui essa característica, como pode ser observado na figura 3.5. Mas, frequentemente é possível transformar esses dados, através da aplicação da "Análise de Componentes Principais" [LEE -76] que vai criar novas chaves, as quais poderão ter essa propriedade.

3.2.1. Análise de Componentes Principais (ACP)

O uso da Análise de Componentes Principais para se encontrar novas chaves e assim melhorar a busca, implica em um trabalho muito maior de pré-processamento. Nesse pré-processamento devem ser encontrados novos eixos de coordenadas para depois fazer a transformação das chaves.

Colocado mais formalmente, o problema consiste em: dado um conjunto de n pontos com k chaves K_1, K_2,..., K_k encontrar uma direção

$$d = (a_1, a_2, \ldots, a_k), \quad (a_1^2 + a_2^2 + \ldots + a_k^2 = 1)$$

tal que, depois de projetados os pontos em d, a variância dos pontos projetados seja máxima.

Na verdade essa direção d é um autovetor X_1 da matriz de covariância C dos dados, que tem o maior auto vetor λ_1. Além disso, a variância dos pontos projetados é precisamente λ_1. A direção X_1 é chamada primeiro eixo principal.

De maneira genérica, seja F um conjunto de pontos. Seja C a matriz de covariância de F e k a ordem de C.

Sejam

$$X_1, X_2, \ldots, X_k$$

os autovetores de C correspondentes aos autovalores

$$\lambda_1, \lambda_2, \ldots, \lambda_k$$

onde

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$$

e

$$||X_i|| = 1 \qquad 1 \leq i \leq k$$

As propriedades dos X_i são:

1. Todos os X_i's são mutuamente ortonormais, isto é,

$$X_i \cdot X_j = 0 \qquad i \neq j$$

$$X_i \cdot X_j = 1 \qquad i = j.$$

2. Se o conjunto F de pontos é projetado em X_i, a variância dos pontos projetados em X_i é λ_i.

3. Entre todas as direções possíveis, X_1 é a direção que vai produzir a maior variância, projetando-se os pontos nela. X_2 é a direção no espaço perpendicular a X_1 que vai produzir a maior variância projetando-se os pontos nela. Em geral, X_i, $1 < i \leq k$ é a direção no espaço perpendicular a $X_1, \ldots . X_{i-1}$ que vai produzir a maior variância, projetando-se os pontos nela.

Com base nestas propriedades dos X_i's pode-se considerar os X_i's como um novo conjunto de coordenadas no qual pode-se projetar os pontos e obter novas chaves que provavelmente vão diminuir o domínio da busca [SCARPELLI-83].

Por exemplo, no caso da figura 3.5, após aplicar-se ACP ter-se-ia a situação mostrada na figura 3.6.

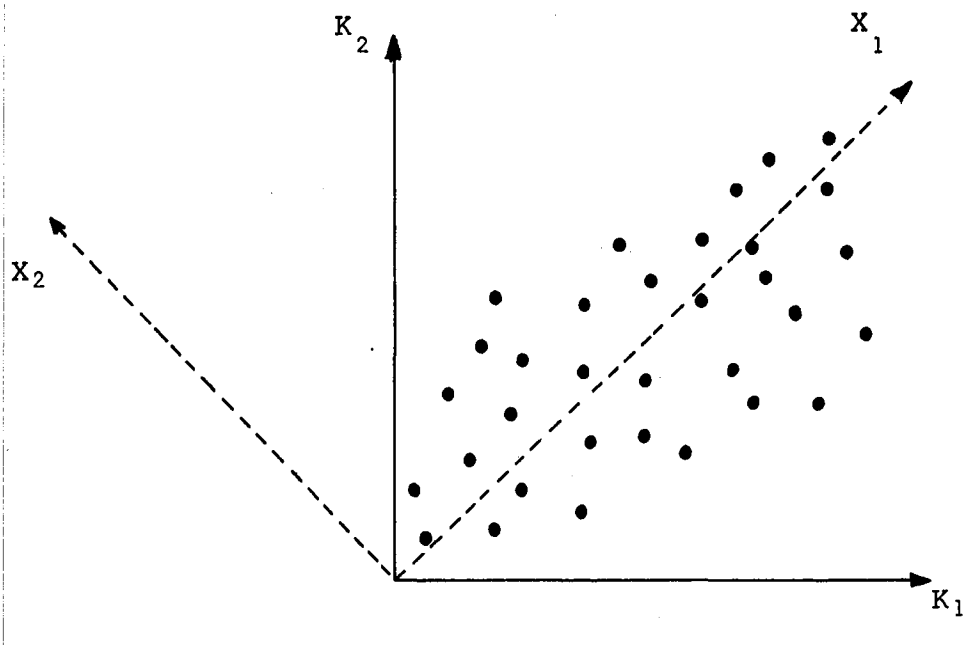


Fig. 3.6

A variância λ_1 dos pontos projetados em X_1 é agora bem maior que a variância λ_2 dos pontos projetados em X_2. Assim, X_1 deve ser escolhida como a coordenada em que os registros serão inicialmente projetados, pois, para intervalos com tamanhos semelhantes, provavelmente será eliminado um maior número de registros.

Assim concluimos que, o método de projeção funciona melhor quando um dos intervalos da consulta permite eliminar a grande maioria de registros.

Na situação em que os intervalos de busca são semelhantes e uma das chaves tenha variância bem maior que as outras é provável que o método de projeção seja eficiente. No caso em que as variâncias são semelhantes, deve-se analisar se o custo do pré-processamento usando ACP é compensado com o ganho

que será obtido na busca.

A técnica de projeção mostra-se igualmente eficiente para responder consultas de outros tipos, como busca exata ou problema do vizinho mais próximo.

A análise de custo da projeção é bastante simples. O pré-processamento consiste em realizar k ordenações de n elementos cada. O armazenamento do arquivo consiste em guardar essas k listas de n elementos. Isso implica que:

$$P_p(n,k) = O(knlog_2n),$$

$$S_p(n,k) = O(kn)$$

Friedman, Baskett e Shustek [FRIEDMAN-75] mostram que para buscas que tem região quase cúbica e encontram um número pequeno de registros (e são portanto similares a busca do vizinho mais próximo) o tempo de busca é dado por

$$Q_p(n,k) = O(n^{1-(1/k)}) \quad \text{(caso médio)}.$$

3.3. Células

O terceiro tipo de estrutura envolve a noção de "células". Para obter as células, faz-se uma partição do k-espaço em hiper-retângulos de lado ℓ, como uma grade no caso de duas dimensões. Os pontos do k-espaço que estão na mesma célula são armazenados juntos em uma implementação.

Por exemplo, considerando-se um mapa de uma cidade, impresso em escala reduzida na forma de livro, a primeira página traz a área toda e as outras páginas trazem detalhes de regiões: quadrados com um certo tamanho, digamos, um kilômetro de lado. Quando se quer saber quais as escolas que se localizam em uma certa região, verifica-se na primeira página quais os quadrados de um kilômetro que se sobrepõem a essa região e procura-se as escolas apenas nas páginas do livro relativas a esses quadrados [BENTLEY-79c] .

Esse é o processo envolvido na utilização da estrutura de células. Os quadrados correspondem a células do k-espaço e a primeira página do livro corresponde a um diretório que permite procurar as células que interessam para a consulta. No procedimento de busca pesquisa-se esse diretório e seleciona-se as células que devem ser verificadas, acessando cada uma delas para comparar seus pontos. Para uma consulta do tipo busca por intervalo, que define uma região do k-espaço, seriam selecionadas as células que se sobrepõem à essa região.

Para o caso de duas dimensões, a estrutura pode ser ilustrada como na figura 3.7:

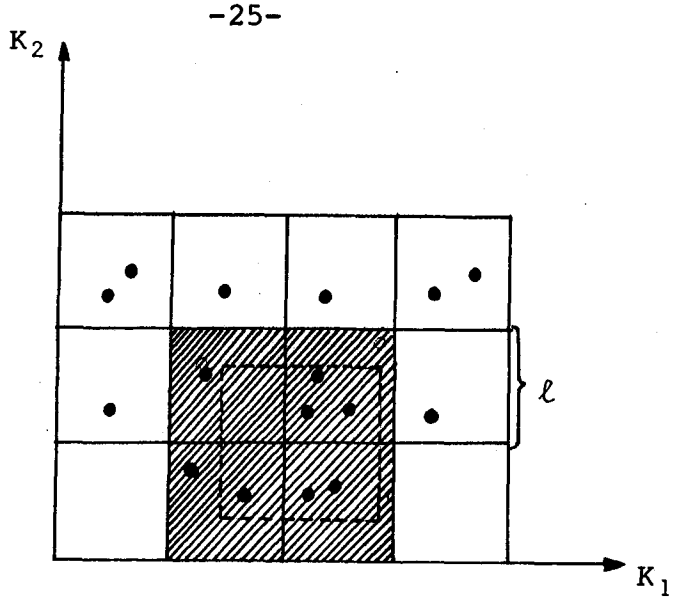


Fig. 3.7

Os pontos representam registros de duas chaves cada um. Os quadrados dessa figura são o diretório, correspondente à primeira página do livro do mapa. Os pontos de cada célula são armazenados juntos na implementação. O retângulo pontilhado representa a região de busca por intervalo, com especificação de intervalo para as duas chaves. Na busca seriam selecionadas as quatro células hachuradas que se sobrepõem ao retângulo, restando a partir daí os oito pontos localizados nessas quatro células para comparação.

Pode-se visualizar o caso genérico, com mais de duas dimensões, considerando que os valores de cada uma das $K_1,...K_k$ chaves são divididos em classes, e cada combinação de classes dessas k chaves forma uma célula. Quando é formulada uma consulta do tipo busca por intervalo, seleciona-se as células que interseccionam o hiper-retângulo dessa consulta.

A seleção de células a partir de uma consulta é chamada de "tradução" de uma consulta para um conjunto de identifi

cadores de células.

A técnica de células é mais adequada para casos em que são especificados intervalos para muitas chaves. Seu funcionamento é mais satisfatório quando se sabe que o tamanho e forma das consultas são praticamente constantes e conhecidos. Assim, as células podem ser definidas com tamanho semelhante ao das consultas. Para consultas com tamanho e forma que variam muito e são diferentes do tamanho das células, esse método pode não ser muito eficiente.

Para outros tipos de consultas o método de células mostra-se também adequado. Para busca exata, seu desempenho é muito bom quando as células são de tamanho pequeno. Para o problema do vizinho mais próximo com raio fixo, isto é, o problema de encontrar o vizinho mais próximo de um ponto dado dentro de um raio p, são selecionadas 3^k células para comparação, como mostra a figura 3.8 para k = 2.

Dado um ponto P, seu vizinho mais próximo localiza-se em uma das células adjacentes aquela em que está o ponto dado, ou nela mesma [BENTLEY-77].

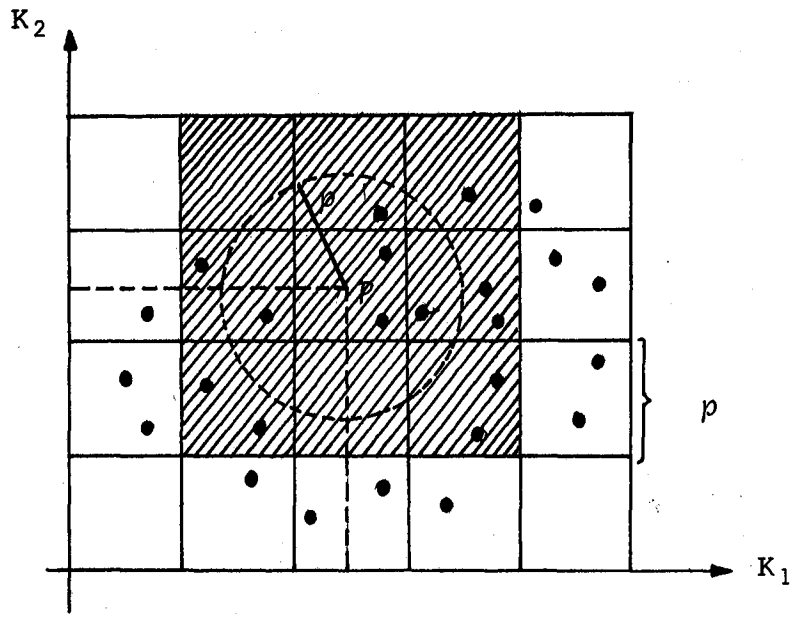


Fig. 3.8

A maneira mais simples de se armazenar um arquivo para usar a técnica de células é usar uma matriz multidimensional para o diretório, que contem ponteiros para as listas onde estão os pontos das células correspondentes.a cada diretório. Esta forma de armazenamento funciona muito bem quando se sabe que os pontos estão uniformemente distribuídos no k-espaço [BENTLEY-79c] .

Por exemplo, se os pontos estão uniformemente distribuidos em $[0,10]^2$ e considerando células unitárias, referenciadas por $[i, i+1]x[j, j+1]$ para $i,j = 0,9$, como é ilustrado na figura 3.9,

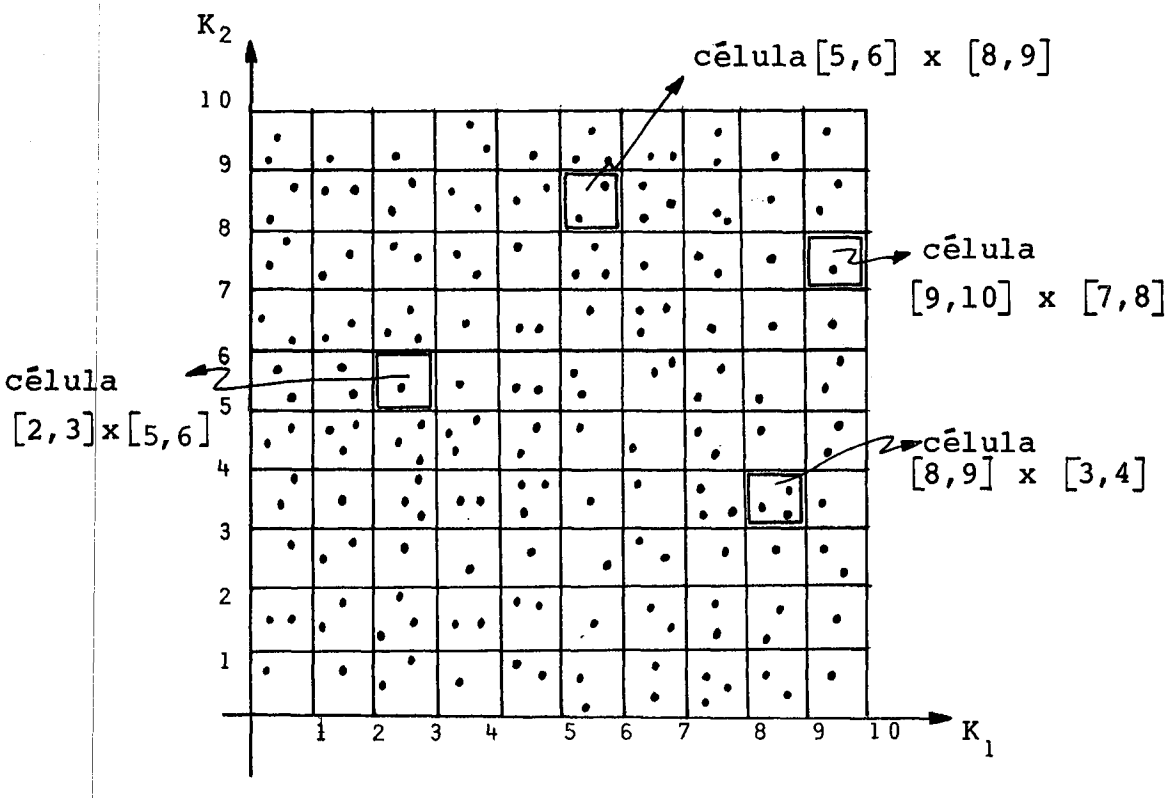


Fig. 3.9

o diretório pode ser implementado na matriz:

DIRETÓRIO [0..9, 0..9]

e DIRETÓRIO[i,j] contém um ponteiro para a lista de todos os pontos na célula [i, i+1] x [j, j+1]. Isto é exemplificado na figura 3.10, onde a ligação terra (⏚) representa um ponteiro nulo.

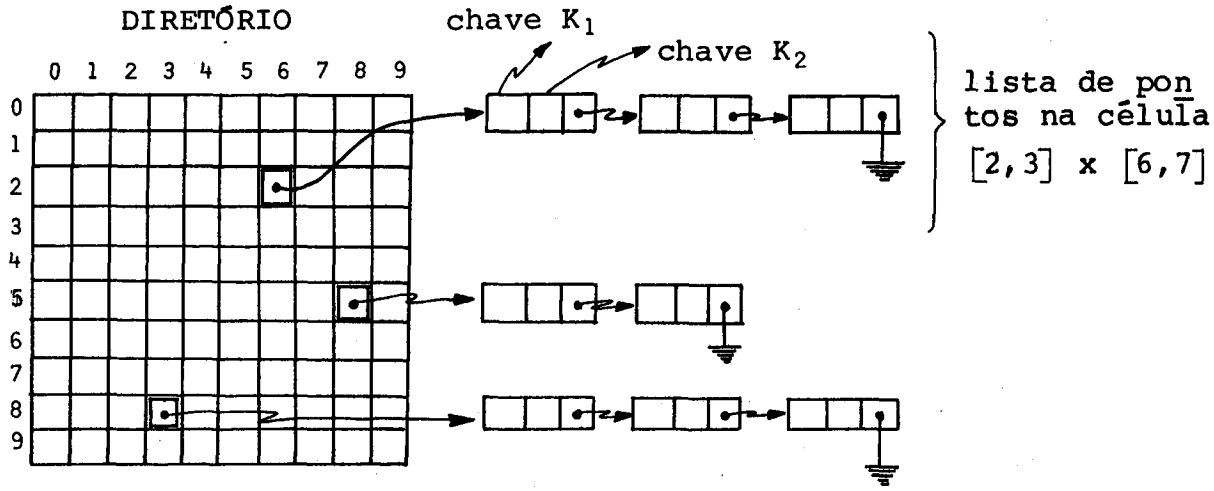


Fig. 3.10

Nesta implementação, todas as posições da matriz DIRETÓRIO contêm ponteiros para listas de pontos. Na figura 3.10 vê-se em destaque a posição DIRETÓRIO[2,6] que dá origem à lista dos pontos na célula [2,3] x [6,7], a qual contém três elementos. As posições DIRETORIO [5,8] e DIRETÓRIO [8,3] apontam para as listas de pontos nas células [5,6] x [8,9] e [8,9] x[3,4] respectivamente.

Se uma busca requer registros que verifiquem as condições

$$7.8 \leq K_1 < 9.2$$

e

$$5.3 < K_2 \leq 6.7$$

a tradução destes intervalos para o conjunto de identificadores de células mostra que deverão ser examinadas as células identificadas pelas posições

[7,5], [7,6], [8,5], [8,6], [9,5], [9,6]

da matriz DIRETÓRIO, ou seja, as células:

[7,8] x [5,6]	[8,9] x [6,7]
[7,8] x [6,7]	[9,10] x [5,6]
[8,9] x [5,6]	[9,10] x [6,7]

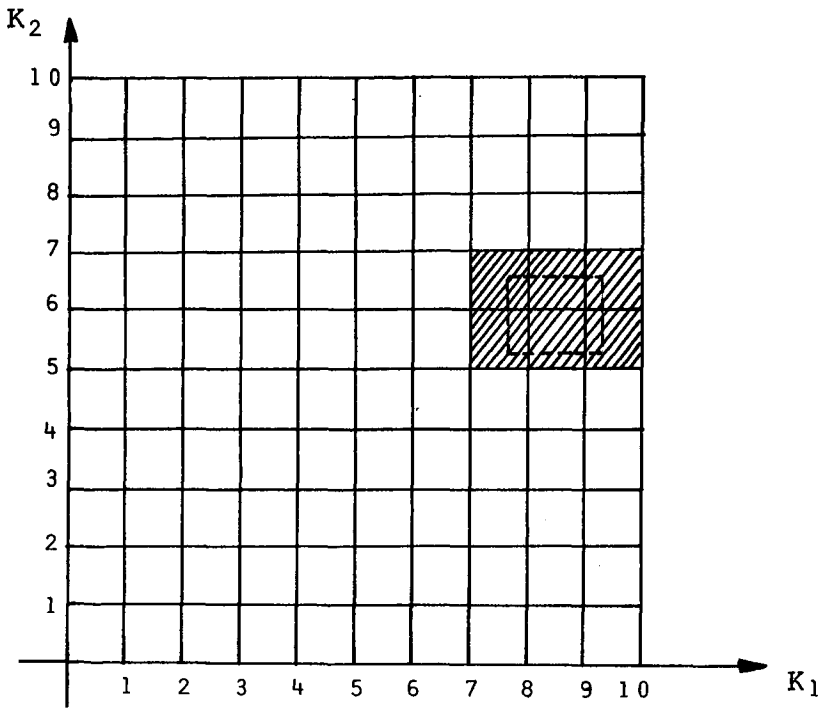


Fig. 3.11

A figura 3.11 ilustra esse exemplo de busca, onde o retângulo pontilhado representa a região de busca e as células hachuradas são as que serão examinadas.

Quando a distribuição das chaves não é uniforme no k-espaço, pode-se usar uma técnica do tipo hash para o diretório [YUVAL-75]. Também neste método, a célula (i,j) aponta para a lista de todos os pontos em $[i, i+1] \times [j, j+1]$, porém, somente as células que contem registros do arquivo são armazenadas. O DIRETÓRIO $[d_i, d_j]$ aponta para a lista de todas as células (i,j) tal que:

d_i = função hash(i)

d_j = função hash(j) com $i = \lfloor K_1 \rfloor$, $j = \lfloor K_2 \rfloor$

Por exemplo, considerando células unitárias e

$d_i = \lfloor K_1 \rfloor \bmod 7$

$d_j = \lfloor K_2 \rfloor \bmod 11$

o diretório pode ser implementado na matriz:

DIRETÓRIO [0..6, 0..10]

como mostra a figura 3.12, onde os identificadores de células $(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)$ que estão na lista apontada por DIRETÓRIO $[d_i, d_j]$ são tais que

$d_i = \lfloor K_{is} \rfloor \bmod 7$

$d_j = \lfloor K_{js} \rfloor \bmod 11$

$s = 1, \ldots, r$

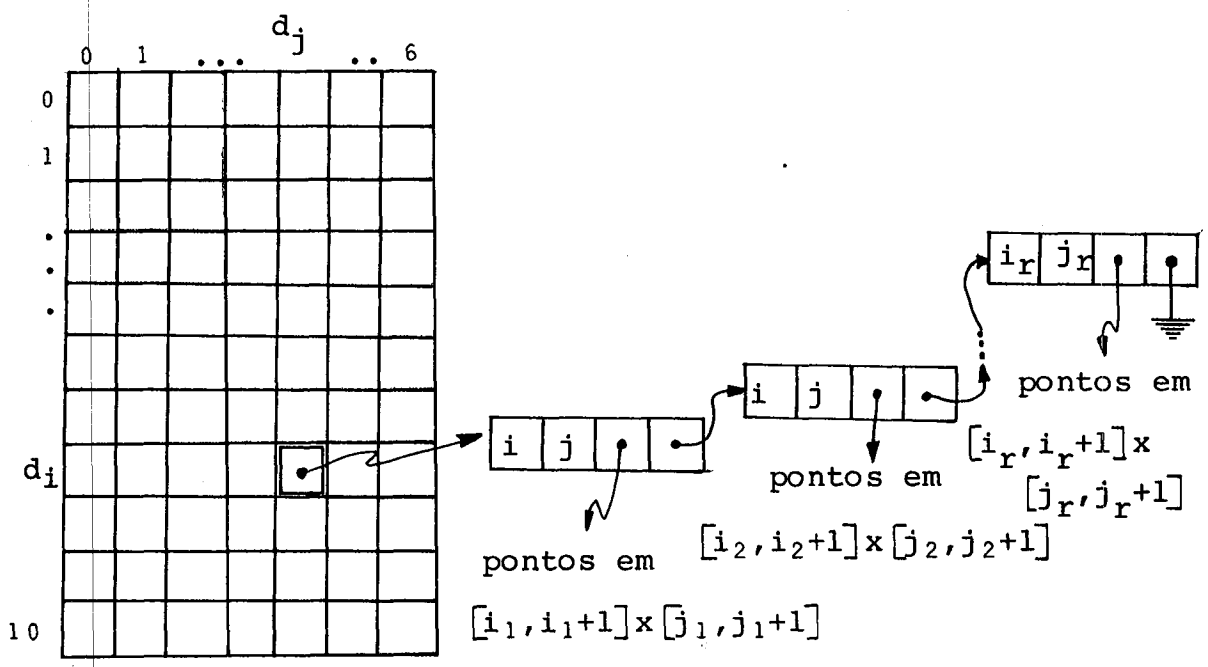


Fig. 3.12

Para qualquer uma dessas duas formas de armazenamento, o espaço necessário consiste das listas representando os pontos das células e o diretório e o pré-processamento resume-se na montagem do diretório e construção das listas. Com isso conclui-se que

$$P_c(n,k) = O(kn)$$

e

$$S_c(N,K) = O(kn)$$

Como as características básicas da estrutura de células são o tamanho e a forma das células, a escolha desses parâmetros será decisiva para a análise de busca. Para tanto considera-se o número de acessos a células e testes de inclusão. Se o tamanho da célula é muito grande, haverá poucos acessos e mui

tos testes de inclusão. Se o tamanho da célula é muito pequeno, haverá muitos acessos a células e poucos testes de inclusão. Logicamente, ambos os extremos devem ser evitados.

O melhor tamanho e forma para as células depende do tamanho e forma do hiper-retângulo da consulta. Nas aplicações em geral, as consultas variam em tamanho e forma, bem como em localização, logo existe grande dificuldade para se fazer uma boa escolha de forma e tamanho para as células. Entretanto pode-se apresentar alguns resultados de análise para o caso particular em que o hiper-retângulo das consultas tem tamanho e forma constantes, variando apenas sua localização no espaço de coordenadas. Neste caso, o tamanho e forma ideais para as células seriam os mesmos do hiper-retângulo da consulta. Com isso o número de acessos a células seria de 2^k que é o número máximo de células que o hiper-retângulo pode sobrepor, como se pode ver na figura 3.13, para registros com duas chaves:

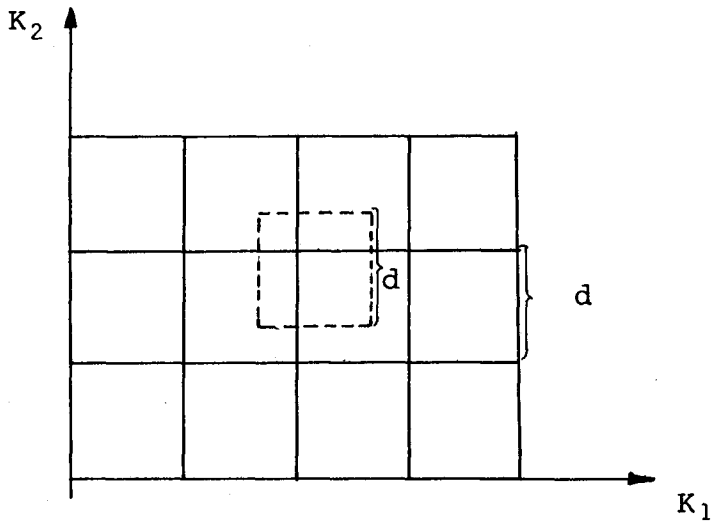


Fig. 3.13

Para completar a análise de custo da busca, é necessário considerar ainda o número de testes de inclusão de um registro na região de busca. Com base na análise de Bentley, em [BENTLEY-77], conclui-se que, quando o conjunto de pontos é esparso, isto é, cada célula contém no máximo um número constante c de pontos, o número máximo de testes de inclusão é $c2^k$. Quando a condição de esparsidade não vale, ou seja, cada célula contém um número arbitrário de pontos, o número de testes de inclusão será, em média, limitado por 2^k vezes o número de pontos na região. Assim, a análise da estrutura de células tem os resultados:

$$P_C(n,k) = O(kn)$$

$$S_C(n,k) = O(kn)$$

$$Q_C(n,k) = O(2^k F') \quad \text{(caso médio)}$$

onde F' é o número de registros encontrados.

4. ÁRVORES QUAD

A estrutura de árvores Quad é uma generalização de árvore binária, adequada para o tratamento do problema de busca por chaves múltiplas.

Na árvore binária unidimensional, cada nó subdivide a coleção de nós que estão na subárvore da qual ele é raiz em duas subcoleções determinando, com base no valor da sua chave, quais nós ficam na sua subárvore esquerda ou direita. Em uma árvore Quad de k dimensões, que representa um conjunto de registros com k chaves, cada nó subdivide a coleção de nós que estão na subárvore da qual ele é raiz em 2^k subcoleções determinando, com base nos valores de suas k chaves quais nós ficam em cada uma das suas 2^k subárvores. Assim, cada nó terá no máximo 2^k filhos. Nos procedimentos de inserção ou busca, as comparações são feitas com base nos k valores das chaves de cada nó, para dedidir sobre qual subárvore descer para fazer o próximo teste.

Esta estrutura será primeiramente descrita para o caso de duas dimensões ou seja, para registros com duas chaves, pois os conceitos básicos podem ser facilmente estendidos para mais dimensões [FINKEL-74].

Cada nó dessa árvore de duas dimensões representa um registro com duas chaves e tem no máximo quatro filhos. Considerando-se os registros de duas chaves como pontos no plano, e se o ponto A é a raiz da árvore, essa raiz divide o espaço onde se localizam os registros em quatro quadrantes: NE, NO, SO, SE, que chamaremos de quadrantes 1, 2, 3 e 4 respectivamente. Estes quadrantes são limitados pelas linhas que passam pelo ponto A, paralelas aos eixos K_1 e K_2, como é mostrado na figura 4.1.

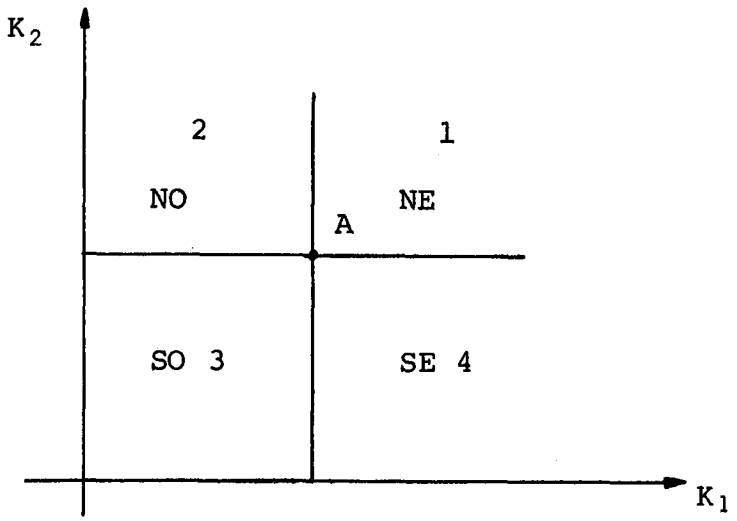


Fig. 4.1

Cada filho do nó A pertece a um destes quadrantes e é a raiz da subárvore que representa esse quadrante.

Os pontos que se localizam no quadrante 1 ficarão na primeira subárvore de A, os pontos do quadrante 2 ficarão na segunda subárvore de A e assim por diante:

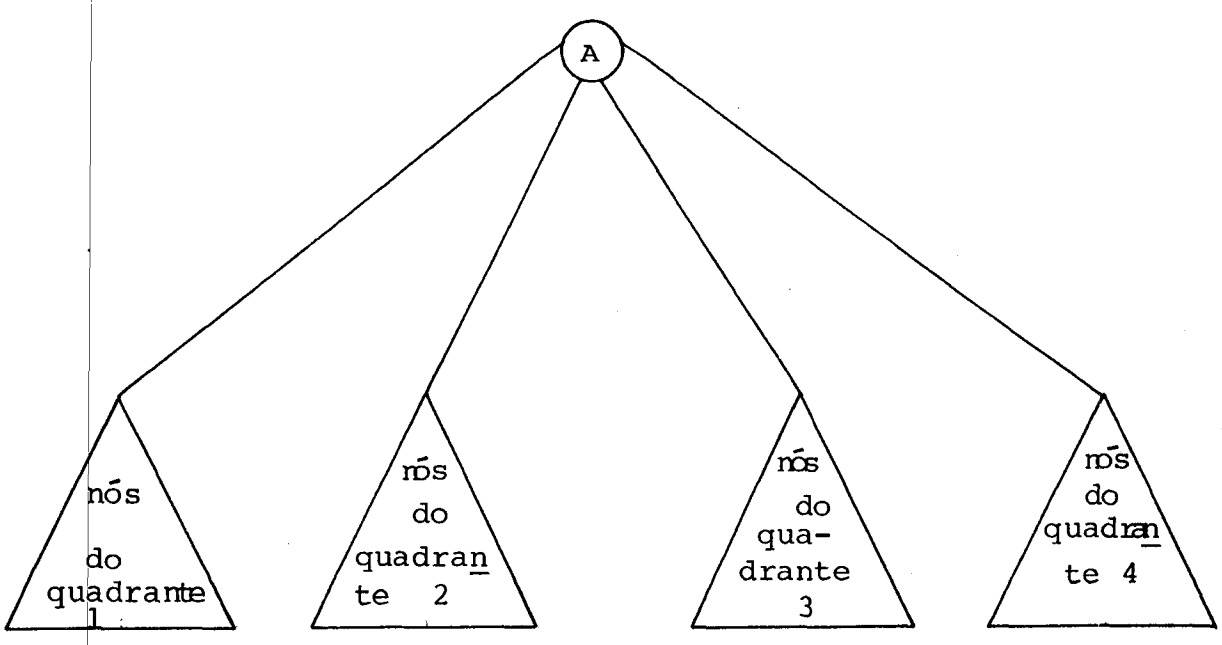


Fig. 4.2

Cada quadrante é subdividido em outros quatro, pelas linhas que passam sobre o nó que é a raiz da subárvore que representa este quadrante, como mostra a figura 4.3:

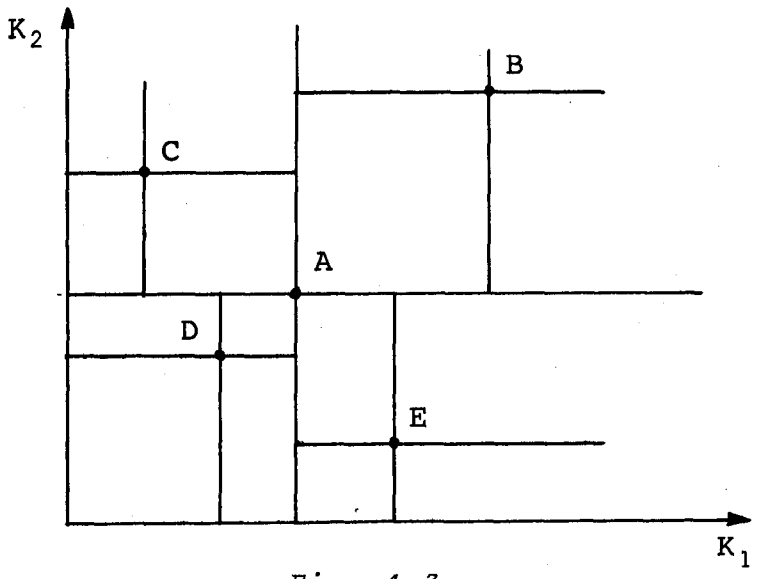
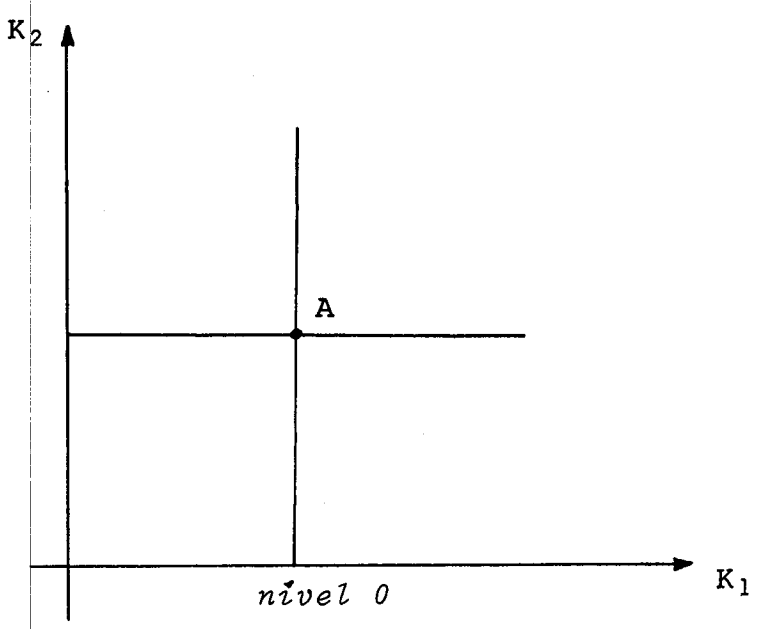


Fig. 4.3

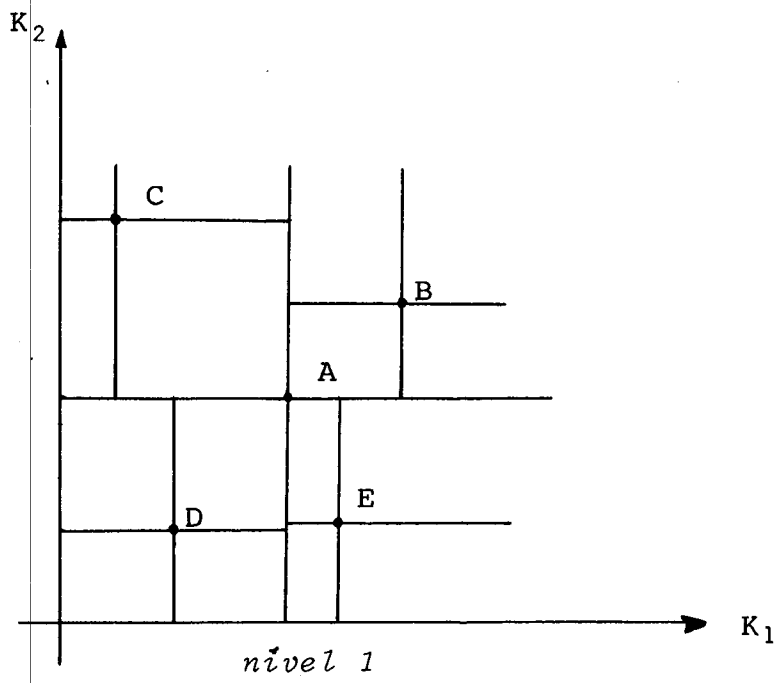
Cada filho do nó A terá no máximo quatro filhos, e assim sucessivamente.

Em geral, cada nó da árvore tem associado a ele uma região retangular do espaço, que contém todos os pontos que estão na subárvore que possui esse nó como raiz. Este retângulo é determinado por quatro linhas retas paralelas aos eixos K_1 e K_2 é chamado de "retângulo limite do nó". Os nós de um determinado nível representam uma partição do espaço e a cada nível seguinte cada um desses subconjuntos é subdividido em outros quatro.

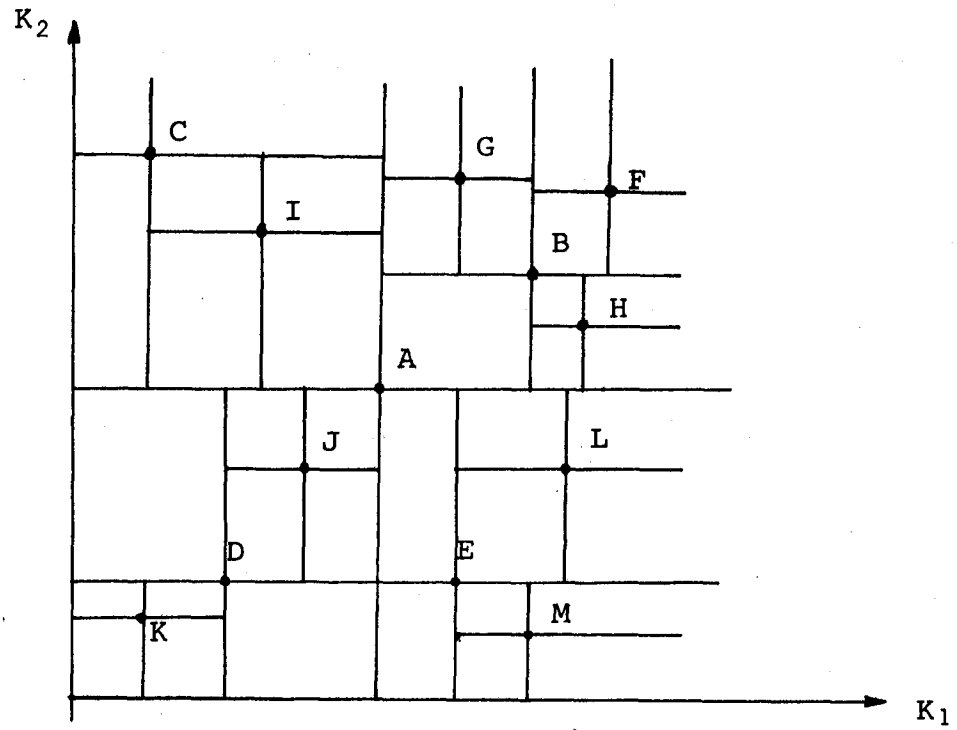
As figuras 4.4 e 4.5 fornecem um exemplo de um conjunto de pontos representado no plano e em forma de árvore Quad. Os nós quadrados da figura 4.5 representam nós externos, que correspondem a subárvores vazias.



(a)

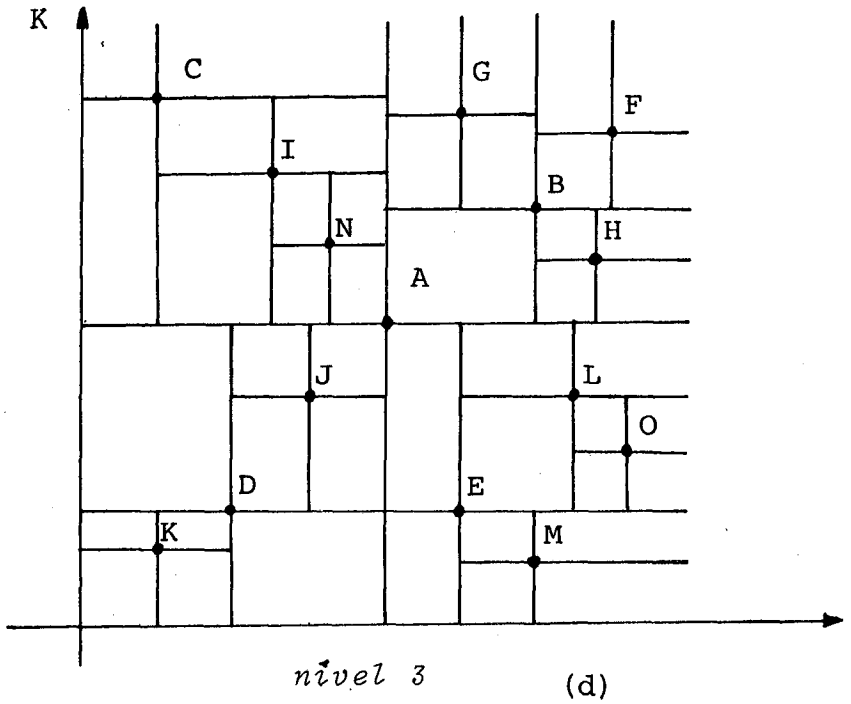


(b)



nível 2

(c)



nível 3

(d)

Fig. 4.4

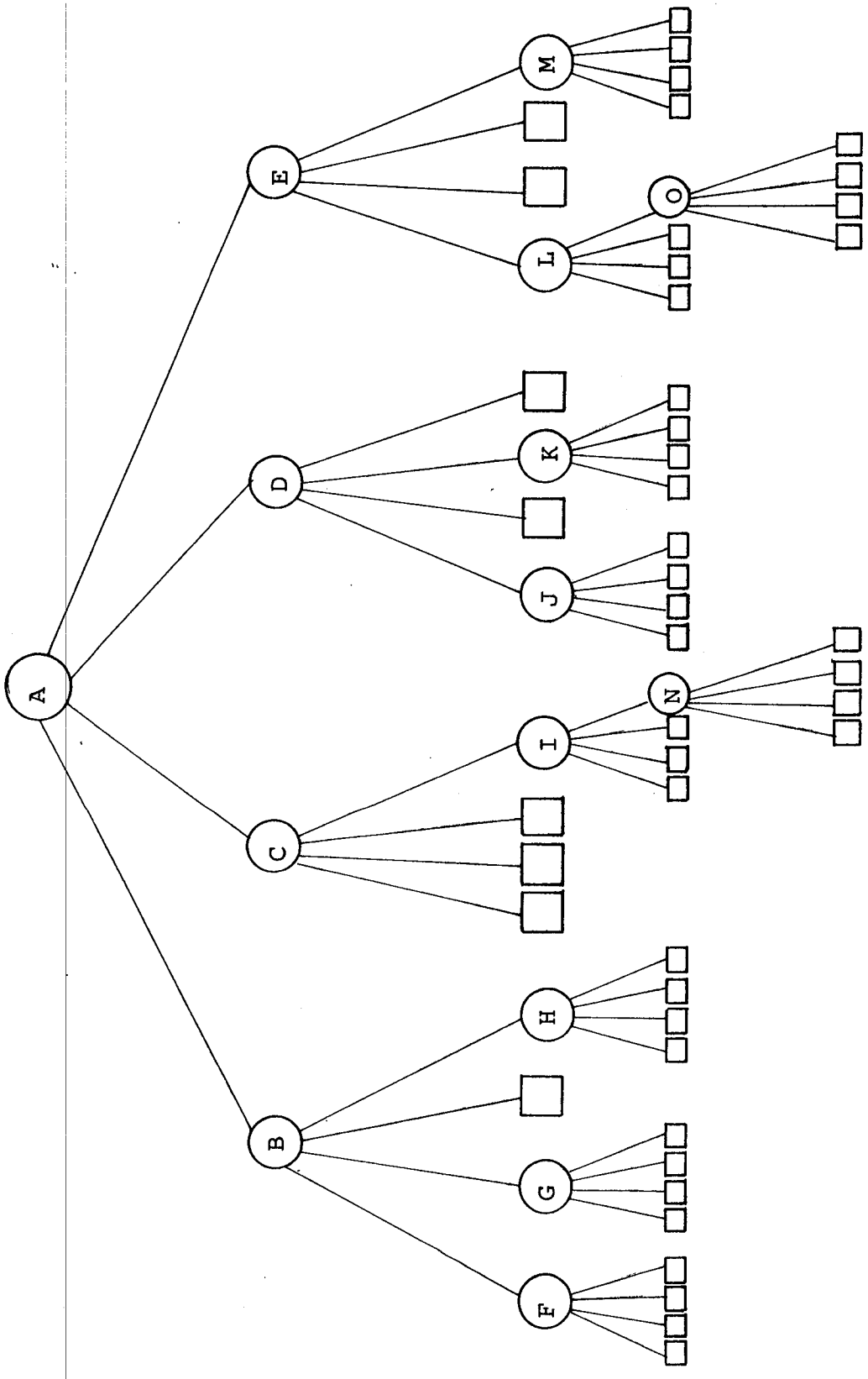


Fig. 4.5

Adota-se a convenção de quadrantes 1 e 3 serem fechados e quadrantes 2 e 4 serem abertos, para resolver casos em que os pontos se localizem sobre a linha que sai de um nó. Por exemplo, se a raiz da árvore é A e um ponto B tem a mesma coordenada K_2 de A, na posição mostrada na figura 4.6, convenciona-se que B pertence ao quadrante 1 de A.

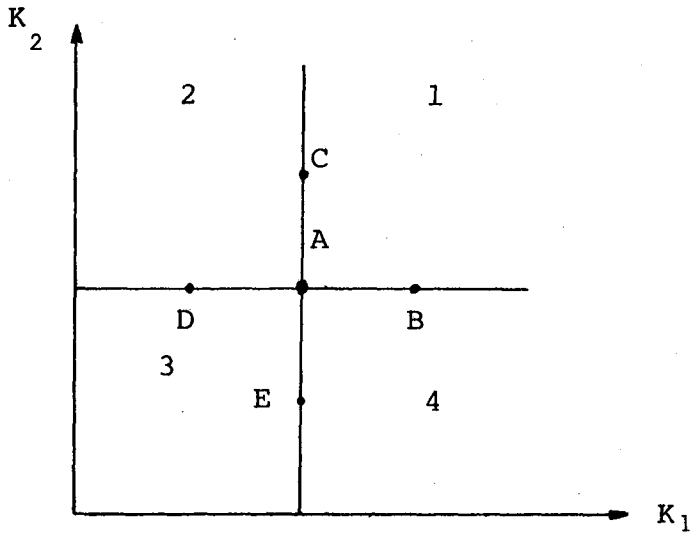


Fig. 4.6

Analogamente, C pertence ao quadrante 1 e D e E pertecem ao quadrante 3.

Além disso, não estão sendo considerados nesta discussão os casos de colisões, isto é, mais de um registro com valores iguais. Quando isso ocorre, esses nós necessitam de um tratamento especial dentro da estrutura, como por exemplo, listas encadeadas, onde cada nó seria um ponteiro para essa lista que conteria todos os registros com valores iguais, ou então algum outro tipo de estrutura mais sofisticada. Aqui, quando fala

se em "registros com valores iguais" ou "registros repetidos", refere-se a registros cujos campos de chaves são iguais, ignorando-se os valores de outros atributos que possam eventualmente existir.

4.1. Inserção

O processo de inserção de um novo registro numa árvore Quad é basicamente o mesmo usado para árvores binárias. A cada nó, começando pela raíz, uma comparação é feita com o nó que vai ser inserido, e o descendente apropriado é escolhido para o próximo teste. Ao ser encontrada uma subárvore vazia, está localizado o ponto onde deve ser inserido o registro. Ao contrário das árvores binárias, a comparações é feita com base nas k chaves do registro, em vez de uma só e o resultado da comparação será um dos k filhos do nó que está sendo visitado. A seguir, apresentamos a implementação de um algoritmo que realiza a inserção, na linguagem PASCAL para o caso k = 2.

4.1.1. Algoritmo de Inserção

O procedimento INSERIR que será detalhado faz a inserção de um novo nó em uma árvore Quad de duas dimensões [FINKEL-74] . Para a construção inicial da árvore, ele seria chamado uma vez para cada nó que deve ser inserido. Nesta implemen

tação, no caso de já existir na árvore um nó com os mesmos valores de chaves que o nó que se quer inserir, a inserção não é feita, uma mensagem é enviada e a rotina termina.

Para descrever este procedimento, será adotada uma estrutura de dados com as seguintes características:

- os nós são registros com dois campos reais para as chaves e cinco campos de ponteiros, indexados de 0 a 4. Os ponteiros de 1 a 4 apontam para cada uma das subárvores do nó, e são nulos quando a subárvore é vazia. O ponteiro zero tem sempre valor nulo.
- os ponteiros são do tipo NO, a ser definido. Os campos dos ponteiros de um nó X, são denotados por X↑.SUB[I] para $0 \leq I \leq 4$. Assim, a terceira subárvore do nó X é acessada por X↑.SUB[3]. Como X↑.SUB[0] tem sempre o valor *nil*, este será o artifício usado para interromper a iteração quando o nó já está na árvore.
- os campos das chaves do nó X são denotados por X↑.CHAVE [I], para I = 1,2.
- o registro tem a forma:

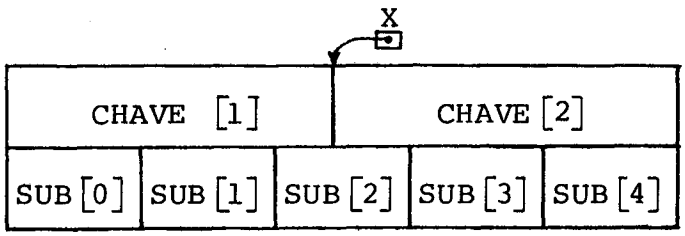


Fig. 4.7

- não serão representados outros campos de atributos, além das chaves.
- a figura 4.8 permite visualizar parte da estrutura de dados no momento em que o nó X vai ser inserido na árvore com raiz apontada por RAIZ. A ligação terra (⏚) representa ponteiro nulo.

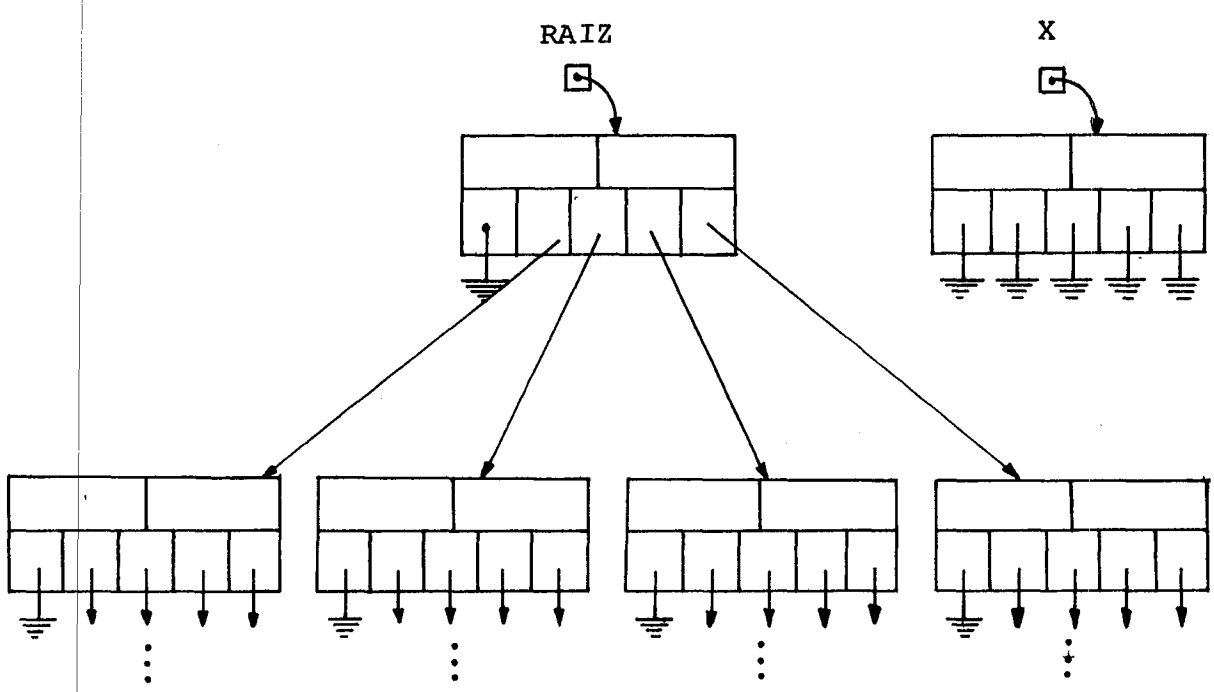


Fig. 4.8

Assim, a definição de tipos e declaração de variáveis, pode ser detalhada como:

```
type  NO = ↑REGISTRO;
      REGISTRO = record
                   CHAVE: array [1..2] of real;
                   SUB: array [0..4] of NO
                   end;
var   RAIZ, X : NO;
```

O procedimento INSERIR faz uso de uma função e de outro procedimento que são definidos da seguinte forma:

- a função QUADRANTE (R,X) onde R e X são ponteiros para nós da árvore, retorna um inteiro de 1 a 4, dependendo de qual quadrante de R, X se localiza. Quando o nó já está na árvore, a função retorna valor 0, isto é, define-se

  QUADRANTE (X,X) = 0.

- O procedimento NO_JA_ESTA_NA_ARVORE(X), onde X é um ponteiro para um nó, envia a mensagem adequada antes de terminar a rotina, quando se tenta inserir um nó que já está na árvore.

Considerações importantes:

- os parametros de INSERIR são R e X, onde R é a raiz da árvore e X é o nó que vai ser inserido.

- os campos de CHAVE de X estão inicialmente com os valores das chaves de X e os campos de ponteiros estão com o valor *nil*.

- quando a árvore está inicialmente vazia o primeiro nó é inse

rido como raiz, sem entrar no processo de iteração.

- supondo que RAIZ aponta para a raiz da árvore e X aponta para o nó que vai ser inserido, o procedimento é chamado com os parâmetros INSERIR (RAIZ, X).

```
procedure INSERIR (var R:NO; X:NO);
(* inserir registro X na árvore cuja raiz é R *)
var L: 0..4;
    Q: NO;
begin
if R = nil then (*árvore vazia*) R:=X
          else begin
                 Q:= R;
                 L:= QUADRANTE (Q,X) ;
                 (*se nó está na árvore L=0 e
                          Q↑.SUB[L] = nil *)
                 while Q↑.SUB[L] <> nil
                 do begin (*cada iteração desce um nível*)
                       Q:= Q↑.SUB[L];
                       L:= QUADRANTE (Q,X)
                    end;
                 if L <>0 then (*inserir no*)
                              Q↑.SUB[L]:= X
                          else NO_JA_ESTA_NA_ARVORE(X)
               end
end ;
```

Esse algoritmo é usado em [FINKEL-74] , que obteve alguns resultados empíricos sobre seu funcionamento. Foram construidas árvores de vários tamanhos, com os nós uniformemente distribuidos nas duas coordenadas, com valores das chaves variando de 0 a $2^{31}-1$. A tabela 4a resume os resultados obtidos.

TABELA 4a

Resultados Obtidos nos Testes de Construção de Árvores Quad

Número de Nós	Comprimento de Caminho Interno			X	Número de Árvores
	Mínimo	Médio	Desvio Padrão		
25	48	67.21	8.699	0.8352	300
50	123	168.4	17.34	0.8608	300
100	288	403.5	30.68	0.8763	150
1000	4547	6288	325.6	0.9103	30
10000	62721	84705	2882	0.9197	10

A coluna "mínimo" mostra o comprimento de caminho interno para uma árvore balanceada com o respectivo número de nós, portanto, o menor comprimento de caminho interno que essa árvore pode ter [KNUTH-68].

A coluna X é o valor médio obtido pela divisão do comprimento de caminho interno médio das árvores testadas por $n\log_e n$, onde n é o número de nós da árvore e o logaritmo natural é usado. Como seus valores crescem muito vagarosamente, isso implica que o comprimento de caminho interno de uma árvore Quad construida com inserção randônica é aproximadamente propor

cional a $n \log_e n$. Logo, a busca exata vai provavelmente exigir cerca de $\log_e n$ testes. O pior caso de inserção e busca em uma árvore Quad é $O(n^2)$ e isto acontece quando cada nó é inserido como filho do nó que está mais abaixo na árvore. Nesse caso o comprimento de caminho interno será $n(n-1)/2$.

4.2. Busca

Para realizar a busca por intervalo em uma árvore Quad, usa-se um processo que consiste de comparações entre a região de busca e o retângulo limite dos nós. Ainda considerando o caso de duas dimensões e supondo que foi feita uma consulta determinando uma região retangular de busca, segue-se os seguintes passos:

Iniciando-se pela raiz, compara-se, a cada nó, se este está dentro da região de busca, processando-o ou não, dependendo do resultado da comparação.

Em seguida compara-se a região de busca com o retângulo limite de cada um dos filhos desse nó, para decidir na direção de quais deles deve-se dirigir para continuar a busca. Se o retângulo limite de um nó se sobrepõe à região de busca, esse nó deve ser visitado recursivamente pelo processo [FINKEL-74].

Por exemplo, considerando-se a região de busca da figura 4.9 e sendo A a raiz da árvore;

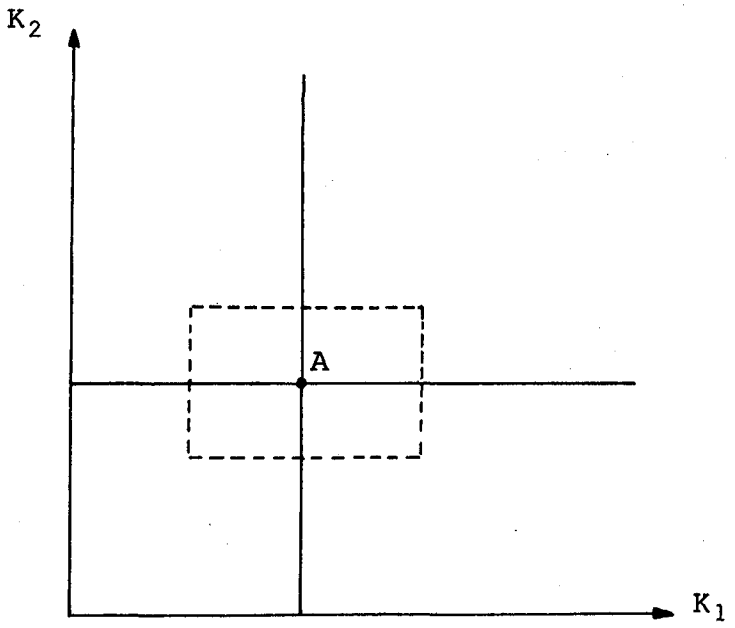


Fig. 4.9

no primeiro teste conclui-se que A se localiza na região de busca, então ele é um dos pontos que estão sendo procurados. Além disso a região se sobrepõe ao 1º quadrante de A, logo, a primeira subárvore de A pode ter nós que satisfaçam a condição de busca. Por isso essa subárvore deve ser pesquisada. O mesmo acontece com os outros três quadrantes de A, portanto as quatro subárvores de A devem ser visitadas e o processo é aplicado recursivamente às raízes dessas subárvores.

Se no próximo nível a árvore contém os nós B, C, D e E nas posições da figura 4.10, pode-se ver quais subárvores destes nós serão visitadas.

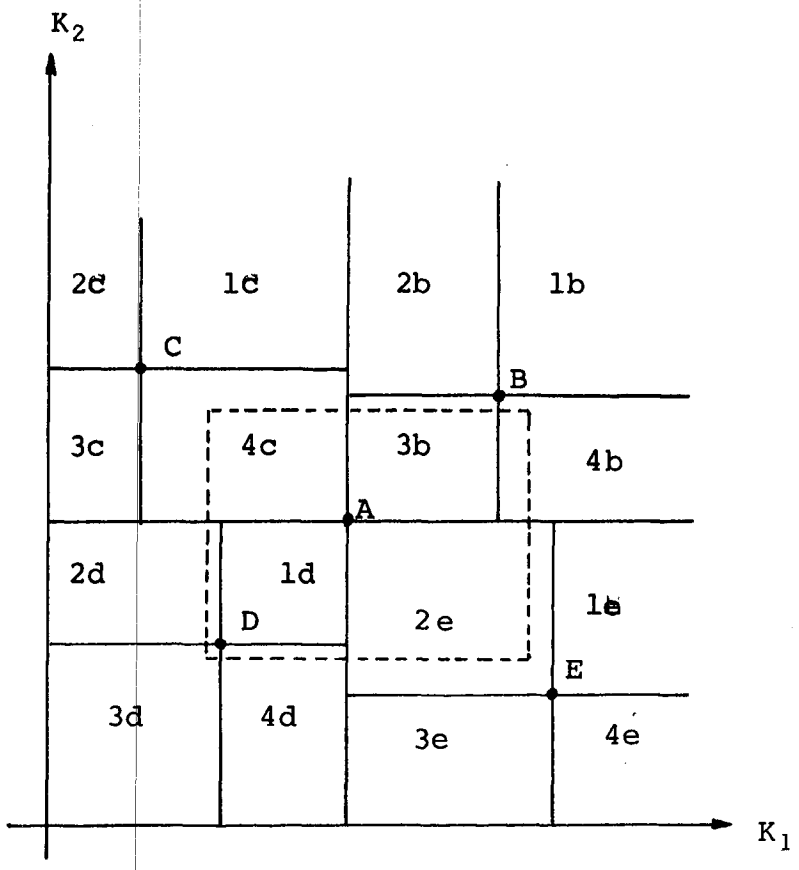


Fig. 4.10

Na busca sobre o nó B, conclui-se que B não está na região e somente os quadrantes 3 e 4 de B (3b e 4b da figura 4.10) se sobrepõem à região. Assim, somente as subárvores 3 e 4 de B serão visitadas. Da mesma forma serão visitadas a subárvore 4 do nó C, equivalente a 4c na figura 4.10, as subárvores 1, 2, 3 e 4 do nó D equivalentes a 1d, 2d, 3d, 4d na figura 4.10 e a subárvore 2 do nó E, equivalente a 2e na figura 4.10.

Descendo mais um nível e supondo que os nós estejam dispostos como na figura 4.11,

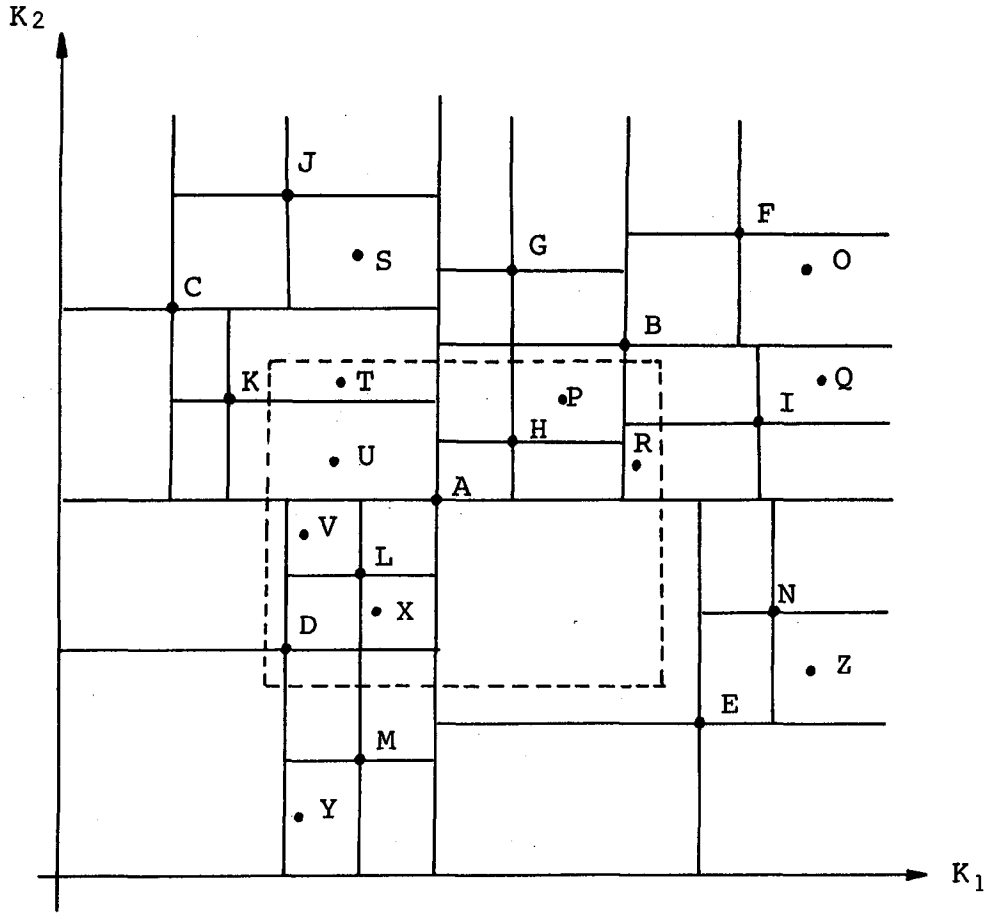


Fig. 4.11

serão visitadas:

subárvores 1, 2, 3 e 4 do nó H

subárvores 2 e 3 do nó I

subárvores 1 e 4 do nó K

subárvores 1, 2, 3 e 4 do nó L

subárvores 1 e 2 do nó M

e os nós encontrados por essa busca serão; A, D, H, L, P, R, T, U, V e X.

A figura 4.12 mostra esses pontos na estrutura de ár

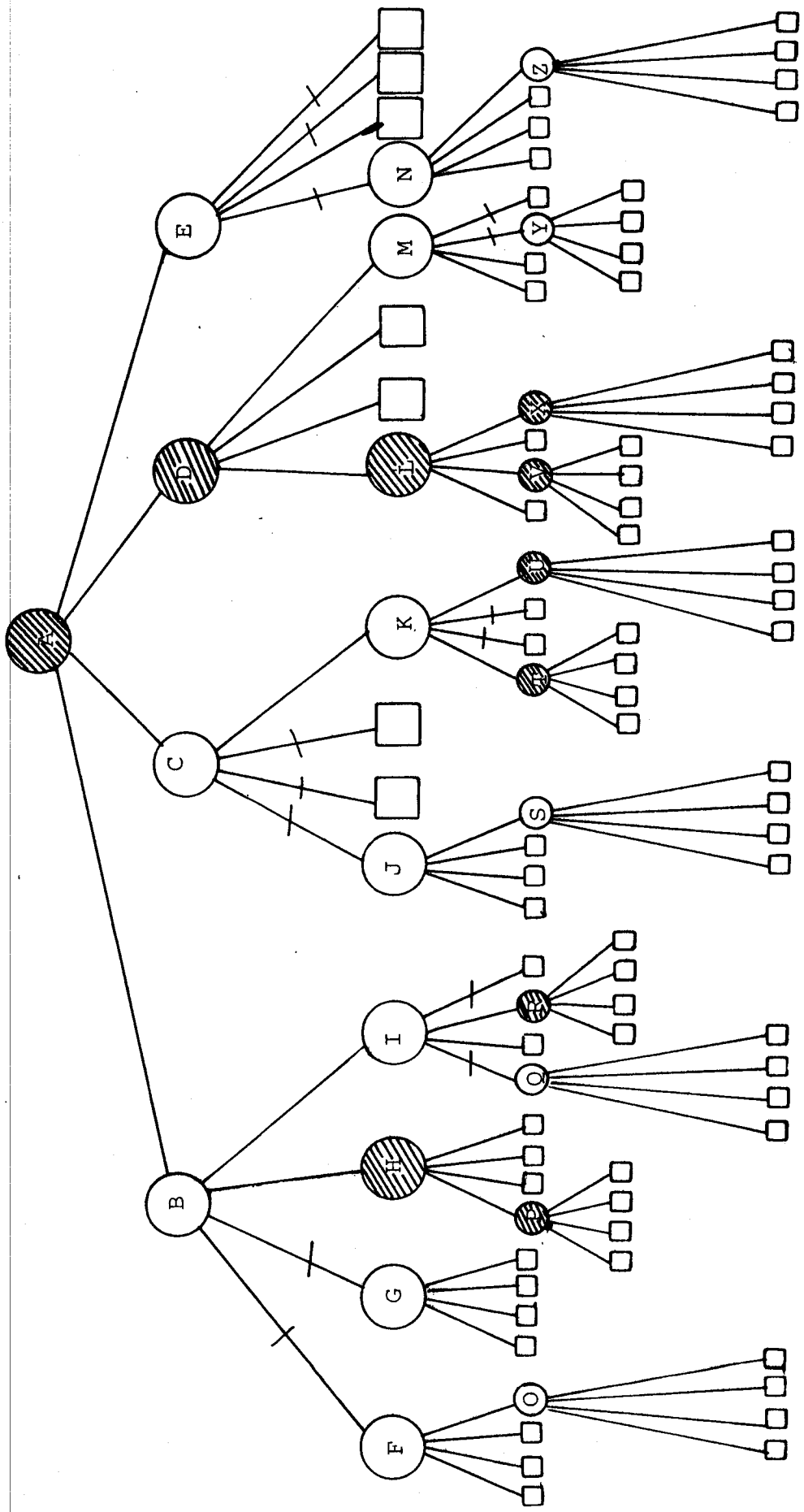


Fig. 4.12

vore e indica em que ramos a busca continua ou é interrompida. O traço perpendicular sobre um ramo indica que a busca foi interrompida, e os nós hachurados são os que se localizam na região de busca. Os nós quadrados representam subárvores vazias.

4.2.1. Algoritmo de Busca

Será detalhado a seguir, o procedimento BUSCA-INTERVALO de [FINKEL-74] que executa a busca por intervalo em uma árvore Quad de duas dimensões. Para isso são necessárias algumas considerações.

Já foi visto que o procedimento para busca por intervalo envolve dois tipos de comparações:

1) O teste de inclusão de um ponto na região de busca, e
2) O teste de sobreposição da região de busca com um retângulo limite.

Para estes testes são definidas as funções booleanas:

- ESTA_NA_REGIÃO (X), onde X é o ponteiro para um nó da árvore, retorna "true" se esse nó está na região e "false" caso contrário.
- RETANGULO_SOBRE_REGIÃO (RETANDIR, RETANESQ, RETANBAIXO, RETANCIMA) onde os parâmetros dão o retângulo limite de um nó, retorna "true" se a região de busca se sobrepõe a esse retângulo e "false" caso contrário.

Além dessas funções,é definido o procedimento PROCESSAR (P) onde P é ponteiro para algum ponto. Este procedimento, executa o processamento necessário com P quando ele se encontra na região de busca.

O procedimento BUSCA_INTERVALO tem como parâmetros um ponteiro para um nó e os valores que determinam o retângulo limite desse nó, sendo chamado recursivamente para cada nó que é visitado. A mudança de parâmetros desse procedimento e os testes das funções são mais claramente vistos se forem consideradas as seguintes situações:

Supondo que todos os pontos contidos em uma árvore Quad estão distribuidos no espaço limitado por

RETANESQ $\leq K_1 \leq$ RETANDIR e RETANBAIXO $\leq K_2 \leq$ RETANCIMA

e que o ponto A, que é a raiz da árvore é dado pelas coordenadas $K_1 \triangleq$ XA e $K_2 \triangleq$ YA têm-se a situação mostrada na figura 4.13.

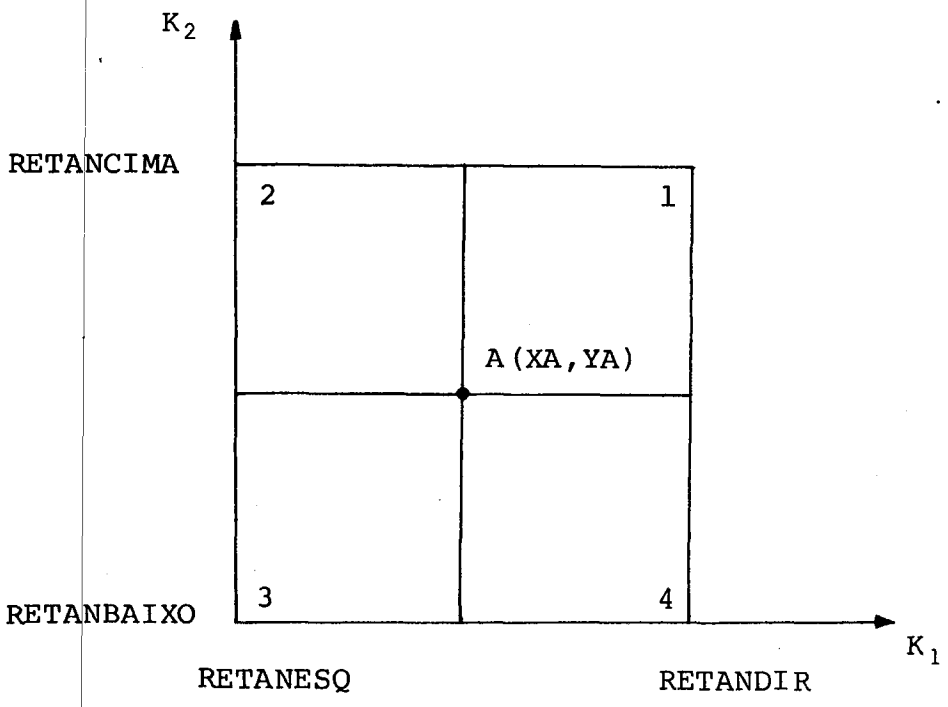


Fig. 4.13

Assim o retângulo limite da raiz A da árvore é dado pelos parâmetros (RETANESQ, RETANDIR, RETANBAIXO, RETANCIMA). O ponto A particiona esse espaço em quadrantes que são os retângulos limites dos seus filhos, dados pelos parâmetros:

- quadrante 1 - (XA, RETANDIR, YA, RETANCIMA)
- quadrante 2 - (RETANESQ, XA, YA, RETANCIMA)
- quadrante 3 - (RETANESQ, XA, RETANBAIXO, YA)
- quadrante 4 - (XA, RETANDIR, RETANBAIXO, YA)

Supondo agora que a região de busca é determinada pelos intervalos

$$REGIESQ \leq K_1 \leq REGIDIR \text{ e } REGIBAIXO \leq K_2 \leq REGICIMA$$

pode-se ter, por exemplo, a situação da figura 4.14.

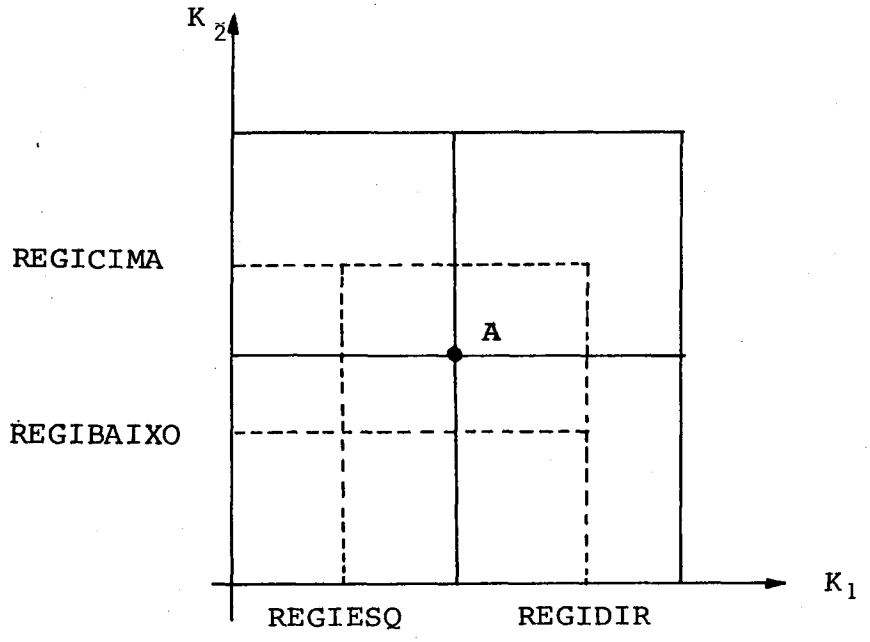


Fig. 4.14

Se a árvore tem a raiz A apontada por RAIZ e pontos com coordenadas nos intervalos

$$(MINK1 \leq K_1 \leq MAXK1) \quad e \quad (MINK2 \leq K_2 \leq MAXK2)$$

este procedimento será inicialmente chamado por

```
BUSCA_INTERVALO (RAIZ, MINK1, MAXK1, MINK2, MAXK2);
```

Iniciada a execução, para verificar se o nó A encontra-se na região de busca, compara-se suas coordenadas com os extremos da região. Em seguida deve-se comparar os quadrantes 1, 2, 3 e 4 gerados por A com a região para ver se eles se sobrepõem. Por exemplo, para a primeira subárvore de A compara-se a região de busca com o retângulo limite dado pelo quadrante 1, chamando a função RETÂNGULO SOBRE REGIÃO (XA, RETANDIR, YA, RETANCIMA). Se o valor retornado é TRUE, faz-se a chamada recursiva do procedimento com os parâmetros BUSCA_INTERVALO (A↑.SUB[1], XA, RETANDIR, YA, RETANCIMA). Assim por diante, o mesmo teste é feito com os quadrantes 2, 3 e 4, com os respectivos parâmetros de cada quadrante.

Seguem as funções e procedimentos detalhados, que pressupõem a existência dos limitantes globais declarados por:

```
var REGIESQ, REGDIR, REGIBAIXO, REGICIMA; real;
```

e baseiam-se na mesma estrutura de dados definida para inserção.

```
function ESTA NA REGIÃO (X:NO) : boolean;
(*retorna TRUE se X está na região de busca*)
var  X1, X2: real;
begin
     X1:= X↑,CHAVE[1];
     X2:= X↑,CHAVE[2];
     ESTA_NA_REGIÃO := (REGIESQ ≤ X1)and(X1 ≤ REGIDIR)and
                       (REGIBAIXO ≤ X2)and(X2 ≤ REGICIMA)
end ;
```

```
function RETANGULO_SOBRE_REGIÃO (RETANESQ, RETANDIR,RETANBAIXO,
         RETANCIMA: real) : boolean;
(*retorna TRUE se o retangulo dado pelos parâmetros se sobrepõe
 à região de busca*)
begin
  RETANGULO SOBRE REGIÃO := ((RETANESQ ≤ REGIDIR) and
     (RETANDIR ≤ REGIESQ) and (RETANBAIXO ≤ REGICIMA) and
     (RETANCIMA ≤ REGIBAIXO))
end ;
```

```
procedure BUSCA_INTERVALO (A:NO; RETANESQ, RETANDIR,RETANBAIXO,
                           RETANCIMA: real);
(*faz a busca na árvore de raiz A, com retangulo limite dado pe
los demais parâmetros*)
var XA, YA: real; (*coordenadas de A*)
begin
     XA:= A↑.CHAVE[1];
     YA:= A↑.CHAVE[2];
     if ESTA_NA_REGIÃO(A) then PROCESSAR (A);
        (A↑.SUB[1]<> nil) and
        RETANGULO_SOBRE_REGISTRO (XA, RETANDIR, YA, RETANCIMA)
     then BUSCA_INTERVALO (A↑.SUB[1], XA, RETANDIR, YA,
                           RETANCIMA);
     if (A↑.SUB[2]<> nil) and
        RETANGULO_SOBRE_REGIAO (RETANESQ, XA,YA, RETANCIMA)
     then BUSCA_INTERVALO (A↑.SUB[2], RETANESQ, XA,YA,
                           RETANCIMA);
     if (A↑.SUB[3]<> nil) and
         RETANGULO_SOBRE_REGIAO (RETANESQ, XA, RETANBAIXO, YA)
     then BUSCA_INTERVALO (A↑.SUB[3], RETANESQ, XA,RETANBAIXO,
                           YA);
     if (A↑.SUB[4]<> nil) and
        RETANGULO SOBRE REGIAO (XA, RETANDIR, RETANBAIXO, YA)
     then BUSCA_INTERVALO (A↑.SUB[4],XA,RETANDIR,RETANBAIXO,YA)
end;
```

Alguns testes empíricos são apresentados em [FINKEL-74] , para investigar a eficiência desse algoritmo. Os pontos considerados tem valores de coordenadas uniformemente distribuidos no intervalo $[0,1]^2$, e as regiões de busca são quadrados retilineamente orientados com um dado tamanho de lado localizadas randomicamente. Para cada tamanho de árvore considerado são geradas randomicamente quatro árvores. Para cada comprimento de lado considerado para o quadrado de busca, são feitas 25 buscas em cada uma das quatro árvores de um determinado tamanho, portanto, um total de 100 buscas por tamanho de quadrado. Na tabela 4b é mostrado, para cada combinação de tamanho de árvore e quadrado o número de nós visitados nas 100 buscas (coluna visitado), entre estes os que estavam na região (coluna encontrados) e as relações significativas entre estes dados. Os valores da coluna Visitados/Encontrados indica a quantidade total de trabalho feito pelo algoritmo para encontrar um nó particular na região que ele está pesquisando. A coluna Visitados /Busca mostra a quantidade de trabalho feita para encontrar todos os nós em uma certa região.

TABELA 4b

Resultados Obtidos nos Testes de Busca em Árvores Quad

Número de nós	Tamanho de lado	Visitados	Encontrados	Visitados/ Busca	Encontrado/ Busca	Visitados/ Encontrados
125	0.03175	598	13	5.98	0.13	46.00
	0.0625	789	41	7.89	0.41	19.24
	0.125	1218	206	12.18	2.06	5.91
	0.25	2195	799	21.95	7.99	2.75
	0.5	5188	3130	51.88	31.30	1.66
250	0.03125	777	22	7.77	0.22	35.32
	0.0625	1074	103	10.74	1.03	10.43
	0.125	1820	387	18.20	3.87	4.70
	0.25	3562	1536	35.62	15.36	2.32
	0.5	9550	6390	95.50	63.90	1.50
500	0.03125	975	55	9.75	0.55	17.73
	0.0625	1493	205	14.93	2.05	7.28
	0.125	2641	754	26.41	7.54	3.50
	0.25	6248	3163	62.48	31.63	1.97
	0.5	17453	12860	174.53	128.60	1.36
1000	0.03125	1316	99	13.16	0.99	13.29
	0.0625	2144	376	21.44	3.76	5.70
	0.125	4246	1611	42.46	16.11	2.64
	0.25	10100	6172	101.00	61.72	1.64
	0.5	31845	24880	318.45	248.80	1.28
2000	0.03125	1619	183	16.19	1.83	8.85
	0.0625	2906	786	29.06	7.86	3.70
	0.125	6803	3106	68.03	31.06	2.19
	0.25	18347	12627	183.47	126.27	1.45
	0.5	60581	49943	605.81	499.43	1.21
4000	0.03125	2407	397	24.07	3.97	6.06
	0.0625	4369	1504	43.69	15.04	2.90
	0.125	11096	6218	110.96	62.18	1.78
	0.25	33133	24610	331.33	246.10	1.35
	0.5	114767	99875	1147.67	998.75	1.15

4.2.2. Árvore Quad Ótima

Para alguns casos pode ser conveniente pensar em gastar um tempo maior de pré-processamento para construir uma árvore otimizada que garanta um melhor tempo médio de busca. Esta árvore Quad otimizada a ser descrita para k = 2 tem a seguinte propriedade: para todo nó X, nenhuma subárvore de X tem mais da metade dos nós da árvore cuja raiz é X. Para que ela seja construida, inicialmente deve-se ordenar léxico-graficamente os nós pela primeira chave e depois pela segunda. A raiz da árvore é escolhida como o elemento médio dessa sequência ordenada, que depois é subdividida em 4 subcoleções para formar os descendentes da raiz. Para cada subcoleção aplica-se o processo recursivamente. Com este método, os nós que antecedem o elemento médio da sequência ficarão nos quadrantes 2 e 3 e os nós que o sucedem ficarão nos quadrantes 1 e 4. Logo, nenhuma subárvore do nó terá mais da metade dos nós.

O tempo para otimização será também $O(n \log_2 n)$ pois:

- a ordenação de n elementos exige $n\log_2 n$ operações;
- em cada nível de recursão, a seleção da mediana requer n operações, o reagrupamento de registros requer n operações e pode ser feito de tal forma que cada grupo permaneça ordenado.
- o nível máximo de recursão será o comprimento de caminho máximo, que na árvore otimizada é dado por $\lfloor \log_2 n \rfloor$, portanto limitado por $\log_2 n$.

Logo, o tempo total para otimização será também da ordem de $n\log_2 n$.

Estudos empíricos de [FINKEL-74] mostram que o comprimento de caminho interno de uma árvore otimizada é cerca de 15% menor que a árvore construida por inserção simples e as medidas de eficiência como Nós Visitados/Nós Encontrados e Nós Visitados/Busca permanecem aproximadamente as mesmas.

4.3. *Análise de Busca por Intervalo em uma Árvore Quad Perfeita*

Devido à sua versatilidade, a estrutura de árvore Quad é muito difícil de ser analisada matematicamente, mas é possível apresentar algum resultado sobre seu comportamento, analisando o algoritmo de busca aplicado a uma forma particular dessa estrutura, a *árvore perfeita* [BENTLEY-75b]. Serão mantidas ainda nesta análise, as mesmas condições assumidas até agora, ou seja, registros com duas chaves que tem valores reais uniformemente distribuidos no intervalo $[0,1]^2$.

Uma árvore Quad de h níveis é "perfeita", se e somente se:

a) ela tem 4^i nós em todo nível i, $0 \leq i \leq h-1$, onde define-se que a raiz está no nível 0, e
b) todo nó está no centro do seu retângulo limite.

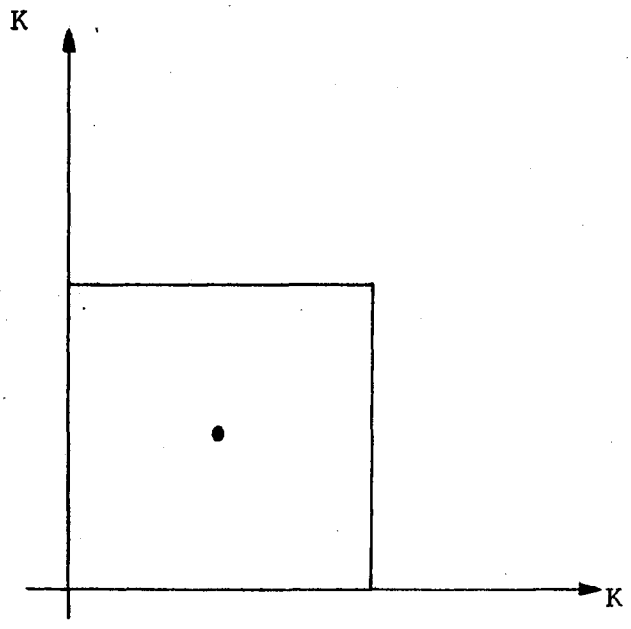
Nestas condições, cada nó do nível i tem um retângulo limite que é um quadrado com lado $(\frac{1}{2})i$. Logo, os retângulos limites de um nível i de uma árvores perfeita particionam o espaço $[0,1]^2$ em um "tabuleiro" de 4^i quadrados e os limites entre retângulos de nós no mesmo nível são as linhas

$K_1 = K_2 = t/2^i$ para $0 \leq t \leq 2^i$, t inteiro.

Por exemplo, considerando que os registros

A: (1/2, 1/2)	H: (5/8, 5/8)	O: (1/8, 3/8)
B: (3/4, 3/4)	I: (7/8, 5/8)	P: (1/8, 1/8)
C: (1/4, 3/4)	J: (3/8, 7/8)	Q: (3/8, 1/8)
D: (1/4, 1/4)	K: (1/8, 7/8)	R: (7/8, 3/8)
E: (3/4, 1/4)	L: (1/8, 5/8)	S: (5/8, 3/8)
F: (7/8, 7/8)	M: (3/8, 5/8)	T: (5/8, 1/8)
G: (5/8, 7/8)	N: (3/8, 3/8)	U: (7/8, 1/8)

São inseridos nesta ordem numa árvore Quad inicialmente vazia, teremos as situações mostradas na figura 4.15 (a), (b) e (c) para os níveis 0,1 e 2 respectivamente.



nível 0 (i = 0)

nº de nós = 1

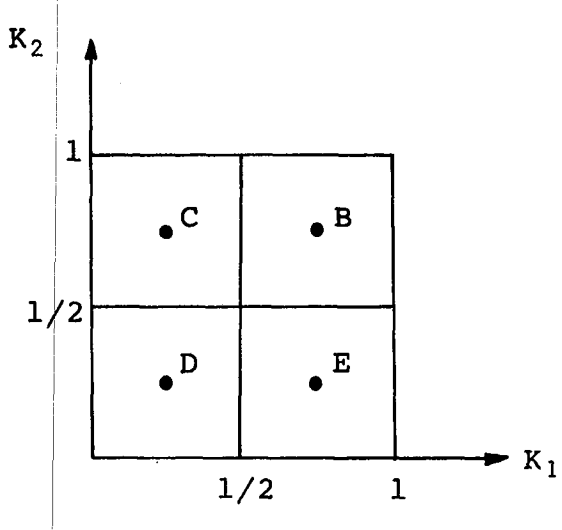
lado do retângulo = 1

partição = 1 quadrados

linhas limites =

t	0	1
$k_1=k_2$	0	1

(a)



nível 1 (i=1)

nº de nós = 4

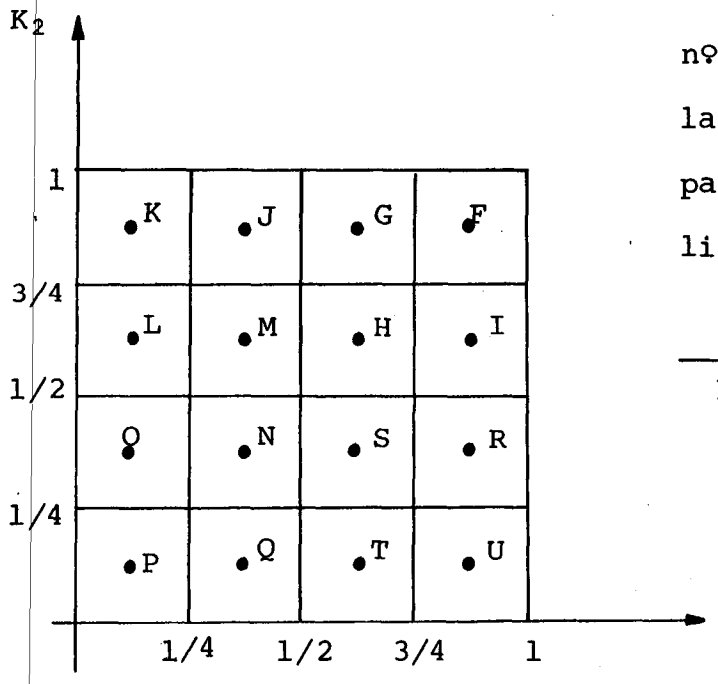
lado do retângulo = 1/2

partição = 4 quadrados

linhas limites =

t	0	1	2
$k_1 = k_2$	0	1/2	1

(b)



nível 2 (i = 2)

nº de nós = 16

lado do retângulo = 1/4

partição = 16 quadrados

linhas limites =

t	0	1	2	3	4
$k_1 = k_2$	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1

(c)

Fig. 4.15.

E para o conjunto dos nós da figura 4.16,

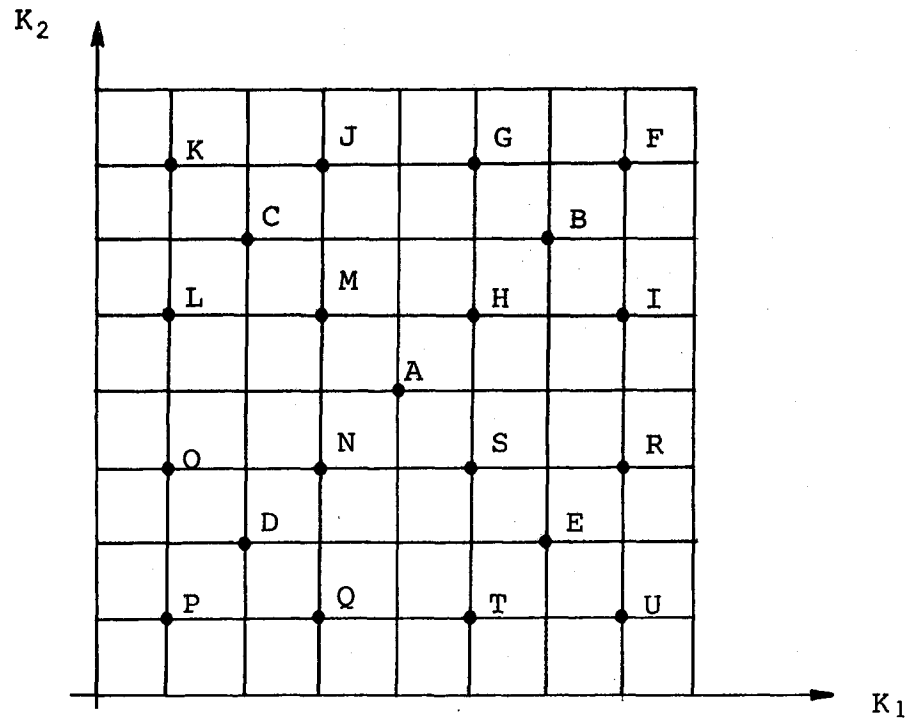


Fig. 4.16

a árvore equivalente tem a forma mostrada na figura 4.17.

No algoritmo BUSCA_INTERVALO um nó é visitado se e somente se, seu retângulo limite se sobrepõe à região de busca. Logo, para determinar quantos nós são visitados durante uma busca, é suficiente determinar quantos nós em cada nível i tem um retângulo limite que se sobrepõe à região e somar esses valores para todos os níveis da árvore.

As regiões de busca são retângulos em $[0,1]^2$ com lados x_1 e x_2, o que corresponde a uma consulta de busca por intervalo que pede registros com valores de chaves tal que

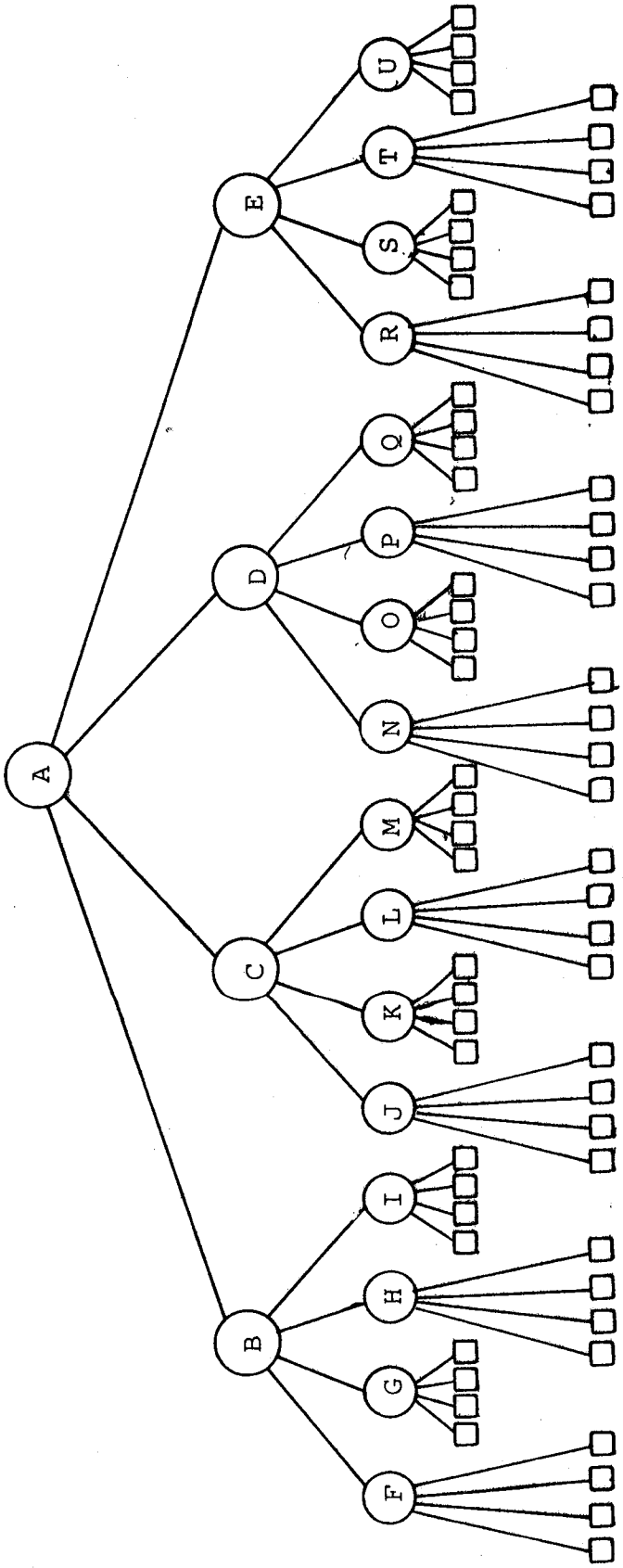


Fig. 4.17

$$b \leq K_1 \leq b + x_1 \qquad e \qquad c \leq K_2 \leq c + x_2$$

Vamos definir $V(i, x_1, x_2)$ como o número de retângulos limite no nível i que se sobrepõem à região de busca, que é exatamente o número de nós visitados no nível i pelo algoritmo BUSCA_INTERVALO.

Como já foi visto, na árvore perfeita os retângulos do nível i particionam o espaço em 4^i quadrados regulares como um tabuleiro. Além disso, 2^i quadrados estão alinhados ao longo dos eixos K_1 e K_2. Por exemplo, no nível 2, tem-se 16 quadrados no total e 4 quadrados ao longo de cada eixo, isto é, ao se traçar retas paralelas aos eixos intercepta-se sempre quatro quadrados como vemos na figura 4.18.

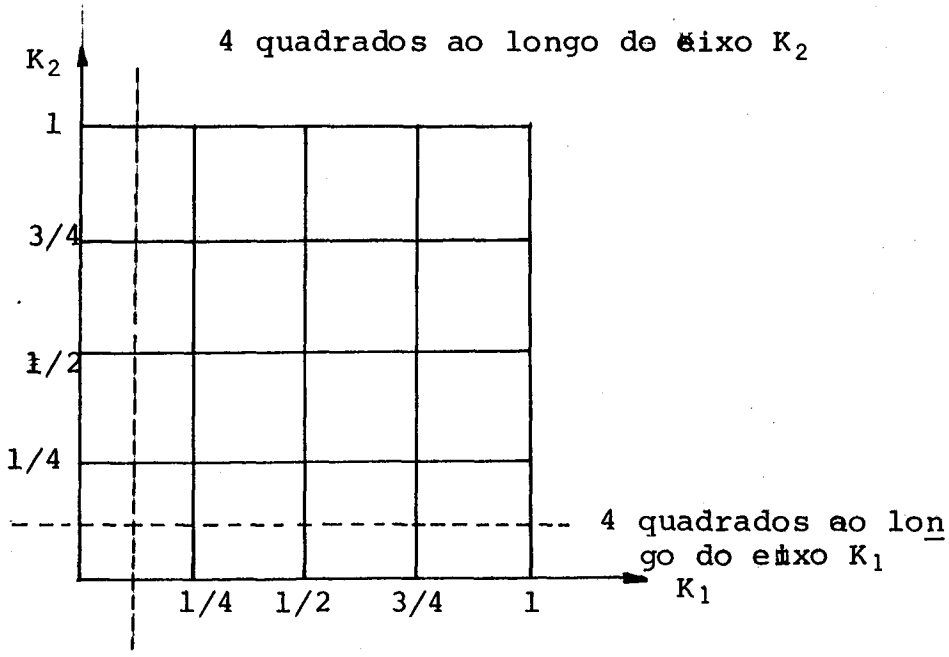


Fig. 4.18

Como a região de busca é um retângulo de lados x_1 e x_2, o número esperado de quadrados ao longo da direção K_1 que interseccionam a região, está entre $x_1.2^i$ e $x_1 2^i + 1$ e ao longo da direção K_2 esse valor está entre $x_2.2^i$ e $x_2.2^i + 1$. Esses números médios têm como limites superiores os valores

$$\lceil x_1 2^i + 1 \rceil \quad \text{e} \quad \lceil x_2 2^i + 1 \rceil .$$

Por exemplo, considerando uma região de busca com lados $x_1 = 0,4$ e $x_2 = 0,3$, com a localização mostrada na figura 4.19 para o nível 2,

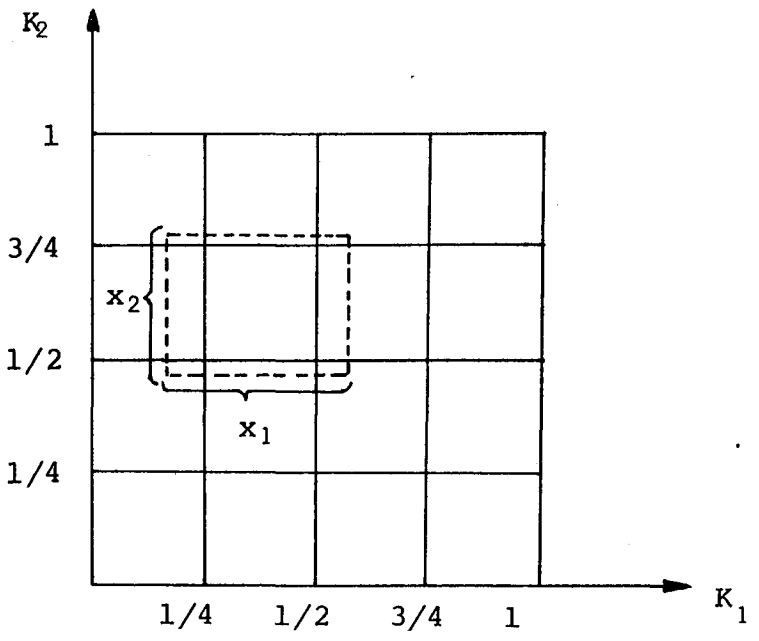


Fig. 4.19

vemos o pior caso possível para esse tamanho de região de busca onde ele se sobrepõe à 3 quadrados na direção K1 e 3 na direção K_2, o que é dado por:

$$\lceil x_1.2^i + 1 \rceil = \lceil (0.4) 2^2 + 1 \rceil = 3$$

$$\lceil x .2^i + 1 \rceil = \lceil (0.3).2^2 + 1 \rceil = 3$$

Para esses valores de x_1 e x_2, o número esperado de quadrados que interseccionam a região está entre

$$x_1 2^i \quad \text{e} \quad x_1 2^i + 1$$

na direção do eixo K_1 e entre

$$x_2 2^i \quad \text{e} \quad x_2 2^i + 1$$

na direção do eixo K_2. Assim os valores

$$x_1 2^i + 1 \quad \text{e} \quad x_2 2^i + 1$$

são boas estimativas do número de intersecções ao longo dos eixos, a menos que i esteja próximo de 0 e x_j próximo de 1. Portanto, o número total de quadrados que interseccionam a região no nível i, é dado aproximadamente pelo produto.

$$V(i,x_1,x_2) = (x_1 2^i + 1)\ (x_2 2^i + 1)$$

Este valor dá uma boa estimativa do número médio de nós visitados no nível i pelo algoritmo BUSCA-INTERVALO aplicado à uma árvore Quad perfeita.

Para avaliar o número total de nós visitados durante uma busca é necessário somar os valores dados por $V(i,x_1,x_2)$ para todos os níveis i. Denotando por $E(h,x_1,x_2)$ o número de nós visitados em uma busca numa árvore de altura h, tem-se:

$$E(h,x_1,x_2) = \sum_{0 \le i \le h-1} V(i,x_1,x_2)$$

$$= \sum_{0 \le i \le h-1} (x_1 2^i+1)(x_2 2^i+1)$$

$$= \sum_{0 \le i \le h-1} [x_1 x_2 4^i + (x_1 + x_2) 2^i + 1]$$

$$= x_1 x_2 \sum_{0 \le i \le h-1} 4^i + (x_1 + x_2) \sum_{0 \le i \le h-1} 2^i + h \qquad [4.1]$$

Para expressar esta soma como função de n, número de nós da árvore, usa-se o fato de que cada nível i de uma árvore perfeita tem 4^i nós para estabelecer que:

$$n = \sum_{0 \le i \le h-1} 4^i = \frac{4^h - 1}{3}, \qquad [4.2]$$

$$h = \log_4 (3n + 1) \qquad [4.3]$$

Além disso,

$$\sum_{0 \le i \le h-1} 2^i = 2^h - 1$$

$$\simeq 2^h$$

$$= 2^{\log_4 (3n+1)}$$

$$\simeq 2^{\log_4 3n}$$

$$= (3n)^{1/2} \qquad [4.4]$$

Substituindo-se [4.2], [4.3] e [4.4] em [4.1], vem:

$$E(n, x_1, x_2) \simeq x_1 x_2 n + (x_1 + x_2)(3n)^{1/2} + \log_4 (3n)$$

que é aproximadamente o número de nós visitados por BUSCA_INTERVALO, para fazer uma busca dos nós que se localizam em uma re

gião retangular de lados x_1 e x_2 em uma árvore Quad perfeita de n nós (onde $n = (4^h-1)/3$).

Em geral, para uma árvore Quad perfeita de ordem k teremos 2^{ki} hipercubos com 2^i deles na direção de $K_1, K_2, \ldots, K_k$. Então para uma consulta com intervalos x_j tal que:

$$a_j \leq R_j \leq a_j + x_j \ , \qquad j = 1,k$$

as equações V e E são:

$$V(i,x_1,x_2,\ldots,x_k) = \prod_{i\leq j\leq k} (x_j.2^i+1)$$

e,

$$E(h,x_1,x_2,\ldots,x_k) = \sum_{0\leq i\leq h-1} V(i,x_1,x_2,\ldots,x_k)$$

$$= \sum_{0\leq i\leq h-1} \prod_{1\leq j\leq k} (x_j.2^i+1) \qquad [4.5]$$

Podemos expressar a equação [4.5] por:

$$= P_k \sum_{0\leq i\leq h-1} 2^{ki} + P_{k-1} \sum_{0\leq i\leq h-1} 2^{(k-1)i} + \ldots + P_1 \sum_{0\leq i\leq h-1} 2^i + \sum_{0\leq i\leq h-1} 1$$

onde,

$$P_k = \prod_{1\leq j\leq k} x_j$$

$$\vdots$$

$$P_3 = \sum_{\substack{1\leq j,\ell,p\leq k \\ j\neq\ell\neq p}} x_j\, x_\ell\, x_p$$

$$P_2 = \sum_{\substack{1 \le j,\ell \le k \\ j \neq \ell}} x_j x_\ell$$

$$P_1 = \sum_{1 \le j \le k} x_j$$

O número total de nós em uma árvore Quad perfeita de ordem k é:

$$n = \sum_{0 \le i \le h-1} 2^{ki}$$

$$= \frac{2^{kh} - 1}{2^k - 1}$$

ou seja,

$$h = \log_{2k} \left[(2^k-1)\ n+1\right] \qquad [4.6]$$

Temos também que:

$$\sum_{0 \le i \le h-1} 2^{(k-i)i} = \frac{2^{(k-1)} - 1}{2^{(k-1)} - 1}$$

$$\le 2^{(k-1)h}$$

$$= 2^{(k-1)\log_{2^k}\left[(2^k-1)n+1\right]}$$

$$\cong 2^{(k-1)\log_{2^k}\left[(2^k-1).n\right]}$$

$$= \left[(2^k-1)n\right]^{(k-1)/k}$$

$$\simeq O(n^{1-(1/k)}) \qquad [4.7]$$

Substituindo [4.6] e [4.7] em [4.5], obtemos:

$$E(n,x_1,\ldots,x_k) = n \cdot \prod_{1 \le j \le k} x_j + O(n.^{1-(1/k)})$$

Essa expressão, que chamamos de avaliadora, apesar de ter sido construida para árvores Quad perfeitas, também pode representar uma boa estimativa do número de nós visitados em uma árvore randômica. Isto pode ser observado na tabela 4c, que permite comparar os resultados empíricos mostrados anteriormente para k=2, com os valores esperados segundo a expressão $E(n, x_1, x_2)$.

Considerando regiões de busca como quadrados de lado x, a coluna "Nós Visitados" apresenta o número de nós visitados nos testes realizados e o número esperado de nós visitados dado pela expressão avaliadora usada para o caso particular de quadrados de lado x:

$$E(n,x) = x^2n + 2x(3n)^{1/2} + \log_4 3n$$

Observa-se apenas que o número esperado de nós visitados tende a ser levemente menor que o número de nós realmente visitados nos testes. Isso acontece exatamente porque a expressão avaliadora foi construida com base em árvores perfeitas, e os resultados empíricos foram observados em árvores randômicas. Nas árvores construidas randomicamente o comprimento de caminho interno é maior ou igual que na árvore perfeita, logo o número de nós visitados deve ser maior.

A coluna "overwork" da tabela 4c nos dá a diferença entre o número de nós visitados e o número de nós na região de busca, tanto para os resultados encontrados nos testes empíricos como para os resultados encontrados pela equação $E(n,x)$. O número esperado de nós em uma região de busca quadrada de lado x, com chaves no espaço $[0,1]^2$ é nx^2. Então, para o caso anali

dado, o "overwork" esperado do algoritmo BUSCA_INTERVALO é aproximadamente $2\sqrt{3x}\, n^{1/2} + \log_4 3n$.

TABELA 4c

Comparação dos Valores Empíricos e Avaliados por E(n,x) para o número de nós Visitados para Intervalos de Busca em $[0,1]^2$. Cada árvore tem n nós e a Região de Busca é um Quadrado de Lado x.

Número de nós (n)	Tamanho de lado x	Nós visitados		Overwork	
		Empíricos	Avaliados por E(n,x)	Empíricos	Avaliados por $E(n,x)-nx^2$
125	0.03125	5.98	5.61	5.85	5.49
	0.0625	7.89	7.18	7.48	6.70
	0.125	12.18	11.07	10.12	9.17
	0.25	21.95	21.77	13.96	13.96
	0.5	51.88	54.89	20.58	23.64
250	0.03125	7.77	6.73	7.55	6.49
	0.0625	10.79	9.18	9.71	8.20
	0.125	18.20	15.53	14.33	11.62
	0.25	35.62	34.09	20.26	18.47
	0.5	95.50	94.66	31.60	32.16
500	0.03125	9.75	8.18	9.20	7.70
	0.0625	14.93	12.07	12.38	10.17
	0.125	26.41	22.77	18.87	14.96
	0.25	62.48	55.90	30.85	24.64
	0.5	174.53	169.01	54.93	44.01
1000	0,03125	13.16	10.18	12.17	9.20
	0.0625	21.44	16.53	17.68	12.62
	0.125	42.46	35.09	26.35	19.47
	0.25	101.00	95.66	39.28	33.16
	0.5	318.45	310.55	69.65	60.55
2000	0.03125	16.19	13.07	14.36	11.20
	0.0625	29.06	23.77	21.20	15.96
	0.125	68.03	56.89	36.97	25.64
	0.25	183.47	170.00	57.20	45.01
	0.5	605.81	583.73	106.38	83.74
4000	0.03125	27.07	17.53	20.10	13.62
	0.0625	43.69	36.09	28.65	20.46
	0.125	110.96	96.66	48.78	34.16
	0.25	331.33	311.55	85.23	61.55
	0.5	1147.67	1116.32	148.92	116.32

4.4. *Análise do Pior Caso de Busca por Intervalo para Árvores Quad Balanceadas*

Para analisar o pior caso de tempo de processamento do algoritmo BUSCA_INTERVALO para árvores Quad, vamos considerar somente árvores balanceadas [LEE-78]. Sem perda de generalidade podemos assumir que todas as chaves são positivas e o número total de registros é $n = \frac{2^{hk} - 1}{2^k - 1}$. Assim, essa árvore será uma árvore balanceada com todas as folhas aparecendo no nível h-1. Para ilustração será discutido o que acontece com a árvore bi-dimensional, ou seja, quando k = 2. Nesse caso, o número total de registros é $n = \frac{4^h - 1}{3}$ e os nós estão no primeiro quadrante. A região de busca tem a forma de um retângulo e podemos assumir, sem afetar a complexidade, que ela tem os dois eixos de coordenadas como os limites inferiores, isto é, ela sempre se localiza de acordo com a figura 4.20, onde L(i) e U(i) são respectivamente os limites inferior e superior do intervalo para a chave K_i, i = 1,2.

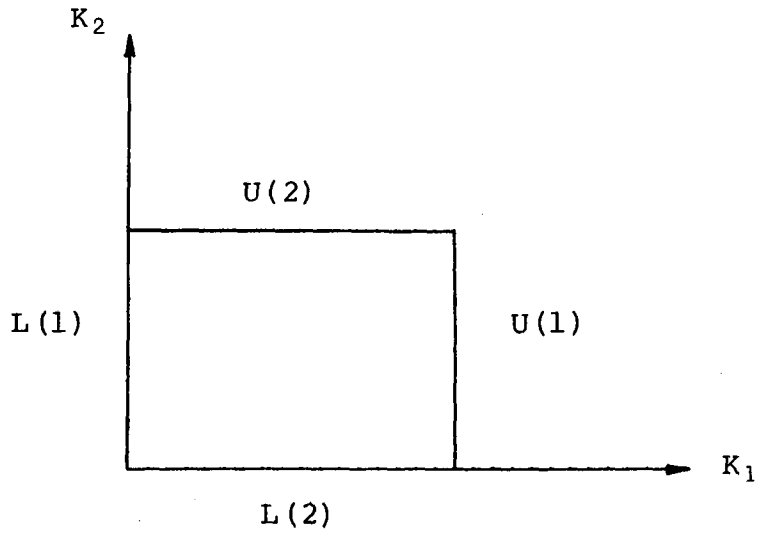


Fig. 4.20

A análise é feita contando-se o número de visitas feitas a nós durante a busca, com uma restrição importante: no algoritmo BUSCA_INTERVALO fazemos uma visita a um nó se e somente se seu retângulo limite se sobrepõe à região de busca; nesta análise contamos um número de visitas que não é o número total de visitas feitas em BUSCA_INTERVALO, pois consideramos que, se o retângulo limite de um nó está totalmente dentro da região de busca, as visitas a ele e aos seus descendentes não serão mais consideradas na contagem. O sub-arquivo que esse nó representa será recuperado como um todo. Logo, um nó será eliminado da busca em dois casos: se ele representar uma subárvore que não pode conter nenhum nó que esteja na região de busca, ou se todos os nós da sua subárvore estão nessa região.

Como estamos interessados no pior caso, podemos arranjar os nós de maneira a conseguir, para uma dada região de busca, o maior número possível de visitas. Assim, sendo A a raiz de uma árvore e B, C, D e E seus filhos, no pior caso a região de busca se sobrepõe às 4 regiões representadas pelos filhos de A, o que será obtido, se eles tiverem por exemplo, as localizações mostradas na fig. 4.21:

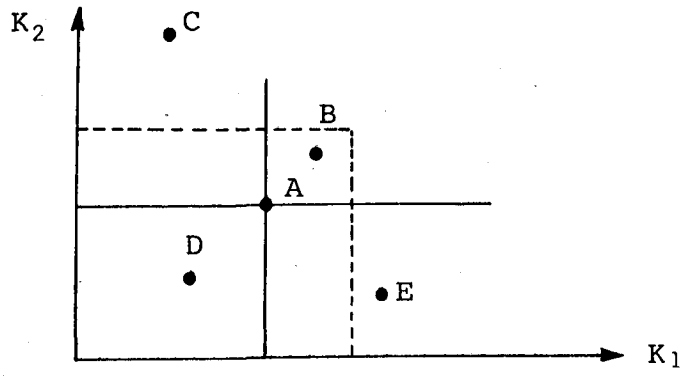


Fig. 4.21

Nesta situação, apesar de D ser eliminado nesta análise, porque seu retângulo limite está totalmente dentro da região de busca, ainda serão feitas mais 3 visitas. Em qualquer outra situação seriam feitas apenas duas visitas, pois se a região de busca não intersecciona os quatro quadrantes, só pode interseccionar dois deles, como pode ser verificado na figura 4.22.

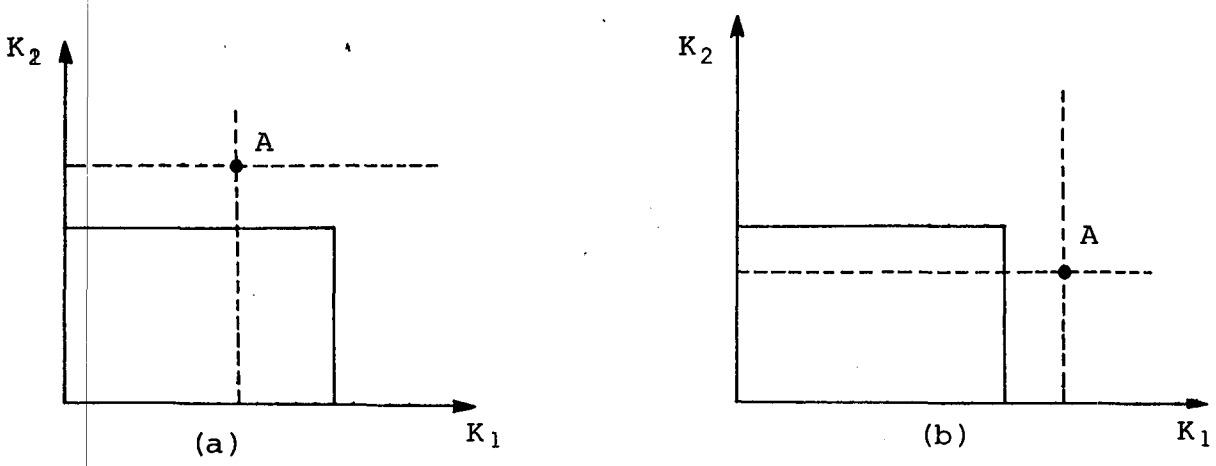


Fig. 4.22

Neste ponto, já foram feitas visitas a A, B, C e E, e dos 4 filhos de A, um foi eliminado da contagem (D).

Vejamos o que acontece com a segunda subárvore de A representada por C. Depois de visitar C, duas das suas quatro subárvores, J,K,L e M serão eliminadas, ou por estarem completamente fora da região de busca, que é o que acontece com J e K, conforme mostra a figura 4.23,

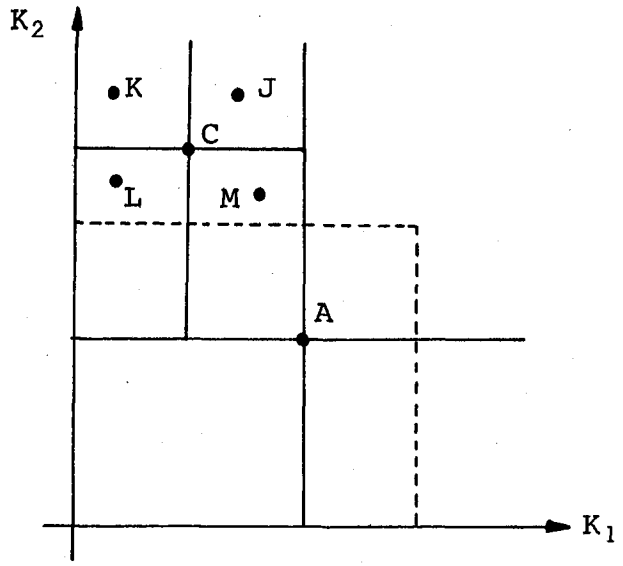


Fig. 4.23

ou por estarem completamente dentro da região, que é o que acontece com L e M conforme mostra a figura 4,24.

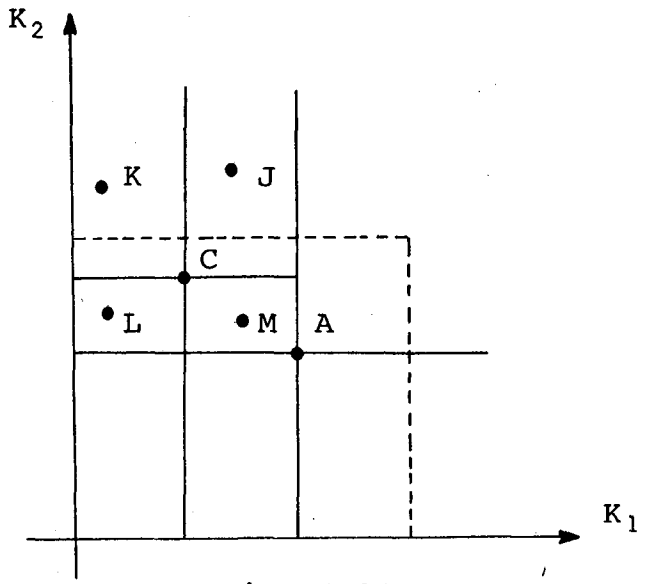


Fig. 4.24

Na subárvore E da figura 4.21 a situação é semelhante à de C, exceto no que diz respeito ao sentido em que o limite da região de busca corta os quadrantes de E e C.

Para cada uma das duas subárvores de C e E visitadas dois de seus filhos são também eliminados, independentemente da posição dos nós. No exemplo mostrado na figura 4.25, os nós L e M tem duas subárvores cujos retângulos limites não se sobrepõem à região de busca.

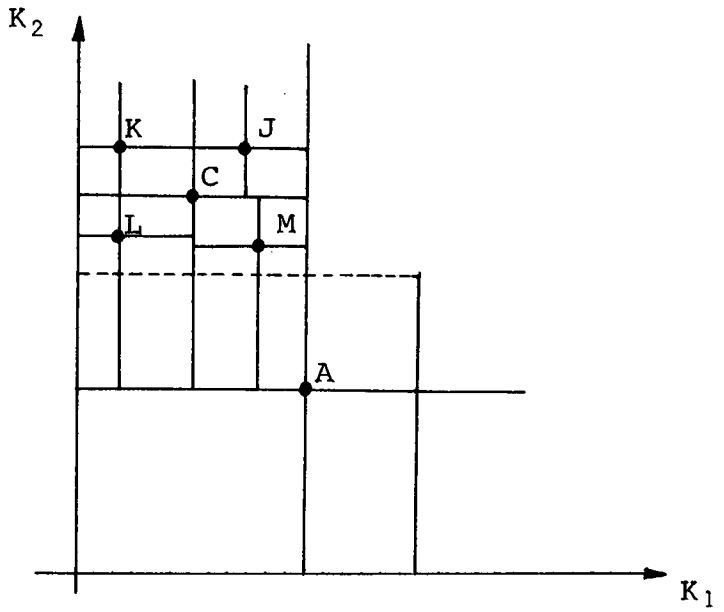
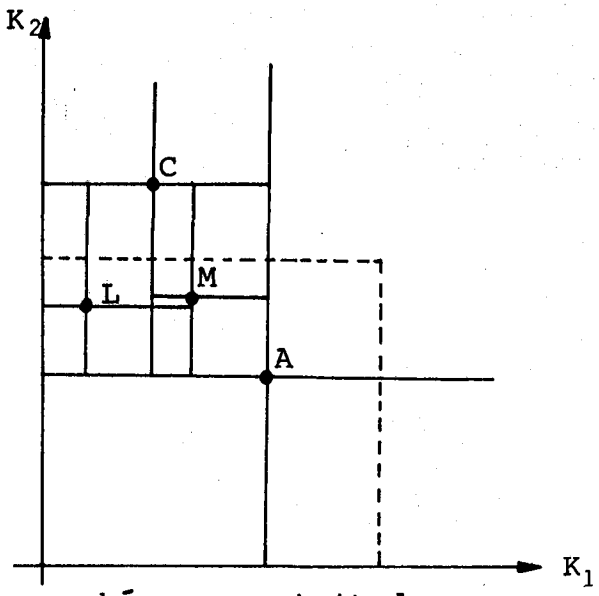


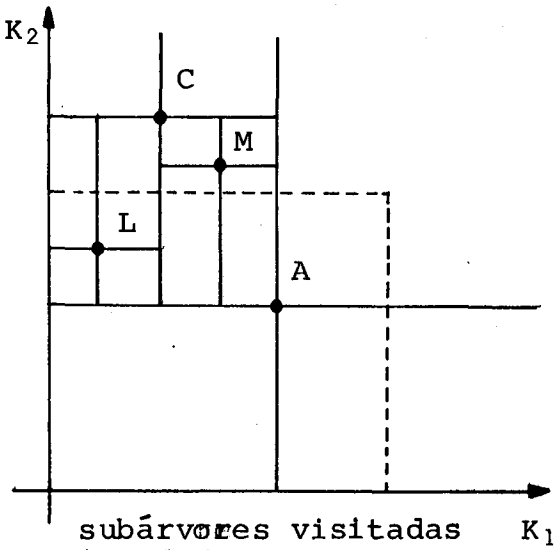
Fig. 4.25

Se a localização desses nós fosse uma das mostradas na figura 4.26, ainda assim seriam eliminados dois filhos de L e dois de M.



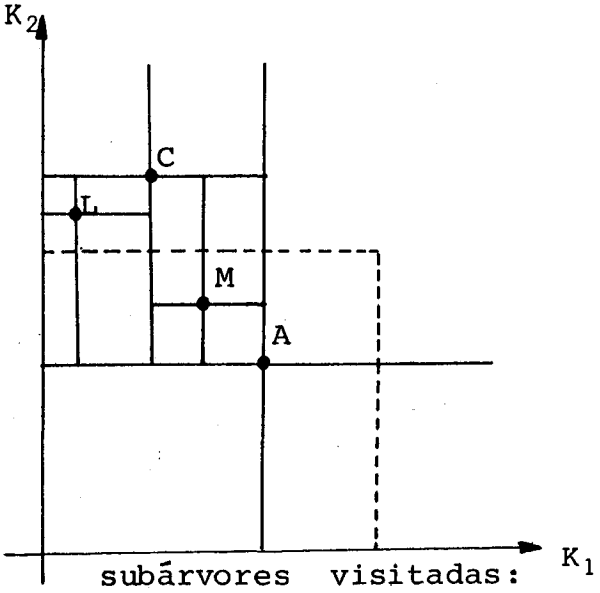
subárvores visitadas:
1 e 2 de L
1 e 2 de M

(a)



subárvores visitadas
1 e 2 de L
3 e 4 de M

(b)



subárvores visitadas:
3 e 4 de L
1 e 2 de M

(c)

Fig. 4.26

Portanto, conclui-se que, acontece exatamente o mesmo com as subárvores C e D no pior caso, ou seja, dois de seus quatro filhos são visitados e dois eliminados. Dos filhos de seus dois filhos visitados apenas dois são visitados e assim por diante, até o final da árvore, independentemente da localização dos nós.

Com relação à primeira subárvore de A, representada por B pode-se ver que, movendo as coordenadas K_1 e K_2 para K_1' e K_2' como mostra a figura 4.27,

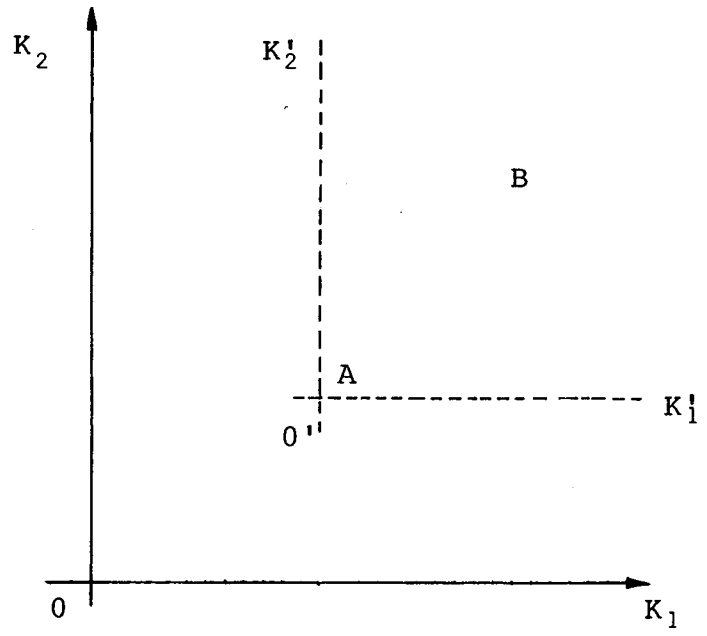


Fig. 4.27

tal que K'_1 e K'_2 tem origem em A, tem-se a mesma configuração do início da busca, e B é analisado da mesma forma que A. Assim, o que acontece com B é o mesmo que acontece com A, a menos do fato de B estar um nível abaixo.

Considere-se o exemplo da figura 4.28, que mostra

parte de um conjunto de nós de uma árvore Quad, e uma região de busca por intervalo, aplicada neste conjunto.

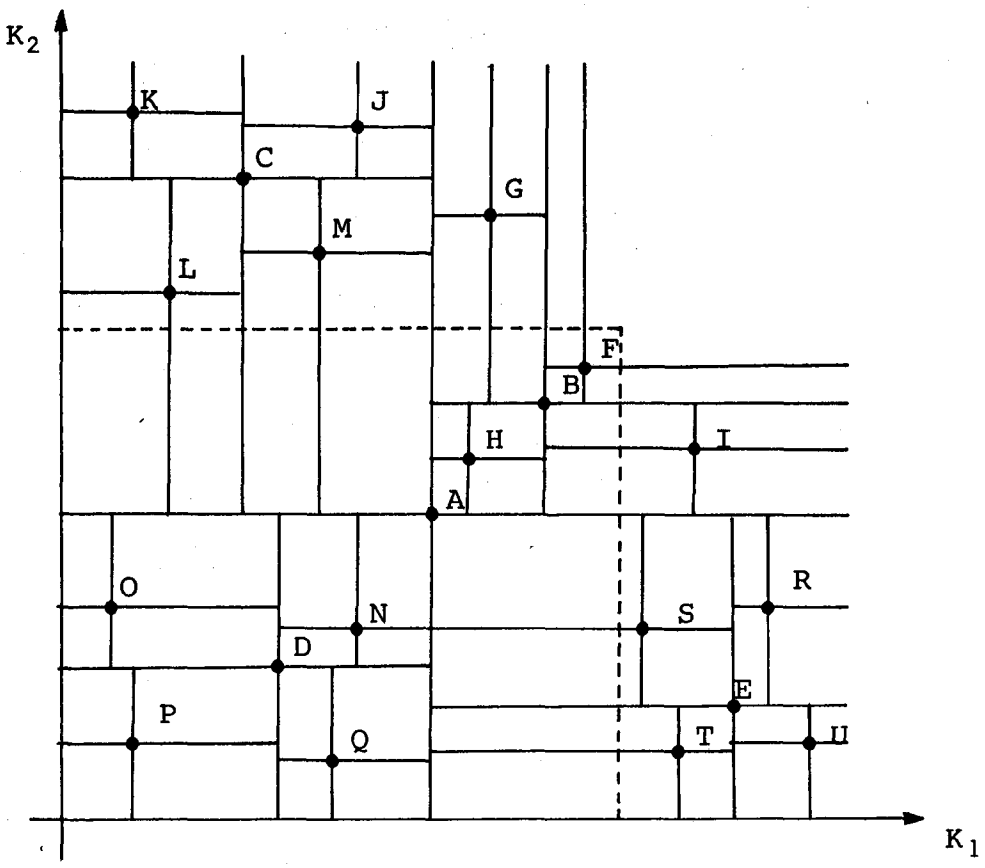


Fig. 4.28

De acordo com a análise desenvolvida vê-se, na figura 4.29 o resultado da busca aplicada à árvore equivalente ao exemplo da figura 4,28, onde os nós hachurados representam nós que não precisam ser visitados.

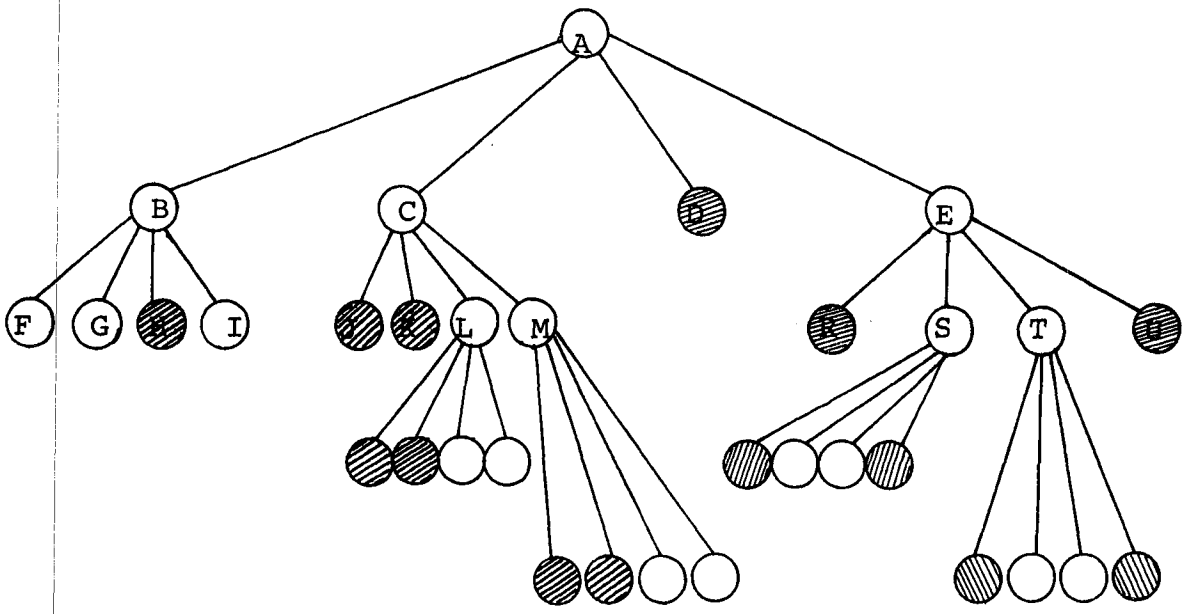


Fig. 4.29

Com os resultados obtidos analisando-se os primeiros níveis da árvore, é possível construir a equação de recorrência do número de nós visitados, pois sabe-se o que acontece com todos os nós.

Assim, sendo h a altura da árvore e T(h) o número total de nós visitados; sendo h_1 a altura da subárvore com raiz C ou E e T'(h_1) o equivalente a T(h) para estas subárvores, tem se que as equações de recorrência são:

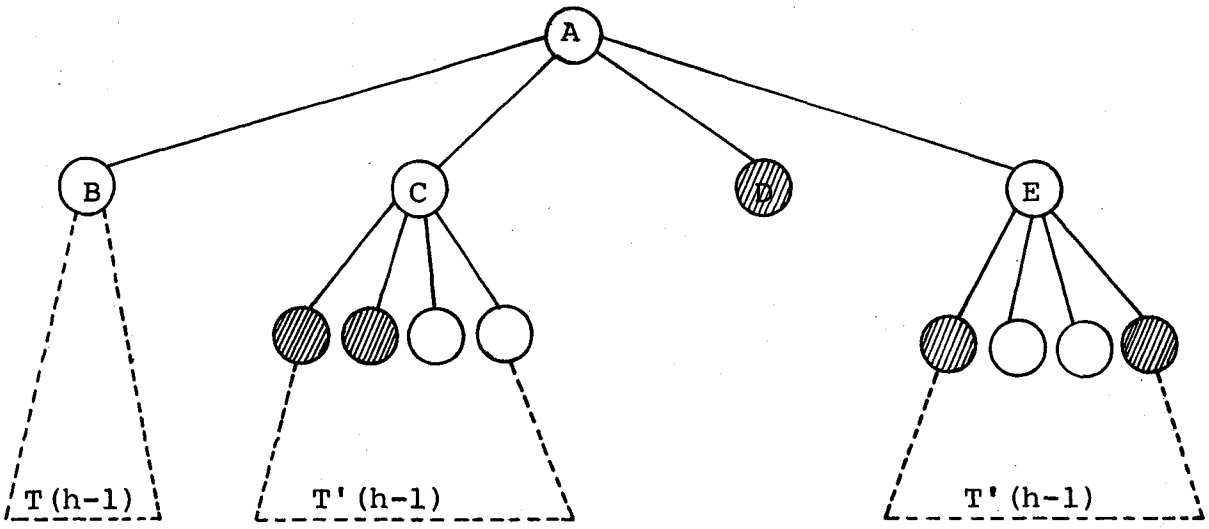
$$T(h) = 1 + T(h-1) + 2T'(h-1)$$

$$T'(h_1) = 1 + 2T'(h_1 - 1)$$

Pode-se visualizar estas equações para o exemplo da figura 4.28 na figura 4.30 que mostra a relação entre o número de nós visitados em uma árvore e suas subárvores.Os nós hachura

dos representam nós que foram eliminados da contagem.

T(h) - número de nós visitados em toda a árvore.



$T'(h_1)$ - número de nós visitados na subárvore com raiz C

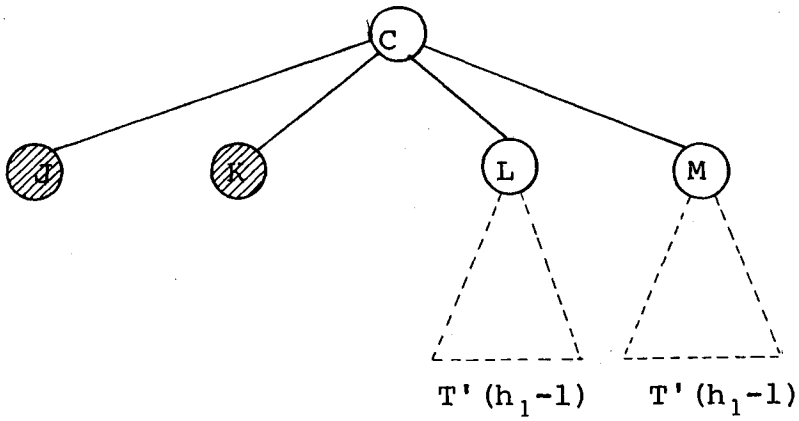


Fig. 4.30

As equações de recorrência:

$$T(h) = 1 + T(h-1) + 2\,T'(h-1)$$

$$T'(h_1) = 1 + 2\,T'(h-1)$$

serão resolvidas com as condições iniciais:

$$T(0) = T'(0) = 0$$

$$T(1) = T'(1) = 1$$

Resolvendo $T'(h_1)$, obtem-se:

$$T'(h_1) = \sum_{0 \leq i \leq h_1} 2^i$$

$$= 2^{h_1} - 1 \qquad [4.8]$$

Substituindo [4.8] em T (h) vem:

$$T(h) = 2^h - 1 + T(h-1)$$

e resolvendo T (h), tem-se

$$T(h) = \sum_{1 \leq i \leq h} 2^i - h$$

$$= 2\,(2^h - 1) - h$$

$$= 2\,(2^{2h(1/2)} - 1) - h$$

Como

$$n = \frac{2^{2h} - 1}{3} \qquad [4.9]$$

substituindo [4.9] em T (h) resulta:

$$T(h) \simeq O(2.n^{1/2})$$

ou seja,

$$O(k.n^{(1-(1/k))}) \quad \text{para } k = 2.$$

A análise do pior caso de árvores Quad balanceadas k-dimensionais, baseia-se na idéia de contagem das regiões "limitadas", pois estas são as regiões que não serão visitadas.Uma região é considerada limitada se ela tem como limite os próprios eixos ou, os hiperplanos paralelos aos eixos que passam pelos nós e dividem uma região em outras 2^k regiões.

Por exemplo, para k=2, depois de visitar a raiz tem-se a primeira partição que gera regiões associadas com cada um dos nós B, C, D e E sendo que a região associada a D é limitada, como pode ser visto na figura 4.31.

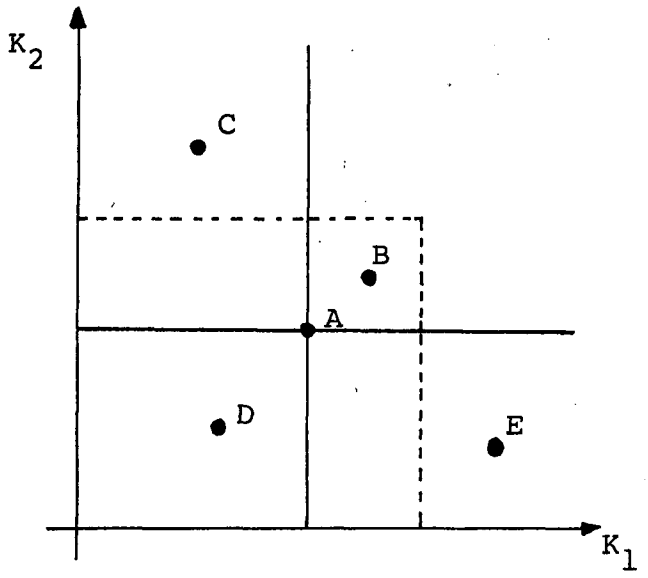


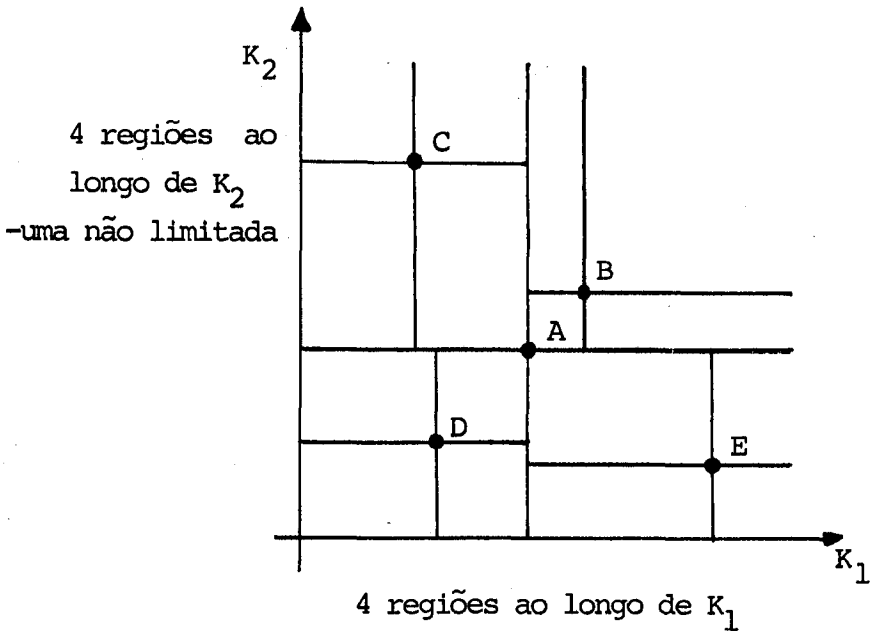
Fig. 4.31

Essa região, considerando-se que ocorre o pior caso, será eliminada da busca por estar totalmente dentro da região de busca. E assim em todos os níveis, sempre que uma região limitada é gerada por uma partição do espaço, uma subárvore será eliminada da contagem. Ao contar-se o número de regiões limi

tadas em cada nível, pode-se saber o número de regiões não limitadas, que é o número de nós visitados nesse nível. Somando-se esse número para todos os níveis tem-se o número total de nós visitados em toda a árvore.

Ainda para ilustração, vamos continuar considerando uma árvore Quad de 2 dimensões, na qual linhas perpendiculares aos eixos particionam o plano onde se localizam os nós. Estas linhas também servem como limites de novas regiões. No nível 1, como já foi visto na figura 4.31, traça-se as primeiras linhas perpendiculares a K_1 e K_2, o que é chamado de primeira "visita" aos eixos K_1 e K_2 com o que obtém-se quatro regiões, sendo uma limitada.

No nível 2 traça-se duas linhas perpendiculares a cada eixo, uma em cada região das geradas na primeira visita. Isso resulta em quatro regiões ao longo de cada eixo, isto é, regiões que tem um dos lados limitados pelos eixos. Destas quatro regiões, apenas uma não é limitada, portanto, neste nível, tem-se um total de 9(nove) regiões limitadas.



nível 2 - segunda visita aos eixos

Fig. 4.32

No próximo nível faz-se a terceira visita aos eixos, e assim por diante tal que, no nível i para k=2, o número de visitas feitas a cada um dos eixos K_1 e K_2 é i e o número de regiões geradas por essas visitas, ao longo de cada eixo, é 2^i. Como, dessas regiões, uma não é limitada, tem-se $(2^i - 1)$ regiões limitadas ao longo de cada eixo, o que dá um total de

$$(2^i - 1)\ (2^i - 1)$$

regiões limitadas no nível i. Essas são as regiões que serão

eliminadas da busca. Com isso, pode-se calcular o número total de regiões não limitadas em cada nível e depois em toda a árvore, o que dará o número total de nós visitados.

Voltando agora ao caso geral de registros com k chaves, tem-se que cada partição de uma região gera 2^k regiões. Assume-se que o número total de nós é $n = \frac{2^{hk} - 1}{2^k - 1}$ e todas as chaves são positivas. As regiões são particionadas por hiper-planos perpendiculares aos eixos. No nível zero tem-se todo o espaço inicial, logo, apenas uma região não limitada. No nível 1 tem-se hiperplanos perpendiculares a K_1, K_2, ..., K_k, o que é a primeira visita aos eixos K_1, K_2, ..., K_k. Essa primeira visita gera duas regiões ao longo de cada eixo, sendo uma não limitada. No nível 2 é feita a segunda visita e são geradas quatro regiões ao longo de cada eixo, uma das quais não é limitada. Em geral, em cada nível i é feita a i-ésima visita aos eixos e são geradas 2^i regiões ao longo do eixo K_i. Como uma só não é limitada, o número total de regiões limitadas nesse nível i é:

$$B(i) = (2^i - 1)^k$$

O número total de regiões não limitadas no nível i é:

$$S(i) = 2^{ki} - (2^i - 1)^k$$

Assim, o número de nós visitados, de acordo com as condições consideradas nesta análise, é a soma desse valor sobre todos os níveis da árvore, ou seja:

$$T(h) = \sum_{0 \leq i \leq h-1} S(i)$$

$$= \sum_{0 \leq i \leq h-1} \left[2^{ki} - (2^i - 1)^k \right]$$

usando a expansão binomial resulta:

$$T(h) = k \cdot \frac{2^{h(k-1)}}{2^{k-1} - 1} + O(2^{h(k-2)})$$

ou seja,

$$T(h) = O(k.n^{1-(1/k)})$$

4.5. *Conclusões*

É importante observar que a árvore Quad é apropriada para o caso particular discutido até agora, onde k=2, mas pode tornar-se inadequada do ponto de vista de armazenamento para dimensões maiores. Na árvore de duas dimensões necessita-se armazenar, para cada nó, quatro ponteiros além das chaves. Para o caso k-dimensional, cada nó tem 2^k filhos, portanto 2^k ponteiros, assim, o custo de armazenamento é $O(2^k n)$.

Em resumo tem-se os seguintes resultados a respeito de árvores Quad:

Com relação ao pré-processamento, os testes empíricos mostram que o tempo médio de inserção aleatória de nós é logarítmico em relação ao tamanho da árvore. Para construção da árvore otimizada, o tempo gasto é $O(n\log_2 n)$, logo, o custo de pré-processamento é:

$$P_Q(n,k) = O(n\log_2 n)$$

Quanto ao custo de armazenamento viu-se que,

$$S_Q(n,k) = O(2^k n)$$

Nas análises do algoritmo de busca em uma árvore Quad perfeita, com valores de chaves uniformemente distribuidos no espaço $[0,1]^2$ para k=2, o número esperado de nós visitados é:

$$E\ (n,x_1,x_2) \simeq x_1x_2n + (x_1+x_2)\ (3n)^{1/2} + \log_4(3n)$$

e para o caso geral,

$$E\ (n,x_1,\ldots,x_k) = n. \prod_{1\leq j\leq k} xj + O(n^{1-(1/k)})$$

Na análise do pior caso para árvores balanceadas, conclui-se que o número de nós visitados em uma busca por intervalo (T(h)), excluindo-se os subarquivos que estão inteiramente dentro ou inteiramente fora da região de busca, em uma árvore de h níveis é,

$$T\ (h) = O(k.n^{1-(1/k)})$$

portanto o custo dessa busca é:

$$Q_Q(n,k) = O(k.n^{1-(1/k)})$$

5. ÁRVORES K-D

A próxima estrutura denomina-se "árvore de busca k--dimensional", que é frequentemente abreviada como "árvore k-d" [BENTLEY-75] . Trata-se de uma generalização da árvore binária de busca de uma dimensão, para representar registros com várias chaves. Na árvore de busca unidimensional representa-se registros com apenas uma chave, que será o discriminador de todos os nós. A propriedade que define esta estrutura é de que todos os nós na subárvore esquerda de um nó X representam registros com valor de chave menor ou igual ao da chave de X e os nós da subárvore direita contém registros com valor de chave maior que o da chave de X. Esse valor da chave será, para todos os nós, o valor básico para as comparações que determinam a direção dos processos de busca e inserção.

Por exemplo, a figura 5.1 mostra uma árvore binária de busca com valores numéricos nas chaves.

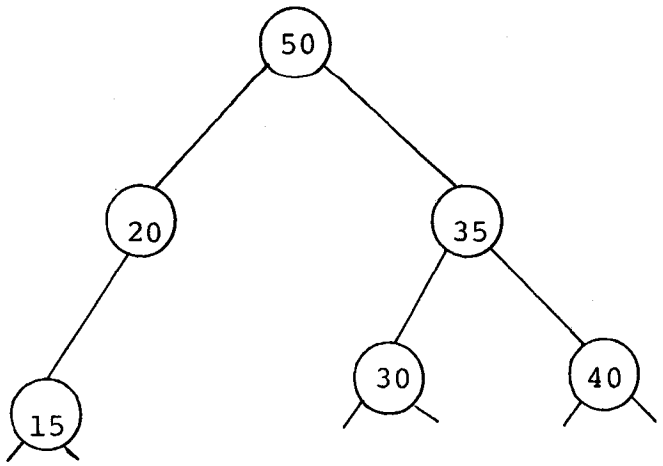


Fig. 5.1

Para verificar se um determinado nó Y encontra-se na árvore, compara-se a partir da raiz, o valor da chave do nó Y com o valor da chave do nó que está na árvore. Se ambos os valores são iguais o nó foi encontrado, caso contrário, a busca deve continuar recursivamente pela subárvore esquerda se a chave do nó Y for menor ou pela subárvore direita se a chave do nó Y for maior. O processo termina quando o nó é localizado ou quando encontra-se uma subárvore vazia, o que significa que o nó procurado não está na árvore. Para inserir um novo elemento o procedimento básico é o mesmo sendo que continua-se com as comparações até encontrar um ramo vazio, isto é, um ramo que leve a uma subárvore vazia, que é onde deve ser inserido o novo nó. No processo de inserção deve estar determinado se nós com mesmo valor de chave, se existirem, serão inseridos mais de uma vez. Em caso afirmativo, deve estar convencionado se o nó com valor repetido entrará à direita ou à esquerda daquele que tem o mesmo valor. O processo básico de construção da árvore consiste em, começando com a estrutura vazia, aplicar o algoritmo de inserção para cada um dos nós que devem ser inseridos, na ordem em que aparecem. Uma variação desse processo permite a construção de uma árvore balanceada, o que garante um melhor tempo de busca, ou seja:

1 - ordenar os valores das chaves de todos os nós

2 - escolher o nó cujo valor de chave seja o valor médio de todos os nós, para ser a raiz

3 - aplicar recursivamente o passo 2 às duas subcoleções obtidas pela escolha da raiz.

Para regitros com várias chaves essa estrutura é inadequada, pois apenas uma chave é usada para organizar a árvore, mas ela pode ser generalizada para ser usada com conjuntos de registros com k chaves cada um. Nos processos de inserção e busca para árvore binária simples, as comparações são feitas com base em uma única chave para tomar a decisão de prosseguir pela subárvore esquerda ou direita. Em uma árvore multidimensional (árvore k-d) essa discriminação é feita com base em chaves diferentes. A cada nó da árvore, um campo de chave será usado como "discriminador" ou "discriminante". Um nó cujo discriminador ou discriminante é a chave K_j, é dito um j-discriminante. Formalmente pode-se definir árvore k-d como uma árvore binária onde todos os nós tem um discriminante e para todo nó X que é um j-discriminante todos os nós na subárvore esquerda de X tem valores de K_j menores que o valor K_j de X e todos os nós na subárvore direita de X tem valores de K_j maiores ou iguais que o K_j de X.

Uma possibilidade de escolha de discriminantes para registros com k chaves $K_1, K_2, \ldots, K_k$, é usar a primeira chave K_1 como discriminador do nível 0(raiz), usar a chave K_2 como discriminador do nível 1, e assim por diante até o nível k-1. No nível k usa-se novamente a primeira chave reiniciando o ciclo. Portanto, para nós no nível i, o discriminante é dado por (i mod k) + 1, assumindo que a raiz está no nível 0.

A seguir, a aplicação de árvores k-d para registros com duas chaves é exemplificada pela inserção dos registros A,B,C, ... numa árvore k-d inicialmente vazia, escolhendo

discriminantes cíclicamente.

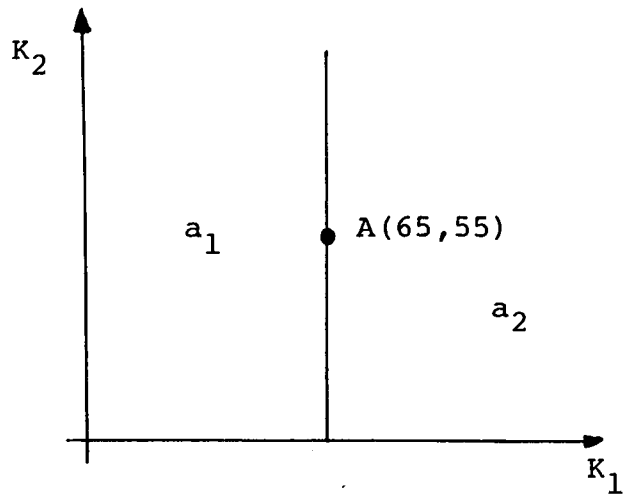


Fig. 5.2

Supondo-se que a raiz se localiza na posição (65,55), como mostra a figura 5.2, seu discriminante será a chave K_1,com valor K_1=65. A linha vertical que passa pelo ponto A divide o espaço onde se localizam os registros em 2 subespaços, a_1 e a_2 da figura 5.2, que formarão as subárvores esquerda e direita de A, respectivamente, originando a árvore mostrada na figura 5.3.

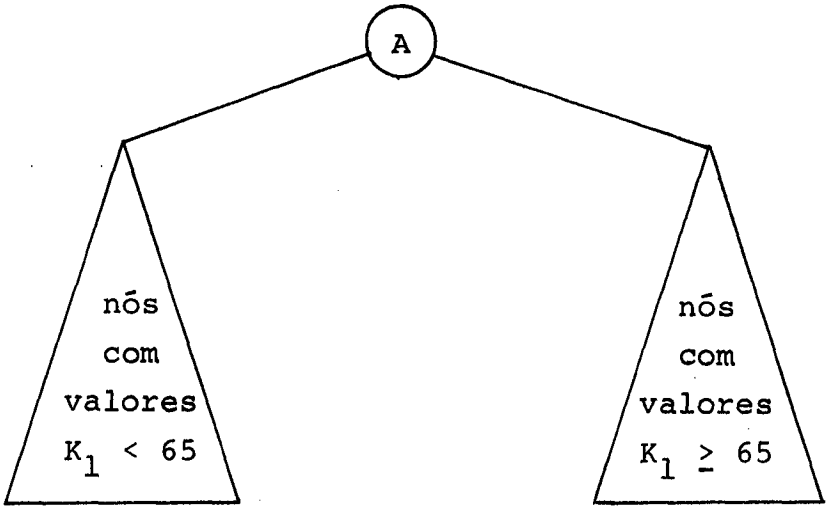


Fig. 5.3

Está convencionado que nós localizados exatamente sobre a linha vertical que passa por A, isto é, que tem valores de K_1 iguais ao de A, localizam-se na subárvore direita de A. Por isso a subárvore direita de A tem nós com valores de $K_1 \geq 65$.

Assumindo-se agora que os filhos à esquerda e à direita do nó A são os nós B e C, respectivamente, B localiza-se na região a_1 da figura 5.2 e representa essa região. O nó C localiza-se na região a_2 da figura 5.2 e representa essa região.

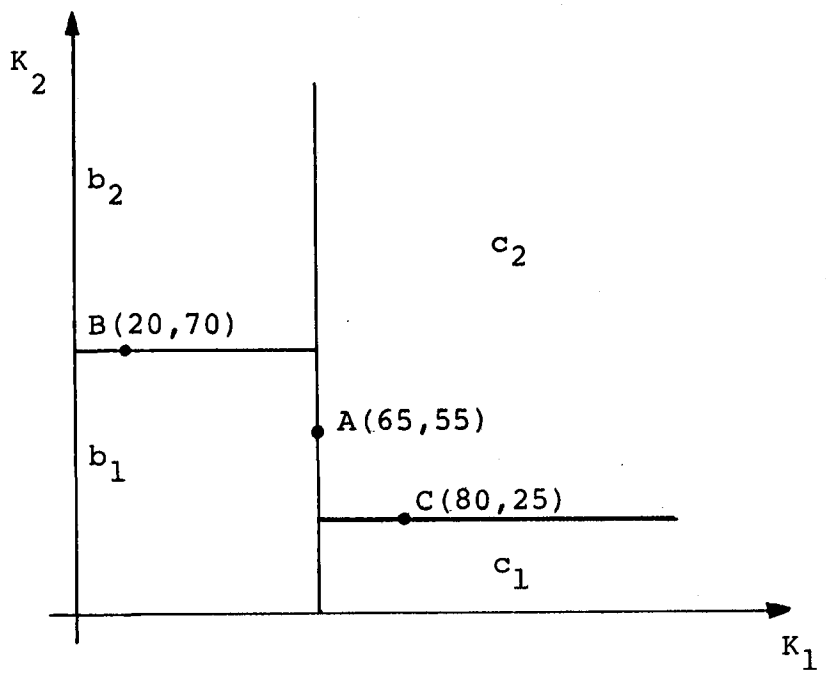


Fig. 5.4

Se os nós B e C tiverem por exemplo as localizações mostradas na figura 5.4, tem-se que as linhas horizontais que passam por B e C, isto é, $K_2=70$ e $K_2=25$ dividem os espaços à esquerda e à direita de A em duas partes. Assim, os pontos que se localizam abaixo da linha $K_2=70$, portanto tem $K_2<70$, e que já estão limitados por $K_1<65$ pois estão à esquerda de A, (região b_1 da figura 5.4) ficarão na subárvore esquerda de B. Os nós com valores $K_2\geq 70$ e $K_1<65$, (região b_2 da figura 5.4), ficarão na subárvore direita de B. Análogamente, os pontos abaixo da linha $K_2=25$ e com $K_1\geq 65$, (região c_1 da figura 5.4) ficam na subárvore esquerda de C e os pontos com valores $K_2\geq 25$ e $K_1\geq 65$, (região c_2 da figura 5.4) ficarão na subárvore direita de C. Assim, a árvore equivalente a esses nós tem a forma da figura 5.5.

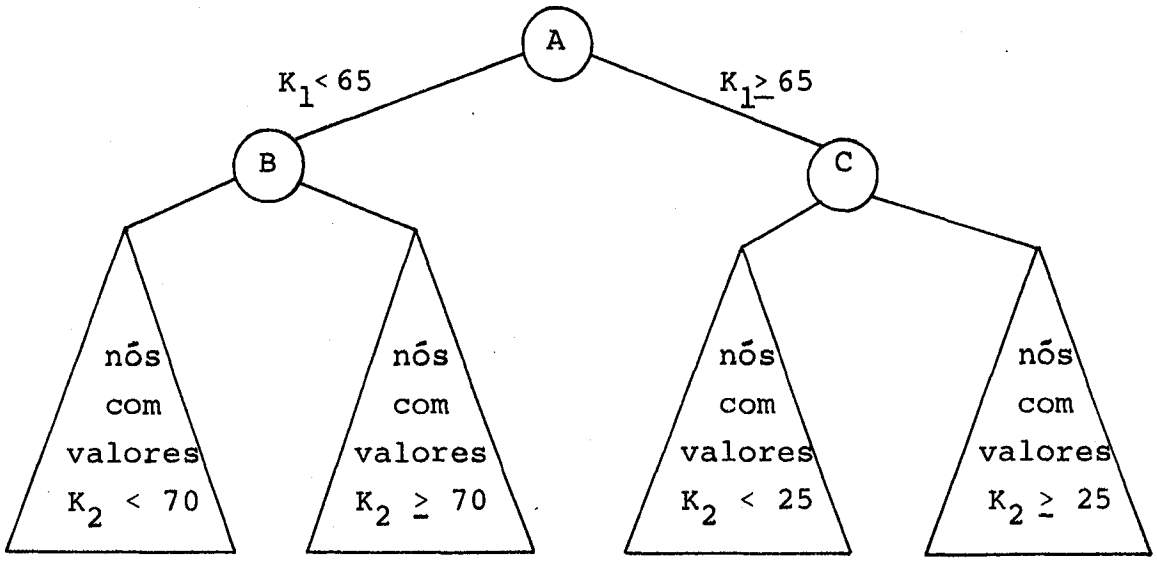


Fig. 5.5

No próximo nível, o discriminante será novamente a chave K_1. Se os filhos dos nós B e C estão nas posições mostradas na figura 5.6 as linhas verticais que passam pelos nós D, E, F e G, dividem o subespaço onde eles estão em duas partes, o que determina quais nós desse subespaço ficam à direita ou a esquerda de D, E, F e G.

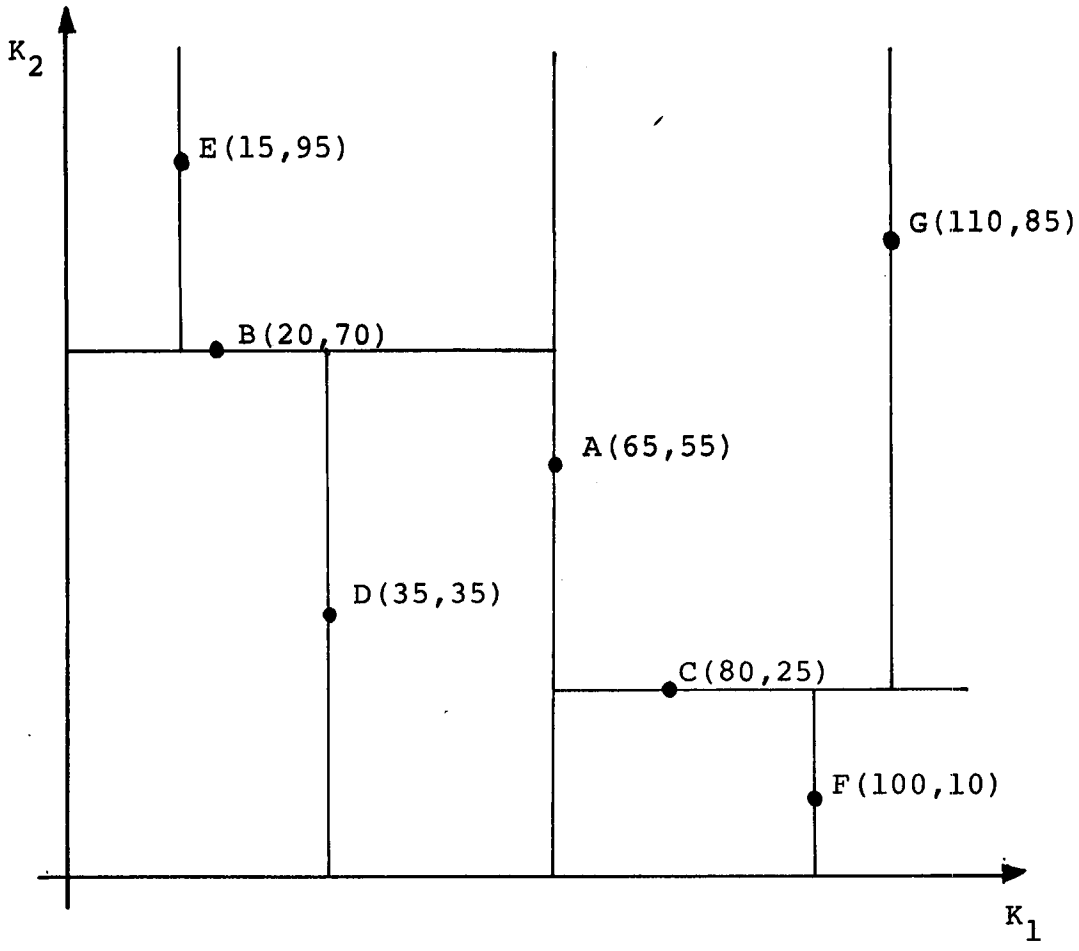


Fig. 5.6

Por exemplo, para o nó D, que está abaixo de B, portanto na subárvore esquerda de B, a linha que determina suas subárvores é K_1=35. Assim, nós no subespaço representado por D e com valores K_1<35, ficam na subárvore esquerda de D. Os nós que ficam no subespaço representado por D e com valores $K_1 \geq 35$, ficam na subárvore direita de D.

Assim por diante, em cada nível, cada subespaço limitado pelas retas horizontais ou verticais que passam pelos nós dependendo se o discriminante é K_1 ou K_2, é dividido em outros dois. O espaço todo é particionado em conjuntos disjuntos, até

que todos os nós estejam representados na árvore. O subespaço que é representado por um determinado nó, é dito "retângulo limite" desse nó.

A figura 5.7 mostra um exemplo completo com os nós distribuídos no plano e a figura 5.8, sua árvore equivalente até o nível 5, sendo que os quadrados representam subárvores vazias.

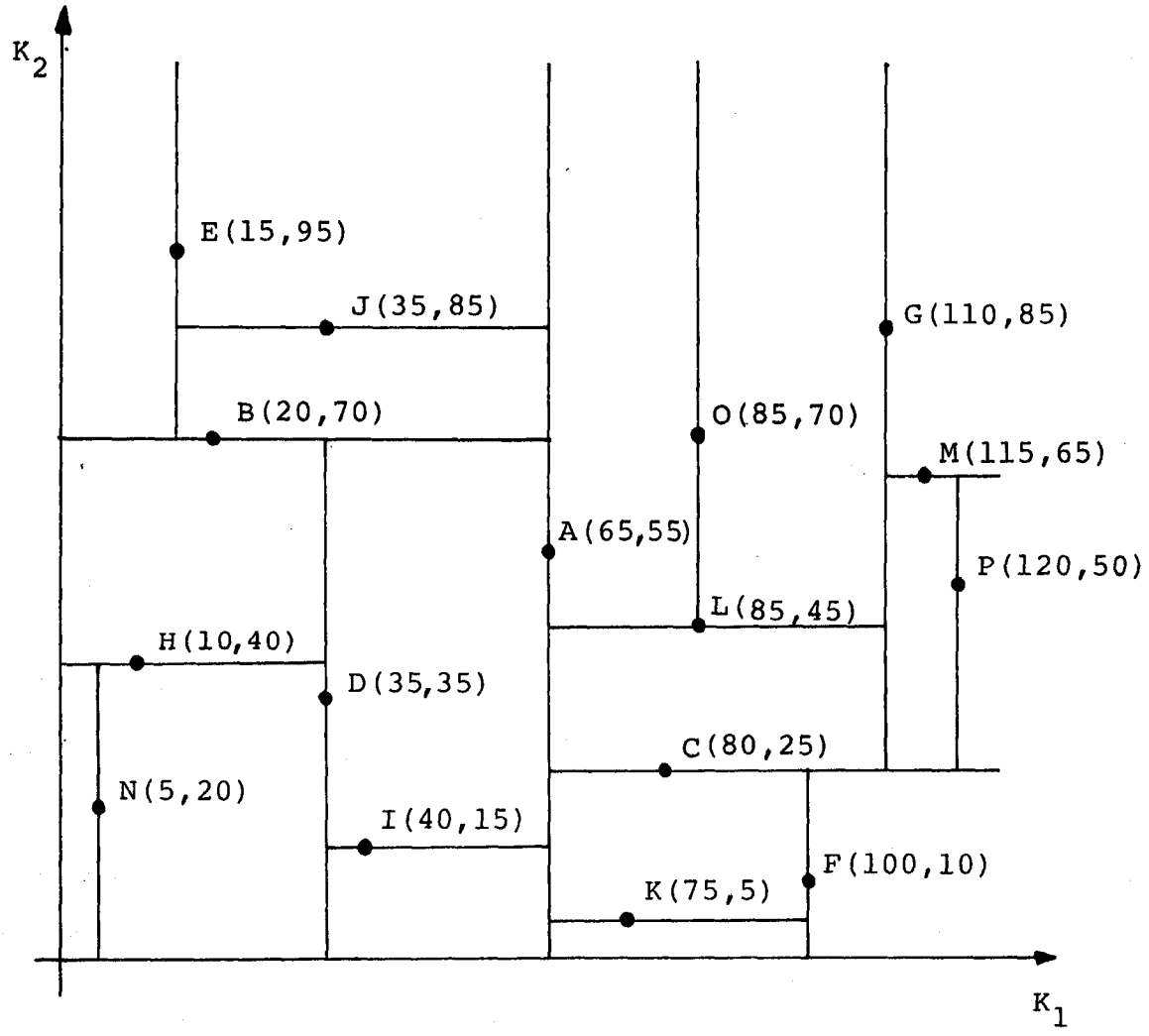


Fig. 5.7

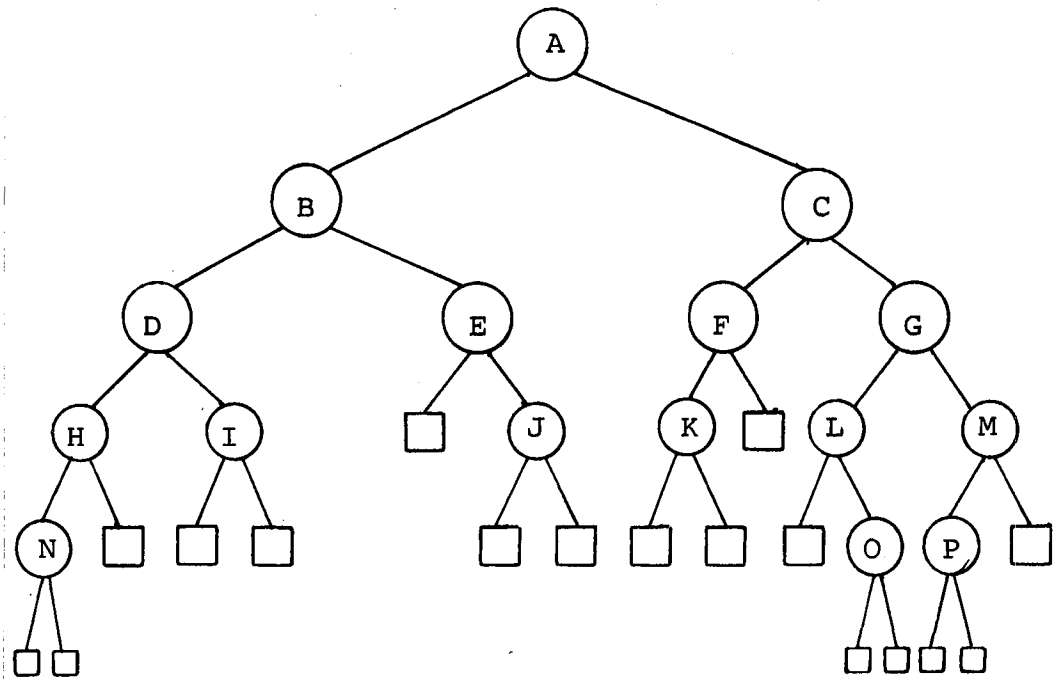


Fig. 5.8

Assim como para árvores Quad, não estão representados os nós com valores de chaves iguais. Esses nós podem ser representados, por exemplo, em forma de lista encadeada onde cada nó da árvore é a origem de uma lista que contém todos os nós com valores das chaves iguais. Quando se fala em "nós com valores iguais" ou "nós repetidos" refere-se a nós com valores de chaves iguais, sendo que eles podem ter outros atributos, iguais ou diferentes.

Uma outra maneira de escolher os discriminantes da árvore é dada por [BENTLEY-79c] onde o discriminante de cada nó é escolhido como a chave K_j para a qual a variação dos valores entre todos os nós é máxima. Essa variação pode ser medida por exemplo com a variância dos valores ou distância entre

mínimo e máximo. Se um nó é um j-discriminante, o valor desse discriminante é a mediana entre todos os valores da chave Kj. Em [SILVA-81] , Silva Filho apresenta uma maneira de fazer a escolha ótima de discriminantes em uma árvore k-d balanceada. Os discriminantes são escolhidos por nível, de tal maneira que o número esperado de visitas a nós durante a busca é minimizado.

A maior vantagem da estrutura de árvores k-d é o fato de que uma única estrutura de dados se adapta a vários tipos diferentes de consulta. Mostra-se também uma estrutura adequada no caso de busca por intervalo em que o tamanho das regiões pode variar bastante.

5.1. Inserção

Assim como para árvores Quad, o procedimento básico para inserção de um novo nó em uma árvore k-d é o mesmo utilizado para árvores binárias. Começando pela raíz, compara-se o nó a ser inserido com cada nó da árvore para decidir se deve-se prosseguir pela subárvore esquerda ou direita, até encontrar uma subárvore vazia, que é onde deve ser inserido o nó ou até encontrar um nó com os mesmos valores de chaves. A comparação entre os nós é feita com base no campo de chave que é o discriminante do nó que está na árvore, e cada nó que é inserido passa a ter um discriminante. Quando um nó repetido é encontrado na árvore no processo de inserção, deve-se dar o tratamento adequado, de acordo com o que tenha sido convencionado

anteriormente, isto é, se ele deve ou não ser inserido e de que forma, em caso afirmativo. É descrito a seguir um algoritmo para inserir um nó em uma árvore k-d que também pode ser usado para fazer a busca de um registro específico na árvore.

5.1.1. *Algoritmo de Inserção*

O algoritmo de inserção a ser detalhado está baseado no algoritmo "Insert" de [BENTLEY-75a] . O procedimento INSERIR faz a inserção de um único nó na árvore e para construir esta árvore ele deve ser chamado uma vez para cada nó a ser inserido. Nesta implementação, quando encontra-se na árvore um nó com os mesmos valores de chave que o nó que se quer inserir, a inserção não é feita, uma mensagem é enviada com a chamada do procedimento NO_JA_ESTA_NA_ARVORE (X) e a rotina termina. Os discriminantes dos nós são escolhidos cíclicamente, um para cada nível da árvore.

Tanto o algoritmo de inserção como o de busca utilizam uma estrutura de dados com as seguintes características:

- os nós são registros com k campos reais para as chaves, dois campos de ponteiro para as subárvores e um campo de discriminante , cujo valor pode ser um inteiro de 1 a k.

- os ponteiros são do tipo NO a ser definido; os campos de ponteiros de um nó X são denotados por X↑.SUB[1] e X↑.SUB[2] para subárvores esquerda e direita respectivamente;os campos das chaves são denotados por X↑.CHAVE [I] para $1 \leq I \leq K$ e o

campo de discriminante por X↑.DISC.

- o registro tem a forma:

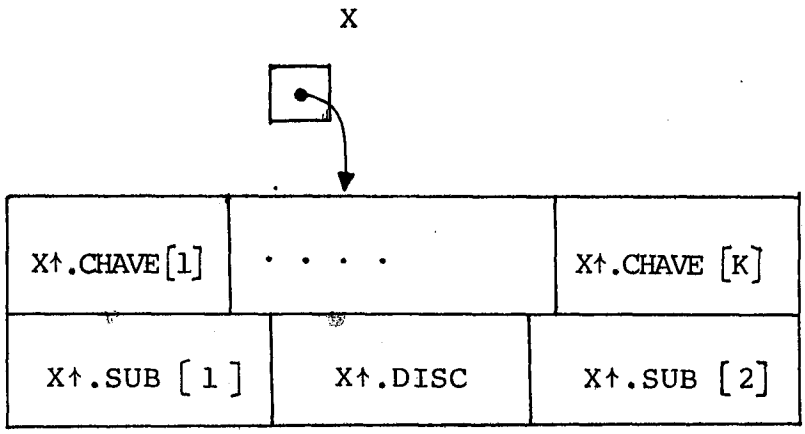


Fig. 5.9

- não estão representados outros atributos que possam existir além das chaves.

- supondo que RAIZ aponta para a raiz da árvore e X aponta para o nó a ser inserido, a figura 5.10 mostra parte da estrutura ao ser iniciado o processo para inserção de X, onde a ligação terra (⏚) representa ponteiro nulo:

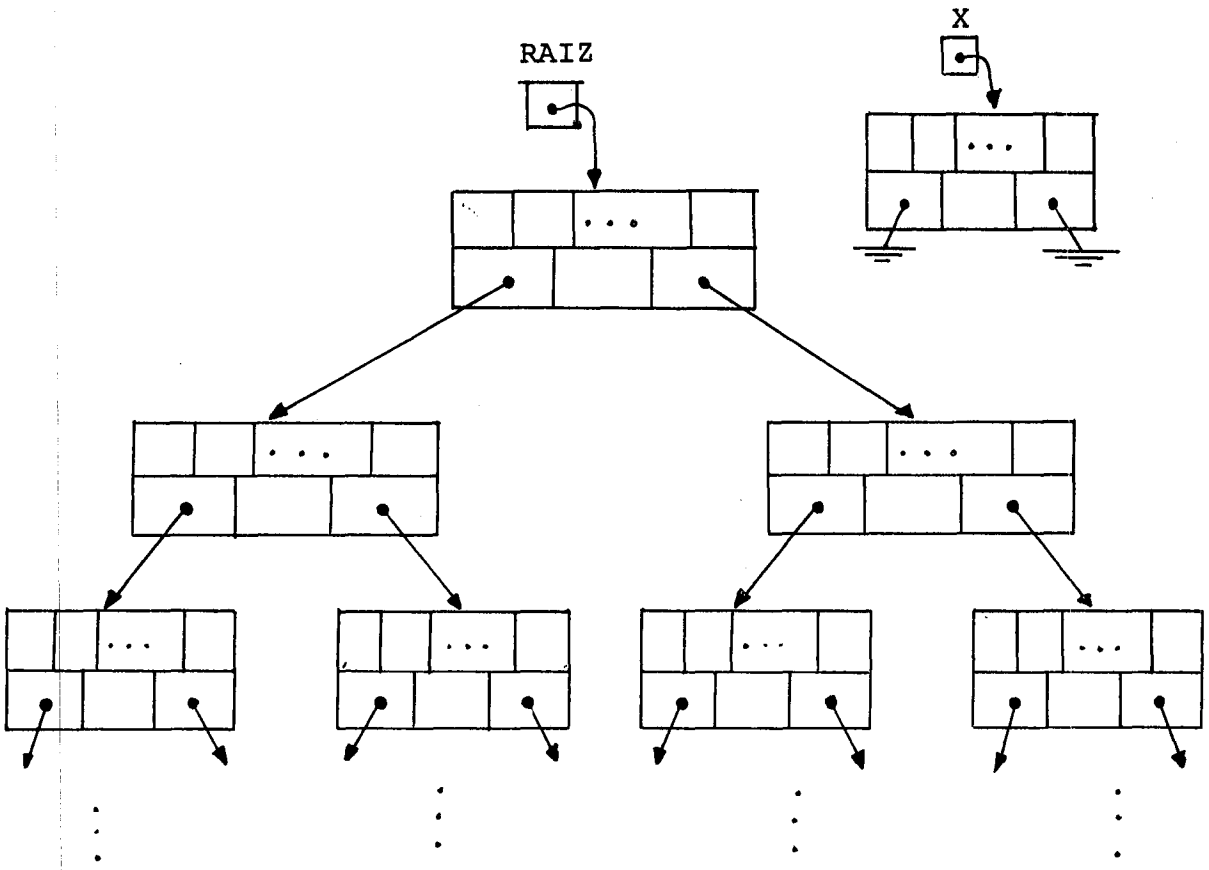


Fig. 5.10

A definição de tipos e declaração de variáveis para essa estrutura pode ser como segue:

```
const K=10;
type NO= ↑REGISTRO;
     REGISTRO = record
                CHAVE : array [1..K] of real;
                SUB: array [1..2] of NO;
                DISC: 1...K
                end;
var   RAIZ,X : NO;
```

O algoritmo para inserção detalhado adiante assume como válidas as seguintes observações:

- os parâmetros do procedimento INSERIR são R e X onde R é a raiz da árvore e X é o nó que vai ser inserido.
- os campos de chave de X estão inicialmente com os valores das chaves de X e os campos de ponteiro estão nulos bem como do discriminante, que será atualizado na hora da inserção.
- quando a árvore está inicialmente vazia o nó é inserido como raiz sem entrar no processo de iteração.
- são utilizadas em INSERIR as seguintes funções e procedimentos:
  1. função SUCESSOR (Q,X), onde Q é ponteiro para um nó de árvore e X é ponteiro para o nó que vai ser inserido.Retorna um valor 1 ou 2 indicando se o processo deve continuar pela subárvore esquerda ou direita de Q, respectivamente.
  2. função booleana COMPARE (Q,X) onde Q é ponteiro para um nó da árvore e X é ponteiro para o nó que vai ser inserido.Retorna valor"true" se X é igual a Q e "false" caso contrário.
  3. o procedimento INSERE_NO (Q,X,I) onde Q é ponteiro para um nó da árvore, X é ponteiro para o nó que vai ser inserido e I é inteiro. Insere X como filho I de Q e atualiza seu campo de discriminante.
  4. O procedimento NO-JA-ESTA-NA-ARVORE (X) onde X é ponteiro para o nó que está sendo inserido. É chamado quando o nó X já existe na árvore e dá o tratamento adequado a esse nó. Este procedimento não está detalhado.
- supondo que RAIZ aponta para a raiz da árvore e X aponta para o nó que vai ser inserido para o procedimento é chamado com os parâmetros:

INSERIR (RAIZ, X)

O procedimento INSERIR e as funções e procedimen - tos que ele utiliza podem ser detalhados como segue:

```
procedure INSERIR (var R : NO ; X : NO);
(*inserir nó X na árvore com raiz R*)
var  Q, F : NO
     ACHOU : boolean;
     I : integer;
begin if R =  nil then (*arvore vazia*)
                  begin R:=X;
                        R↑.DISC:=1
                  end
 else begin
       Q:=R; (*Q desce pela arvore)
       F:=R; (*F-filho de Q,inicialmente não nulo*)
       ACHOU:= COMPARE (Q,X);
       (*se Q=X não inicia processo de iteração*)
       while not ACHOU  do
        begin
             I:= SUCESSOR (Q,X);
             F:= Q↑.SUB [ I ]
             if          F = nil
             then (*achou lugar de inserir nó*)
                  ACHOU:= true
             else begin (*desce um nível na árvore*)
                   Q:= F;
                   ACHOU:= COMPARE (Q,X)
                  end
        end;
       if  F =nil  then   INSERE _NO (R,X,I)
                   else  NO_JA_ESTA_NA_ARVORE (X)
      end

end;
```

```
function   SUCESSOR (Q, X:NO) : 1..2;
(*determina por qual subárvore de Q o processo
 para inserir X deve continuar*)
var      I: integer;
begin
         I:= Q↑.DISC;
         if X↑.CHAVE [ I ] < Q↑.CHAVE [ I ]
         then  (*subárvore esquerda*)
               SUCESSOR:=1
         else  (*subárvore direita*)
               SUCESSOR:=2
end;

function   COMPARE (Q, X:NO) : boolean;
(*determina se Q é igual a X ou não*)
var      I: integer;
         TESTE: boolean;
begin    I:=1; COMPARE:= true;
         while (I≤K) and  TESTE  do
         begin if  Q↑.CHAVE [I]<> X↑.CHAVE [I]
               then (*se uma coordenada diferen
                      te, ponto diferente*)
                      TESTE:= false;
               I:=I+1 (*próxima chave*)

         end;
         COMPARE:= teste;
end;
```

```
procedure   INSERE_NO (Q, X:NO; FILHO:integer);
(* insere nó X na arvore como subarvore FILHO de Q *)
begin
      Q↑.SUB [FILHO] : = X;
      X↑.DISC : = (Q↑.DISC mod K)+1
end;
```

5.1.2 *Propriedades*

Um resultado importante da análise de árvores k-d de [BENTLEY-75a] mostra que: considerando-se uma árvore binária de n nós, a probabilidade de construir essa árvore inserindo-se n nós randômicos em uma árvore k-d inicialmente vazia é a mesma que a probabilidade de obter aquela árvore por inserção randômica em uma árvore binária de busca unidimensional. Assim, os teoremas que tem sido provados sobre árvore de busca binária unidimensional são aplicáveis a árvores k-d.

Portanto, se o algoritmo INSERIR for usado para construir uma árvore k-d com nós randômicos, essa árvore terá as mesmas boas propriedades que uma árvore binária.

Entre todos os resultados sobre árvore binária unidimensional aplicáveis a árvores k-d podemos aproveitar dois de [KNUTH-73] com relação à inserção e busca de um novo nó.

Seja C_n o número médio de nós visitados para encontrar um nó em uma árvore k-d com n nós. Então,

$$C_n = 2(1 + 1/n)\ H_n - 3$$

$$\simeq 1.386\ \log_2 n$$

e a variância é:

$$Var(C_n) = (2 + \frac{10}{n})\ H_n - 4(1 + \frac{1}{n})\ (H_n^{(2)} + \frac{H_n^{(2)}}{n}) + 4$$

onde

$$H_n^{(x)} = \sum_{1 \leq k \leq n} 1/k^x$$

e

$$H_n = H_n^{(1)}$$

A fórmula da variância aparece em [KNUTH-73], pg 672, onde é atribuída a G.D. Knott. A expressão contém erros na primeira impressão, que foram corrigidos na segunda impressão de março de 1975.

A afirmação de que o número médio de comparações para encontrar um nó é aproximadamente $1.386\ \log_2 n$ pode ser facilmente provada.

Seja C_n o número médio de comparações em uma busca com sucesso assumindo que cada um dos n nós é igualmente provável; seja C'_n o número médio de comparações em uma busca com fracasso, assumindo que cada um dos n+1 intervalos entre registros é igualmente provável.

Então, pela definição de comprimento de caminho externo e comprimento de caminho interno [KNUTH-68] tem-se:

$$C_n = 1 + \frac{\text{comprimento de caminho interno da árvore}}{n}$$

$$C'_n \triangleq \frac{\text{comprimento de caminho externo da árvore}}{n + 1}$$

Como o comprimento de caminho externo (E) é sempre 2N mais que o comprimento de caminho interno (I), ou seja,

$$E = I + 2N$$

obtêm-se a seguinte relação de C_n e C'_n

$$C_n = (1 + \frac{1}{n}) C'_n - 1 \qquad [5.1]$$

Por outro lado, o número de comparações necessárias para encontrar um registro é exatamente um a mais do que o número de comparações que foram necessárias quando esse nó foi inserido na árvore. Portanto se C_n é o número médio de comparações envolvidas em uma busca com sucesso e C'_n é o número médio em uma busca sem sucesso, tem-se:

$$C_n = 1 + \frac{C'_0 + C'_1 + \ldots C'_{n-1}}{n} \qquad [5.2]$$

Substituindo-se a equação [5.1] em [5.2] resulta:

$$(n+1)\ C'_n = 2n + C'_0 + C'_1 + \ldots C'_{n-1} \qquad [5.3]$$

A equação [5.3] é uma recorrência e subtraindo-se dela a equação:

$$n\ C'_{n-1} = 2(n-1) + C'_0 + C'_1 + \ldots C'_{n-2}$$

vem:

$$(n+1)\ C'_n - n\ C'_{n-1} = 2 + C'_{n-1}$$

$$C'_n = C'_{n-1} + 2/(n+1) \qquad [5.4]$$

Resolvendo-se as equações [5.4] com a condição inicial :

$$C'_0 = 0$$

resulta:

$$C'_n = 2\ H_{n+1} - 2$$

Aplicando [5.1] nesta equação obtem-se:

$$\frac{(C_n + 1)\ n}{n + 1} = 2\ H_{n+1} - 2$$

e simplificando, vem:

$$C_n = 2\ (1 + 1/n)\ H_n - 3$$

Portanto, tem-se que:

$$C_n \simeq 2\ \ell n\ n$$

$$\simeq 1.386\ \log_2 n$$

Conclui-se que inserções e busca exata em uma árvore k-d vão pesquisar aproximadamente $1.386\ \log_2 n$ nós.

5.2 *Busca*

Faz-se a busca por intervalo em uma árvore k-d com o seguinte procedimento:

A cada nó visitado, começando-se pela raiz, verifica-se se ele está dentro da região de busca ou não. Em seguida, se o nó é um j-discriminante, isto é, tem como discriminante a chave K_j, compara-se o valor da chave K_j desse nó com o

j -ésimo intervalo especificado na consulta.

Se o intervalo está inteiramente abaixo do valor do discriminante, a busca continua na subárvore esquerda do nó e se ele está acima incluso o valor de K_j, a busca continua na subárvore direita. Nestes dois casos, continuando-se a busca por uma das duas subárvores, a outra já está eliminada, pois os nós que ela contém não podem estar na região de busca. Se o valor do discriminante está dentro do intervalo, a busca continua recursivamente nas duas subárvores.

Por exemplo, se tivermos um conjunto de registros com duas chaves e uma consulta determinando uma região de busca com as localizações mostradas na figura 5.11, sua representação na forma de árvore será conforme mostra figura 5.12.

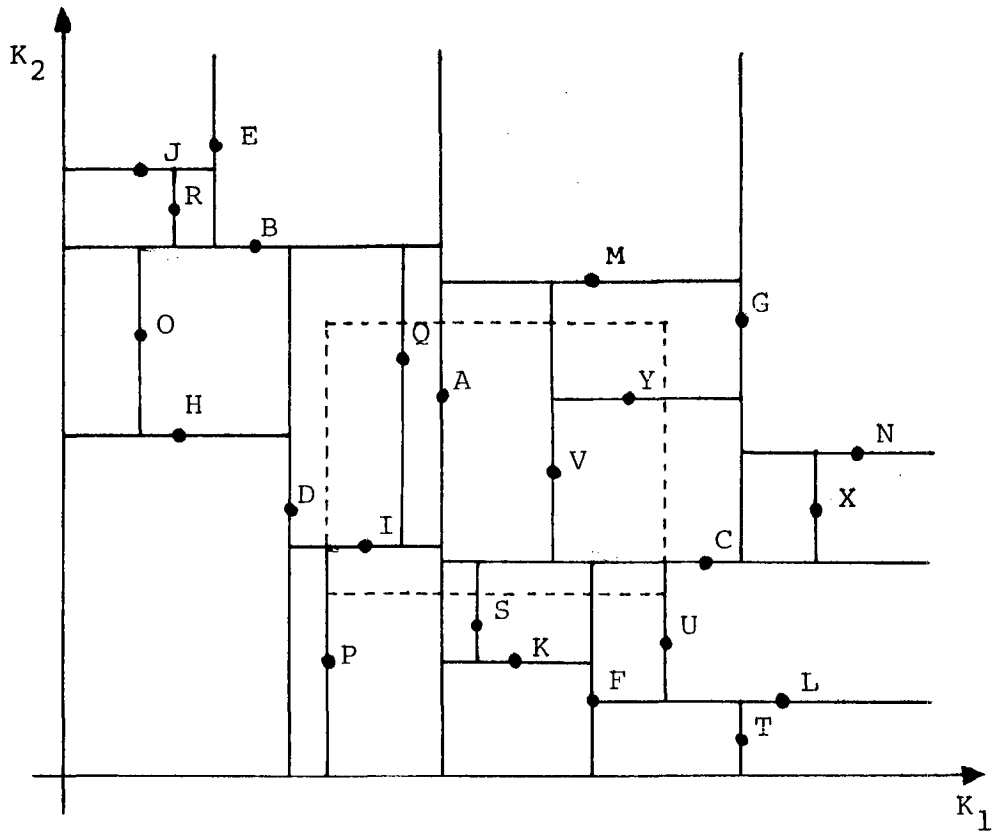


Fig. 5.11

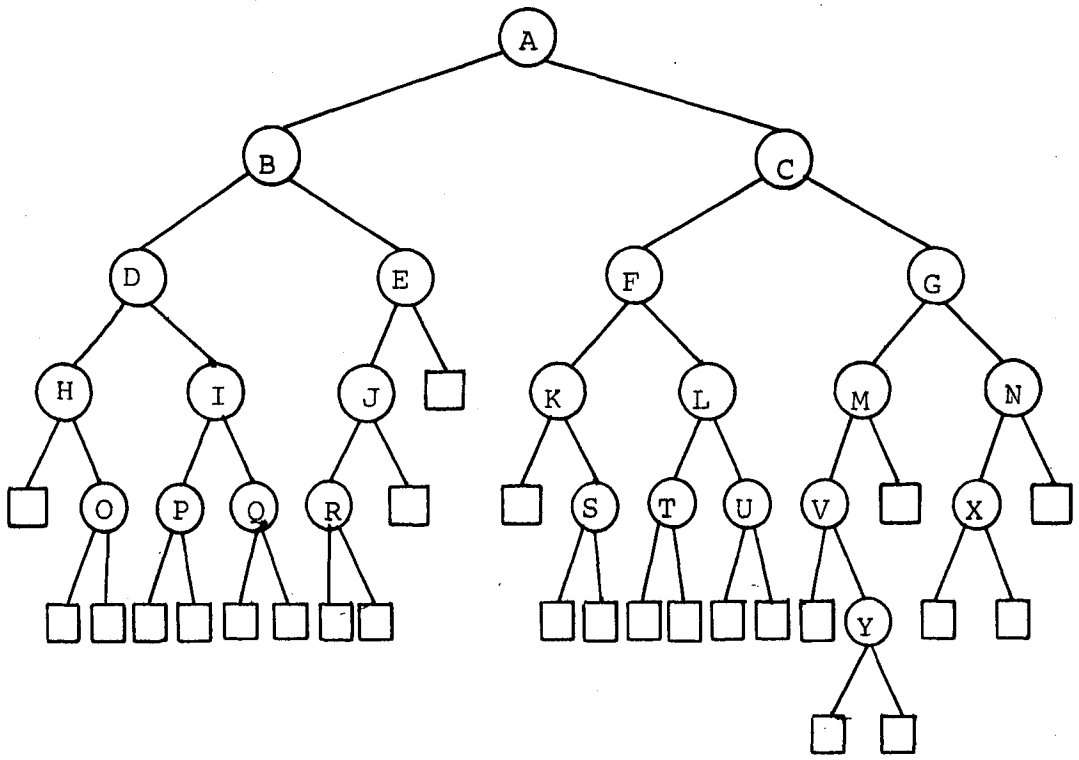


Fig. 5.12

Ao aplicar o processo de busca nesta árvore, ini - ciam-se as comparações com o nó A que é a raiz.

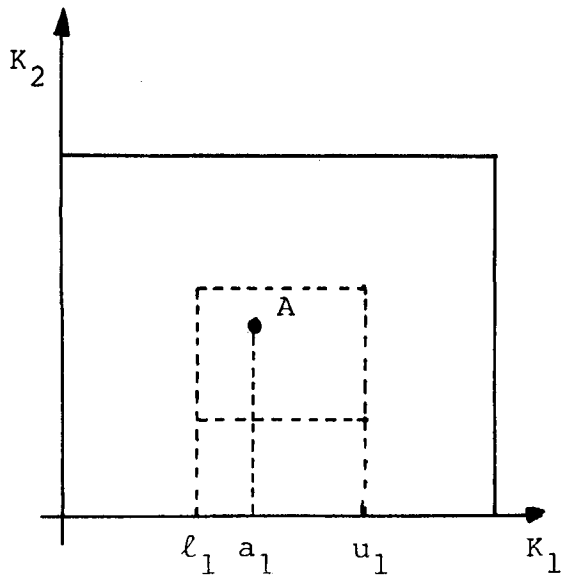


Fig. 5.13

O nó A localiza-se na região de busca (figura 5.13) portanto é um dos nós que estão sendo procurados e será processado. Além disso, como nesse nível o discriminante é K_1 compara-se o valor de K_1 de A, que é a_1, com o primeiro intervalo da consulta, dado por ℓ_1 e u_1. O valor a_1 está dentro do intervalo $[\ell_1, u_1]$, o que significa que a região de busca se sobrepõe tanto ao retângulo limite da subárvore esquerda de A como ao retângulo limite da subárvore direita de A, logo, as duas subárvores serão pesquisadas.

A figura 5.14 mostra a situação no próximo nível.

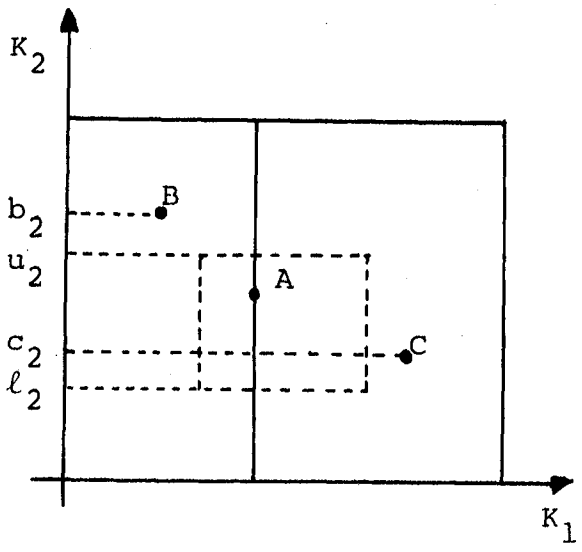


Fig. 5.14

O nó B não se localiza na região de busca. Como seu discriminante é K_2, e seu valor b_2 está acima do intervalo $[\ell_2, u_2]$, apenas sua subárvore esquerda será pesquisada. Não há possibilidade de que nós com valor de $K_2 \geq b_2$ estejam na região de busca, portanto a subárvore direita de B não será mais pesquisada. O nó C também não está na região de busca mas seu valor de K_2 está dentro do intervalo para K_2, porisso suas duas subárvores serão pesquisadas.

Considerando que no próximo nível o discriminante é novamente K_1, a figura 5.15 mostra a nova situação:

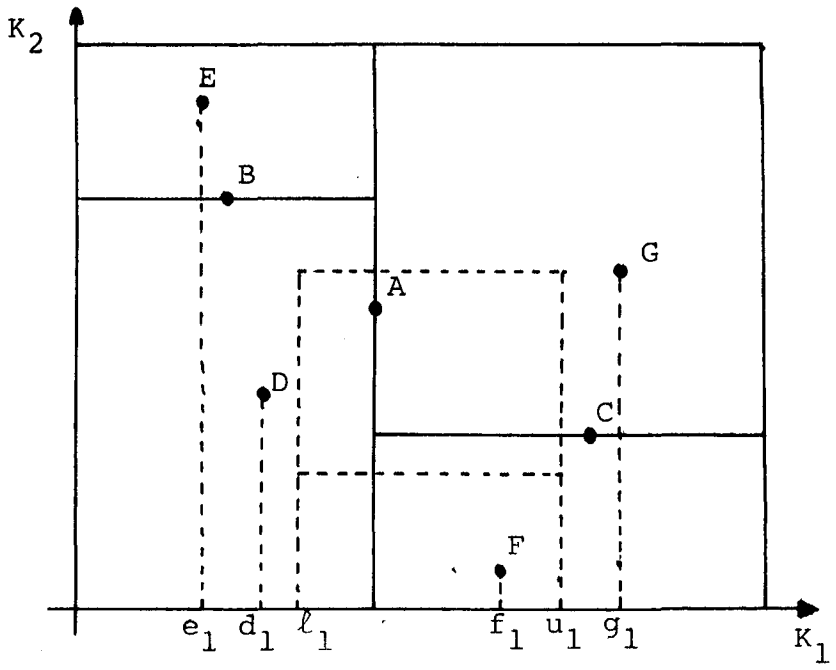


Fig. 5.15

Nenhum dos nós D, E, F, G localiza-se na região de busca, e o processo continua pela subárvore direita de D , subárvore esquerda de G e pelas duas subárvores de F.

Assim, a busca prossegue até encontrar apenas subárvores vazias.

A figura 5.16 indica o que ocorre com toda a árvore deste exemplo depois de terminada a busca.

Os traços transversais sobre os ramos da árvore

indicam as subárvores que foram eliminadas durante o processo. Os nós hachurados são os que se localizam dentro da região de busca. Os nós quadrados representam subárvores vazias.

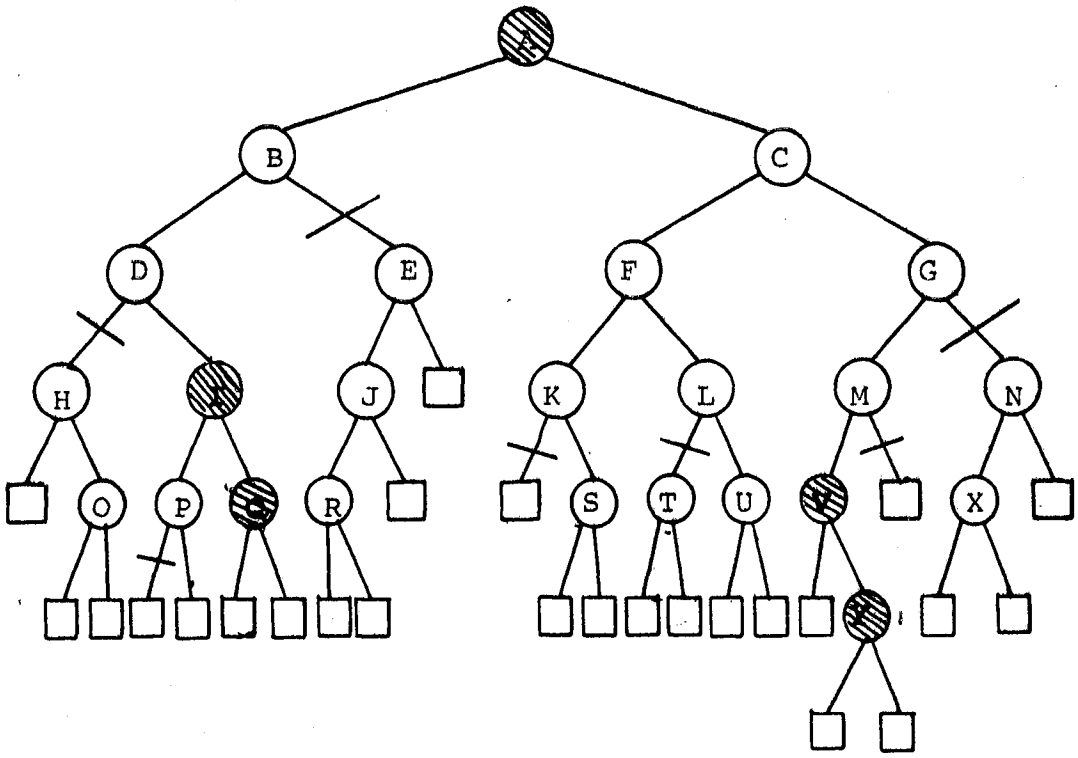


Fig. 5.16

5.2.1. *Algoritmo de Busca*

O algoritmo que realiza a busca por intervalo sobre árvores k-d a ser descrito (BUSCA-INTERVALO) está baseado no algoritmo "Regionsearch" apresentado em [BENTLEY-75a].

Assume-se a mesma estrutura de dados definida para o algoritmo de inserção, além do vetor REGI de 2k posições que contém os limites dos intervalos que determinam a região de busca. As posições 2I-1 e 2I contém os limites inferior e superior respectivamente do intervalo da I-ésima chave. Portanto deve-se acrescentar o seguinte às declarações anteriores:

```
const DOISK = 20;
var   REGI : array [1..DOISK] of real;
```

O procedimento BUSCA-INTERVALO utiliza uma função e dois procedimentos definidos como segue:

- função booleana ESTA_NA_REGIÃO (X), onde X é um ponteiro para um nó. É chamada a cada nó visitado para verificar se ele se localiza na região de busca. Retorna valor "true" se o nó se encontra na região e "false" caso contrário.

- procedimento ARVORE_VAZIA. É chamado quando a árvore está inicialmente vazia. Este procedimento não está detalhado.

- procedimento ACHOU (X), onde X é um ponteiro para um nó. É chamado quando um nó X se localiza na região de busca, para processá-lo. Este procedimento não está detalhado.

O procedimento BUSCA-INTERVALO tem um único parâmetro que é um ponteiro para a raiz da árvore sobre a qual será aplicada a busca. Se RAIZ é esse ponteiro para a raiz da árvore, o procedimento será chamado por:

```
BUSCA_INTERVALO (RAIZ);
```

O procedimento detalhado é apresentado a seguir:

```
procedure   BUSCA_INTERVALO (R:NO);
(* Busca de todos os nós de uma árvore k-d com raiz R
  que estão em uma região retangular dada por REGI *)
var  I : integer;
begin
if R = nil  then     ARVORE_VAZIA
else begin
     if   ESTA_NA_REGIÃO(R)  then   ACHOU(R);
     I:= R↑.DISC; (*I-discriminante de R*)
     if   (R↑.SUB [1]<> nil ) and
          (R↑.CHAVE [I]  >  REGI  [2*I-1])
     then (* se subárvore esquerda não vazia e  a
             região de busca se sobrepõe ao seu retângulo limite, chamar recursivamente*)
             BUSCA_INTERVALO  (R↑.SUB [ 1 ] );
     if   (R .SUB [ 2 ] <> nil)  and
          (R↑.CHAVE [I] ≤ REGI  [ 2*I ]  )
     then (* se subárvore direita não vazia e a região
             de busca se sobrepões ao seu retângulo limite, chamar recursivamente*)
             BUSCA_INTERVALO   (R↑.SUB [ 2 ] );
     end;

end;
```

```
function   ESTA_NA_REGIÃO(R) : boolean;
var    I : integer;
       TESTE : boolean;
begin   I:= 1;
        TESTE : true;
        while   (I≤K)   and   TESTE   do
        begin
           if  (R↑.CHAVE [ I ]  <  REGI[2*I-1])or
               (R↑.CHAVE [ I ]  >  REGI [2*I ])
           then    TESTE:= false;
           I:= I + 1
        end;
           ESTA_NA_REGIÃO:= TESTE
end;
```

5.2.2 Árvore k-d Ótima

O resultado anterior de que a busca exata toma em média 1.386 $\log_2 n$ testes em uma árvore k-d construída aleatoriamente pode não ser satisfatório em algumas aplicações, sabendo-se que é possível melhorar facilmente esse valor médio. Por exemplo, se a árvore é construída uma vez e depois um grande número de buscas vai ser realizado sem que a árvore seja alterada, pode ser conveniente construir uma árvore com certas caracterís

ticas que vão tornar mais rápidos os processos de busca exata e busca por intervalo. Apesar de ser mais custoso, por ter um pré-processamento maior, é possivel construir uma árvore k-d otimizada tal que, todos os nós terminais aparecem em dois níveis adjacentes. Como essa árvore é balanceada, a busca será feita no menor tempo possível.

Este processo de construção baseia-se na escolha do valor médio de uma chave para ser o valor do discriminante de um nó. Todo o conjunto de nós que vai entrar na árvore deve ser inicialmente conhecido e os discriminantes de cada nível são tomados cíclicamente. Logo, o discriminante da raiz será a chave K_1. O nó cujo valor de K_1 é a mediana entre todos os valores de K_1 será a raiz da árvore e o número de nós nas subárvores direita e esquerda da raiz difere de no máximo 1. O discriminante do nível seguinte é a chave K_2 e a idéia de partição pela escolha da mediana é aplicada recursivamente aos dois subconjuntos de nós obtidos pela partição do nível anterior. O mesmo procedimento é adotado nos níveis seguintes, particionando cada subconjunto em outros dois até que todos os nós estejam representados na árvore.

Para descrever este algoritmo pode ser usada, para armazenar os nós que serão inseridos, uma estrutura em forma de lista encadeada onde cada registro tem um campo de ponteiro além dos campos das chaves, como mostra a figura 5.17. A ligação terra ($\perp$) representa ponteiro nulo.

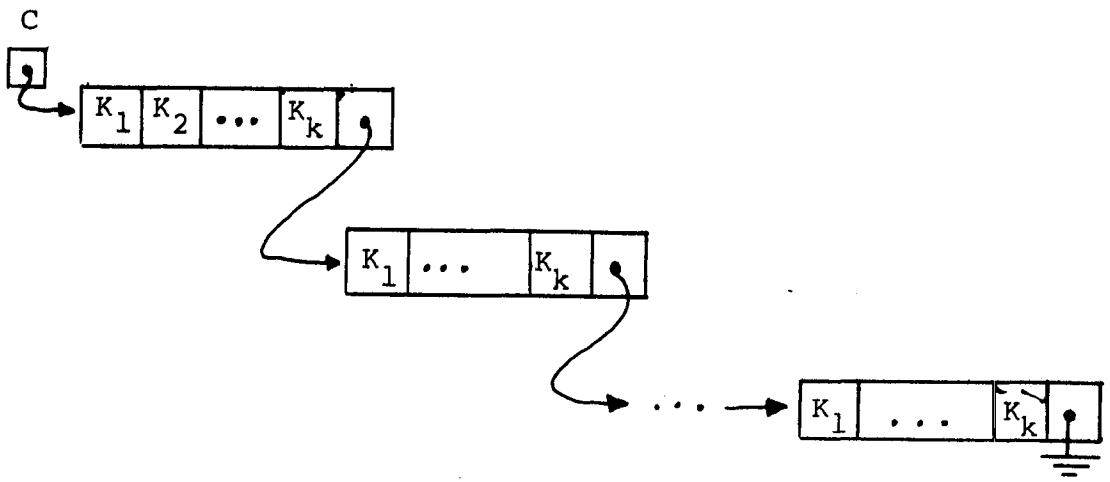


Fig. 5.17

Para o armazenamento da árvore usa-se a mesma estrutura definida para os algoritmos de inserção e busca. Portanto, além das definições de tipos e declarações de variáveis anteriores, supõe-se a existência de :

```
type    PONT  =  ↑LISTA;
        LISTA=   record
                   CHAVE : array [1..K] of real;
                   LIG : PONT
                 end;

var  :  C, CE, CD : PONT;
```

O algoritmo OTIMIZAR que constrói a árvore k-d otimizada utiliza os procedimentos e funções relacionados a seguir onde C é um ponteiro para o início da lista encadeada que contém os nós que serão inseridos na árvore, J é índice de uma chave, I é ponteiro para algum elemento da lista C e R é ponteiro para alguma posição da árvore. Estes procedimentos não estão detalhados:

- ORDENAR (C,J), que faz a ordenação crescente da lista encadeada C pela J-ésima chave
- função J_MÉDIO (C), que encontra o elemento médio da lista C que está ordenada pelos valores da chave de índice J e retorna um ponteiro para esse elemento.
- INSERIR (R,I) que insere o nó que está na posição I da lista encadeada C na posição R da árvore.

A função OTIMIZAR tem os parâmetros: C que é um ponteiro para a lista encadeada e J que é um índice da chave que é o discriminante do próximo nó que vai entrar na árvore. Como neste processo os discriminantes são escolhidos cíclicamente a raiz tem como discriminante a chave K_1, logo o valor de J antes da chamada da função deve ser 1. Se vai ser construída uma árvore cuja raiz é RAIZ, a função será inicialmente ativada por:

```
RAIZ: = OTIMIZAR  (C,J) ;
```

Logo, esse algoritmo pode ser detalhado como segue:

```
function  OTIMIZAR (C: PONT; J: integer): NO;
(*Dada uma coleção de nós C e um discriminante J, retorna um pon
 teiro para uma árvore k-d otimizada cuja raiz é um J- discrimi
 nante*)
var R: NO;
    I: PONT;
    M: integer;
begin
   if C = nil then(*conjunto de nós vazio, árvore vazia*)
      R := nil
   else begin
         (* separar a coleção C em CE e CD*)
         ORDENAR (C,J);
         I: = J_MÉDIO (C);
         if C=I then CE:= nil
         else CE:=C (* início da lista de nós da subárvore es
                       querda*)
         INSERIR(R,I); (*inserir I-ésimo nó da lista no regis
                         tro da árvore apontado por R*)
         M:= (J mod K) + 1; (*discriminante do próximo nível*)
         R↑.SUB [1]: = OTIMIZAR (CE,M);
         CD; = I↑.LIG; (*início da lista de nós da  subárvore
                         direita*)
         R↑.SUB [2]:= OTIMIZAR (CD,M);
      end;
    OTIMIZAR:= R
end;
```

Uma árvore k-d construída com esse algoritmo será uma árvore binária que tem comprimento de caminho interno (CCI) mínimo. Knuth mostra uma relação para árvore 1-d [KNUTH-68] que é válida para árvores k-d otimizadas [BENTLEY-75a]: o comprimento de caminho interno de uma árvore de n nós é:

$$CCI(n) = \sum_{1 \le i \le n} \log_2 i = (n+1)q - 2^{q+1} + 2$$

$$= (n+1)q - 2^{q+1} + 2$$

onde $q = \log_2(n+1)$

Para analisar a quantidade de tempo gasto nesse algoritmo, verifica-se primeiro um único nível de recursão. No j-ésimo nível existem 2^j otimizações sendo realizadas, cada uma para uma subárvore de aproximadamente $n/2^j$. O elemento médio dos $n/2^j$ nós pode ser encontrado em $O(n/2^j)$. A divisão em subcoleções também pode ser realizada em $O(n/2^j)$, portanto o tempo total de processamento em cada nível de recursão é $2^j\ O(n/2^j) = O(n)$. Como a profundidade de recursão é limitada por $\log_2 n$ que é o comprimento de caminho máximo, o tempo total de processamento do algoritmo OTIMIZAR é $O(n\log_2 n)$.

5.3 *Análise de Busca por Intervalo para Árvore k-d Balanceadas*

A análise formal de busca por intervalo para a estrutura de árvores k-d é difícil de ser feita se a árvore for construída randomicamente e se forem considerados todos os tamanhos e formas que a região de busca pode tomar. Por isso, é necessário, para chegar-se a algumas conclusões, considerar um tipo particular desta estrutura, por exemplo, a árvore k-d balanceada na qual todos os nós terminais estão em no máximo dois níveis.

Em [SILVA-79] encontra-se uma análise de busca por intervalo, com a aplicação do algoritmo BUSCA-INTERVALO a árvores balanceadas. Nesta análise é considerada uma árvore com $n = 2^{mk}-1$ nós para algum inteiro m, construída com o algoritmo OTIMIZAR da seção 5.2.2, onde k é o número de chaves do registro. Os registros estão distribuidos uniformemente no hipercubo unitário $[0,1]^k$, e os intervalos especificados na consulta são tais que:

$$a_j \leq K_j \leq a_j+x_j$$

Para determinar o número de nós visitados por BUSCA-INTERVALO é suficiente contar o número de retângulos limites associados aos nós que interceccionam a região de busca especificada pela consulta. O procedimento usado nesta análise consiste em determinar o número de nós visitados a cada nível e somar esse valor para todos os níveis.

É possível verificar que a cada nível i da árvore, qualquer linha paralela ao j-ésimo eixo intercepta $2^{\lceil (i-j+1)/k \rceil}$ retângulos associados aos nós da árvore.

Considere-se inicialmente o caso em que k=2. Por - tanto, as questões de busca por intervalo determinam que:

$$a_1 \leq K_1 \leq a_1 + x_1 \quad \text{e} \quad a_2 \leq K_2 \leq a_2 + x_2$$

e assim, as regiões de busca serão retângulos em $[0,1]^2$ com lados x_1 e x_2.

Para o nível i=0, sendo a raíz da árvore o nó A e seu retângulo limite $[0,1]^2$ como mostra a figura 5.18,

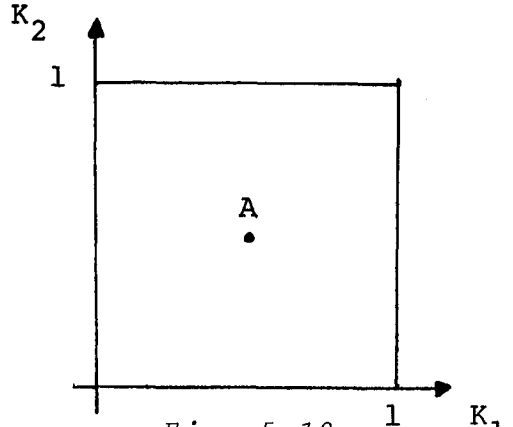


Fig. 5.18

qualquer linha paralela aos eixos K_1 ou K_2 intercepta apenas um retângulo limite, que é o associado ao nó A. Se no nível - i=1 estão os nós B e C, como na figura 5.19, qualquer linha paralela ao eixo K_1 intercepta duas regiões do espaço, ou se ja, as regiões associadas com B e C independentemente de suas localizações. Qualquer linha paralela a K_2 intercepta apenas uma região do espaço, ou a região de B, ou a de C. Logo nesse

nível tem-se duas regiões ao longo de K_1 e uma ao longo de K_2.

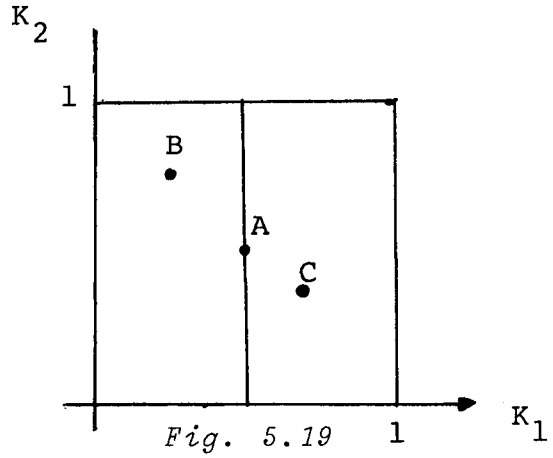


Fig. 5.19

Para o nível i=2, como mostra a figura 5.20, linha paralelas aos eixos K_1 ou K_2 interceptam duas regiões, independentemente da localização dos nós.

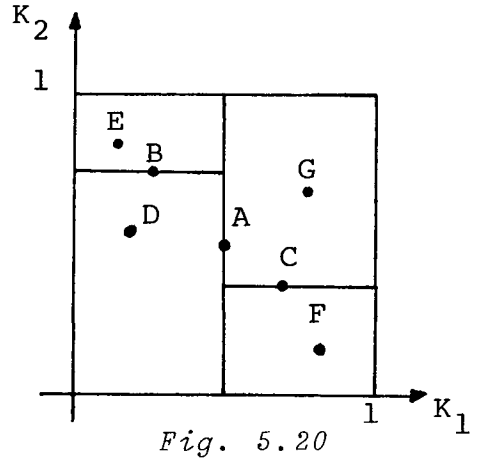


Fig. 5.20

No nível i=3, qualquer linha paralela ao eixo K_1 intercepta quatro regiões e linha paralelas a K_2 interceptam duas regiões, como pode ser visto na figura 5.21. Tem-se portanto quatro regiões na direção K_1 e duas na direção K_2.

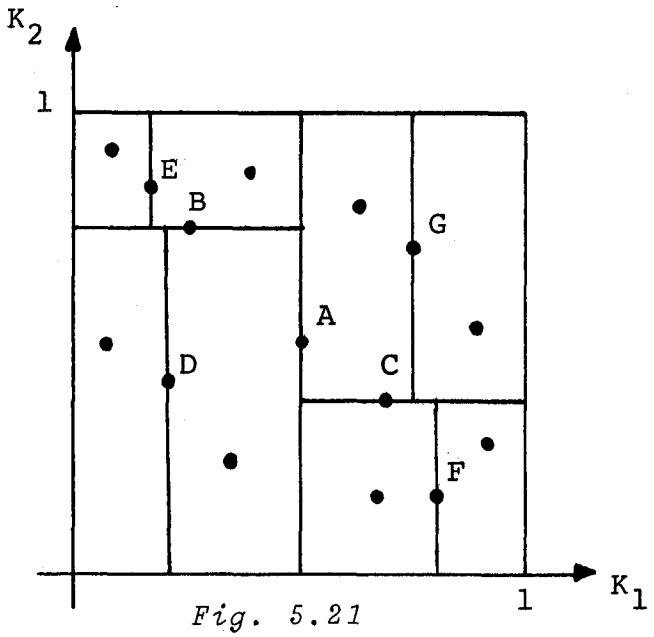


Fig. 5.21

Assim, para k=2, verifica-se que em qualquer nível i as linhas paralelas ao j-ésimo eixo interceptam

$$(2^{\lceil (i-j+1)/2 \rceil})$$

regiões do espaço associadas com nós da árvore, ou seja existem $(2^{\lceil (i-j+1)/2 \rceil})$ regiões ao longo do eixo K_j no nível i.

Assumindo, para valores grandes de n, onde n é o número de nós, que os limites entre retângulos são aproximadamente equidistantes, então o intervalo de busca x_1 intercepta em média,

$$\text{de } (x_1 2^{\lceil (i-1+1)/2 \rceil}) \text{ a } (x_1 2^{\lceil (i-1+1)/2 \rceil}+1)$$

regiões na direção paralela ao eixo K_1 e o intervalo x_2 intercepta em média,

$$\text{de } (x_2 2^{\lceil (i-2+1)/2 \rceil}) \text{ a } (x_2 2^{\lceil (i-2+1)/2 \rceil}+1)$$

regiões na direção paralela ao eixo K_2.

Esses números médios tem como limites superiores -

os valores $\lceil x_1 2^{\lceil (i-1+1)/2 \rceil} + 1 \rceil$. e $\lceil x_2 2^{\lceil (i-2+1)/2 \rceil} + 1 \rceil$

O exemplo da figura 5.22 permite visualisar este resultado para um tipo particular de árvore em que os nós estão no centro de seu retângulo limite e mostra as regiões do espaço associadas com nós da árvore até o nível 3.

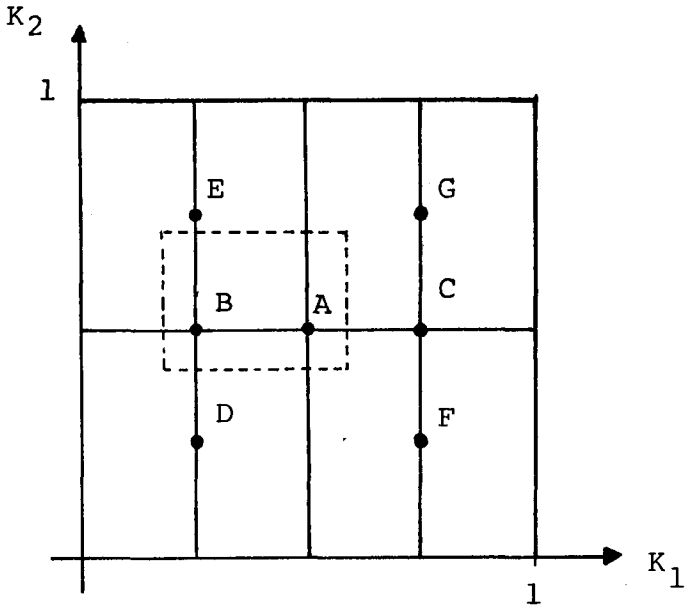


Fig. 5.22

Verifica-se que cada retângulo limite tem lado - x_1=1/4 ao longo do eixo K_1 e x_2=1/2 ao longo do eixo K_2. Se tivermos um retângulo de busca com os lados x_1=0,4 e x_2=0,3 , a figura 5.22 mostra o pior caso possível para sua localiza - ção, com a sobreposição de três retângulos na direção K_1 e de dois na direção K_2, o que é dado por:

$$\lceil x_1 2^{\lceil i/2 \rceil} + 1 \rceil = \lceil (0.4).2^2 + 1 \rceil = 3$$

$$\lceil x_2 2^{\lceil (i-1)/2 \rceil} + 1 \rceil = \lceil (0.3).2 + 1 \rceil = 2$$

Em média o que ocorre é que o intervalo x_1 inter - cecciona nesse nível i=3, entre

$$x_1 2^{\lceil (3/2) \rceil} \quad e \quad x_1 . 2^{\lceil (3-1)/2 \rceil + 1}$$

retângulos na direção K_1 e o intervalo x_2 intercecciona entre

$$x_2 . 2^{\lceil (3/2) \rceil} \quad e \quad x_2 . 2^{\lceil (3-1)/2 \rceil} + 1$$

retângulos na direção K_2.

Logo, para hiperplanos equidistantes tem-se que:

$$(x_j 2^{\lceil (i-j+1)/2 \rceil + 1}) \quad \text{para} \quad j=1,2$$

é o limite superior do número esperado de intersecções ao longo dos eixos K_j e uma boa aproximação, a menos que i esteja próximo de 0 e x_j próximo de 1.

Assim, o número esperado de retângulos associados com nós no nível i que interceccionam a região de busca para $k=2$ é dado por:

$$V(i,x_1,x_2) = (x_1 . 2^{\lceil i/2 \rceil + 1}) \ (x_2 . 2^{\lceil (i-1)/2 \rceil + 1})$$

e o número total de retângulos que se sobrepõem à região de busca é encontrado somando-se esse valor para todos os níveis da árvore, ou seja:

$$E(n,x_1,x_2) = \sum_{0 \leq i \leq h-1} V(i,x_1,x_2)$$

$$= \sum_{0 \leq i \leq h-1} (x_1 . 2^{\lceil i/2 \rceil + 1} \ (x_2 2^{\lceil (i-1)/2 \rceil + 1})$$

Pela hipótese inicial,

$$n=2^{mk}-1$$

Logo, sendo h a altura da árvore, para k=2, tem-se:

$$h=2m$$

Então,

$$E(n,x_1,x_2) = \sum_{0\leq i\leq m-1} (x_1 2^i+1)\ (x_2 2^i+1) + \sum_{0\leq i\leq m-1} (x_1 2^{i+1}+1)\ (x_2 2^i+1)$$

$$= \sum_{0\leq i\leq m-1} [x_1x_2 2^{2i}+(x_1+x_2)\,2^i+1+x_1x_2 2^{2i+1}+x_1 2^{i+1}+x_2 2^i+1] \qquad [5.5]$$

Sabendo-se que:

$$\sum_{0\leq i\leq m-1} 2^{2i} = \frac{2^{2m}-1}{3} \qquad [5.6]$$

$$\sum_{0\leq i\leq m-1} 2^{i} = 2^m-1 \qquad [5.7]$$

$$\sum_{0\leq i\leq m-1} 2^{2i+1} = 2\left(\frac{2^{2m}-1}{3}\right) \qquad [5.8]$$

e substituindo-se [5.6], [5.7], [5.8] em [5.5] obtem-se

$$E(n,x_1,x_2) = x_1x_2 \cdot \frac{2^{2m}-1}{3} + (x_1+2x_2)\ (2^m-1) +$$

$$x_1x_2 \cdot \frac{2}{3}(2^{2m}-1) + 2x_1\ (2^m-1) + 2m$$

$$= x_1x_2(2^{2m}-1) + (3x_1+2x_2)\ (2^m-1) + 2m \qquad [5.9]$$

Como

$$n=2^{2m}-1 \qquad [5.10]$$

vem:

$$2^m = (n+1)^{1/2} \qquad [5.11]$$

e

$$2m = \log_2 (n+1) \qquad [5.12]$$

Substituindo-se [5.10], [5.11] e [5.12] em [5.9], resulta a expressão em termos de n:

$$E(n,x_1,x_2) = x_1x_2n + (3x_1+2x_2)\ ((n+1)^{1/2}-1) + \log_2(n+1)$$

Em geral, para qualquer valor de k, sendo a região de busca dada por

$$a_j \leq K_j \leq a_j+x_j \qquad \text{para } j=1,\ldots,k$$

e considerando-se uma árvore k-d com registros uniformemente distribuidos no hipercubo unitário $[0,1]^k$, a cada nível i qualquer linha paralela ao eixo K_j intercepta

$$2^{\lceil (i-j+1)/k \rceil}$$

regiões associadas com os nós da árvore. Para hiperplanos exatamente equidistantes, o valor aproximado

$$(x_j . 2^{\lceil (i-j+1)/k \rceil}+1)$$

é um limite superior do número esperado de intercecções da região de busca com retângulos limites ao longo do eixo K_j e uma boa aproximação, a menos que i esteja próximo de 0 e x_j - próximo de 1. O número esperado de regiões associadas aos nós

do nível i de uma árvore k-dimensional, que interseccionam a região de busca é dado por:

$$V(i,x_1,x_2,\ldots,x_k) = \prod_{1\leq j\leq k} (x_j.2^{\lceil (i-j+1)/k \rceil + 1})$$

e esse valor, somado sobre todos os níveis da árvore dá uma aproximação do número esperado de nós visitados por BUSCA-INTERVALO:

$$E(h,x_1,\ldots x_k) = \sum_{0\leq i\leq h-1} \prod_{1\leq j\leq k} (x_j 2^{\lceil (i-j+1)/k \rceil + 1})$$

Fazendo o desenvolvimento que é análogo ao de árvores Quad obtem-se:

$$E(n,x_1,x_2,\ldots,x_k) = n. \prod_{1\leq j\leq k} x_j + O(n^{1-(1/k)})$$

É possível verificar que está expressão, dita avaliadora é adequada, comparando seus valores com os resultados empíricos, obtidos por Silva Filho em [SILVA-79] como mostra a tabela 5a. Para esses testes foram feitas simulações para k = 2 e intervalos especificados por um valor fixo x para ambas as chaves. Os registros tem valores em $[0,1]^2$ e a expressão utilizada é:

$$E(n,x) = nx^2+5x((n+1)^{1/2}-1)+\log_2(n+1)$$

Além do número de nós visitados dado pelos testes empíricos e pela expressão E, a tabela 5a mostra os valores de "overwork" realizado, obtidos também pelos testes e pela expressão avaliadora. Como o número esperado de nós na árvore que satisfa-

Tabela 5a

Comparação de valores empíricos e calculados por E(n,x) para o número médio de nós visitados por BUSCA-INTERVALO, com registros em [0,1] x [0,1] . Cada árvore tem n nós e a região de busca é um quadrado de lado x.

número de nós(n)	tamanho de Lado(x)	NÓS VISITADOS		OVERWORK	
		Empírico	Avaliado por E(n,x)	Empírico	Avaliado por E(n,x)
511	0.03125	12.47	12.88	12.09	12.38
	0.065	16.47	17.76	14.59	15.76
	0.125	28.47	30.52	20.51	22.52
	0.25	65.43	68.04	33.20	36.04
	0.5	180.49	191.07	55.73	63.07
1.023	0.03125	14.55	15.84	13.73	14.84
	0.0625	21.25	23.69	17.40	19.69
	0.125	42.82	45.38	26.70	29.38
	0.25	112.48	112.75	45.84	48.75
	0.5	337.70	343.50	82.57	87.50
2.047	0.03125	19.81	19.91	17.81	17.91
	0.0625	31.64	32.83	23.97	24.83
	0.125	66.32	70.66	34.85	34.66
	0.25	187.92	194.31	60.58	66.31
	0.5	631.78	633.64	118.70	121.64
4.095	0.03125	25.17	25.84	21.15	21.84
	0.0625	46.28	47.69	30.34	31.69
	0.125	112.65	115.38	49.28	51.38
	0.25	344.17	346.75	87.69	90.75
	0.5	1181.27	1193.50	163.47	169.50

- zem a consulta é $n\prod_{1\leq j\leq k} x_j$, para registros com valores em $[\phi,1]^k$ o "overwork" esperado da busca é:

$$O(n^{1-(1/k)})$$

Isso significa que o overwork no pior caso e o overwork no caso médio para árvores k-d completamente balanceadas tem a mesma taxa de crescimento assintótica. A diferenca entre as duas é a constante, que para o caso médio pode ser bem pequena.

A análise do caso médio para árvores k-d aleatórias ainda é um problema em aberto.

5.4 *Análise do Pior Caso de Busca por Intervalo para Árvores k-d Balanceadas*

A análise do pior caso de tempo de processamento do algoritmo BUSCA-INTERVALO para árvores balanceadas, é apresentada em [LEE-78]. Para ilustrar o processo utilizado, será discutido primeiramente o caso de duas dimensões. Sem perda de generalidade, assume-se que todos os registros estaõ no primeiro quadrante. O número total de registros é $n=2^h-1$ para algum inteiro positivo h. Assim sendo, essa árvore pode ser construída como uma árvore binária balanceada com todas as folhas aparecendo no nível h-1. A região de busca é equivalente a um retângulo com limites L(1), U(1) para a chave 1 e L(2), U(2) para a chave 2. Sem afetar o resultado da análise assume-se que a região tem dois eixos de coordenadas como limites inferiores, isto é L(1)=L(2)=0.

Com isso a região terá sempre a localização mostrada na figura 5.23.

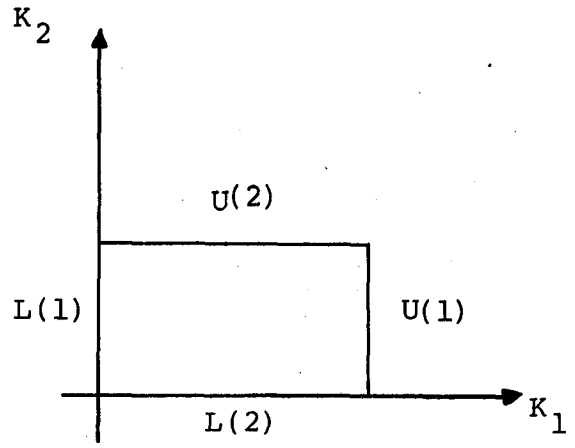


Fig. 5.23

A análise será feita com base no número de nós visitados durante a busca, com uma ressalva importante: no algoritmo original descrito por [BENTLEY-75a], um nó é visitado sempre que seu retângulo limite intercecciona a região de busca; nesta análise porém, quando um retângulo limite está totalmente dentro da região de busca, não serão contadas as visitas feitas ao nó correspondente a esse retângulo e a todos os seus descendentes. Neste caso particular, o retângulo do nó sobrepõe-se à região mas considera-se que ele foi eliminado da busca pois todo o subarquivo nele contido, faz parte da resposta à consulta e pode ser recuperado como um todo.

Como estamos considerando o pior caso, dada uma região de busca podemos dispor os nós de tal forma que seja feito o número máximo de visitas a cada nível. Assim, sendo A a raíz de uma árvore, o pior caso possível de acontecer no início da busca é que as duas subárvores de A devam ser visita -

-das, ou seja, a região de busca tem a localização mostrada - na figura 5.24.

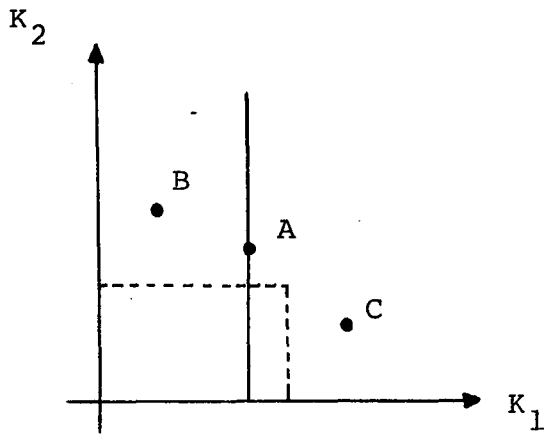


Fig. 5.24

Sendo B e C os filhos à esquerda e à direita de A , respectivamente, neste ponto já se sabe que foram feitas visitas a A,B e C.

No próximo nível, no pior caso um dos subarquivos - representados por D e E, filhos de B respectivamente à es - querda e à direita, está totalmente fora ou totalmente dentro da região de busca. Portanto ou D ou E será eliminado da contagem.Se os nós tiverem as posições mostradas na figura 5.25, a subárvore direita de B está totalmente fora da região, e será eliminada.

Se B estivesse na posição mostrada na figura 5.26 , a subarvore esquerda de B estaria totalmente dentro de região de busca e seria eliminada.

Logo,nesse nível, um dos subarquivos não será mais considerado na contagem,por estar completamente dentro ou com

-pletamente fora da região de busca. Entre os nós D, E, F e G filhos de B e C, pelo menos um pode ser eliminado, independentemente da localização dos nós.

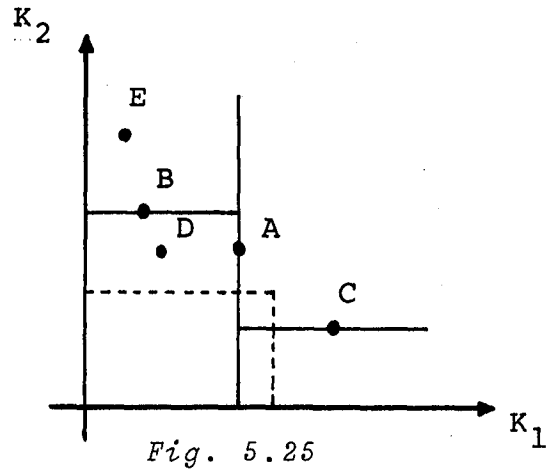


Fig. 5.25

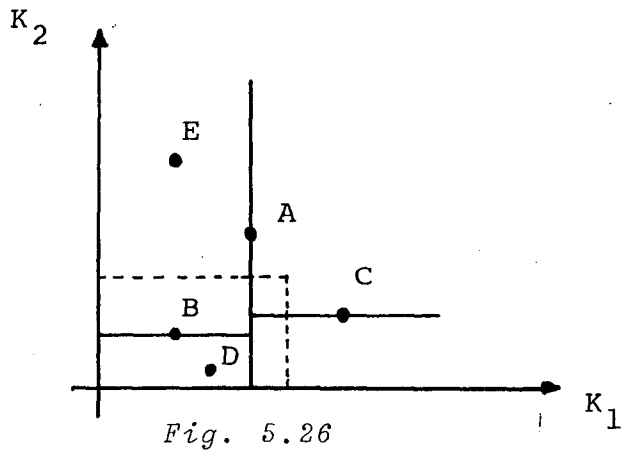


Fig. 5.26

O mesmo que aconteceu com o nó B acontece com F se ele for o filho a esquerda de C: um de seus filhos pode ser eliminado. Na configuração do exemplo considerado na figura

5.27 a subárvore direita de F está totalmente fora da região de busca e vai ser eliminada da contagem.

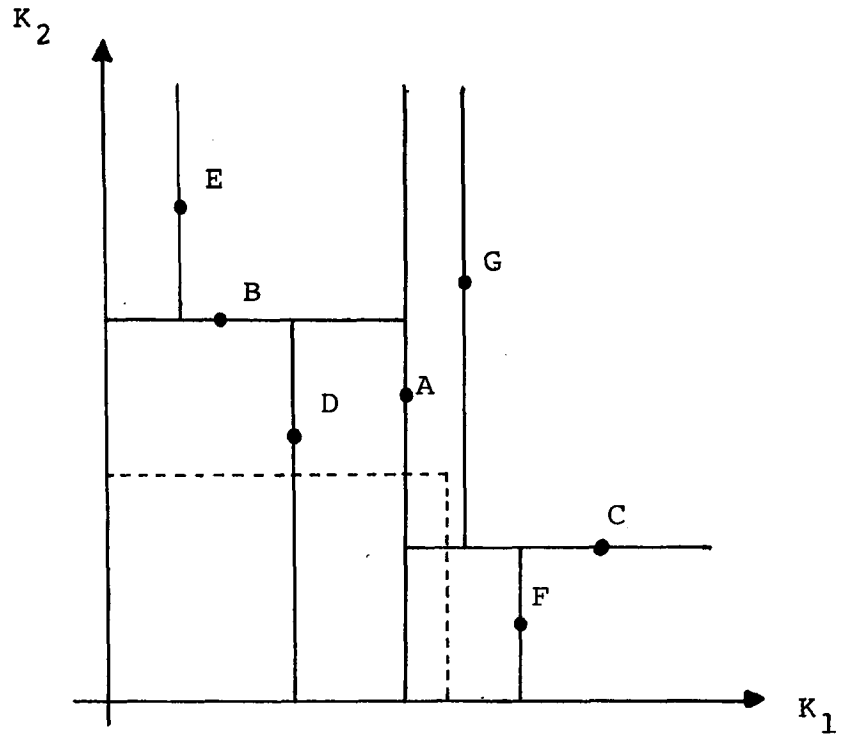


Fig. 5.27

Se F tem uma posição diferente, como na figura 5.28 sua subárvore esquerda está totalmente dentro da região e será eliminada da contagem.

Se D for a subárvore de B a ser visitada, suas duas subárvores serão também pesquisadas, o que pode ser verificado na figura 5.28. Chamando-se de H e I as subárvores de D, - no nível seguinte apenas uma subárvore de H e uma de I, serão visitadas pois em qualquer uma das situações da figura 5.29 vê-se que pelo menos uma subárvore de H e uma de I não serão mais consideradas na contagem.

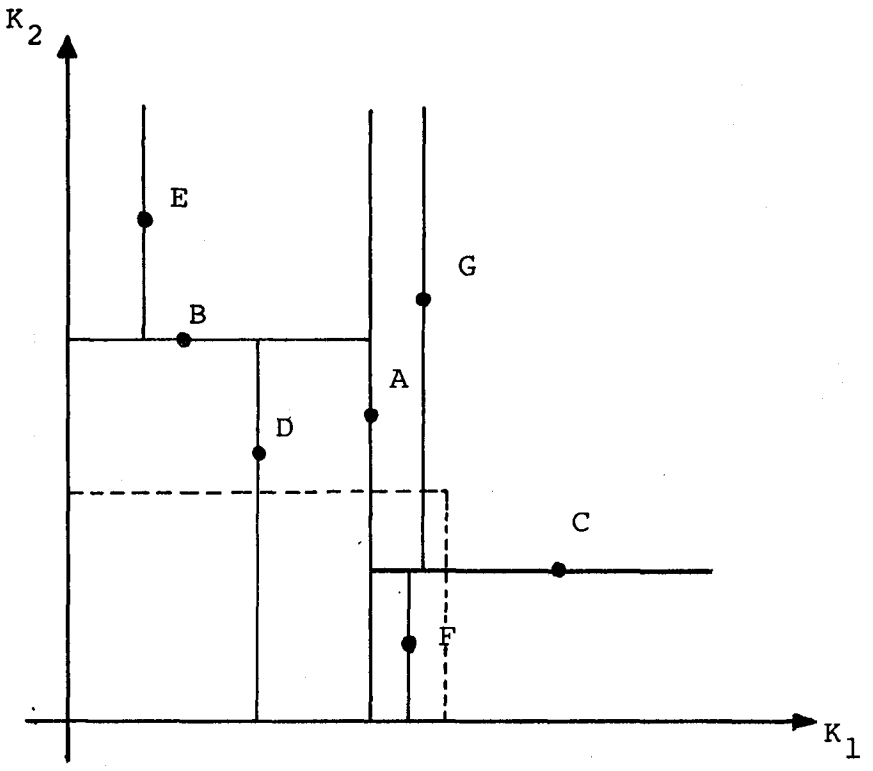


Fig. 5.28

O mesmo acontece com a subárvore de F a ser visitada. Sendo J essa subárvore e L e M filhos de J, serão visitados os nós L e M e no nível seguinte apenas um dos filhos de L e um de M. Portanto, concluímos que, no pior caso, acontece o mesmo com os subarquivos representados por B e F exceto que F está um nível abaixo.

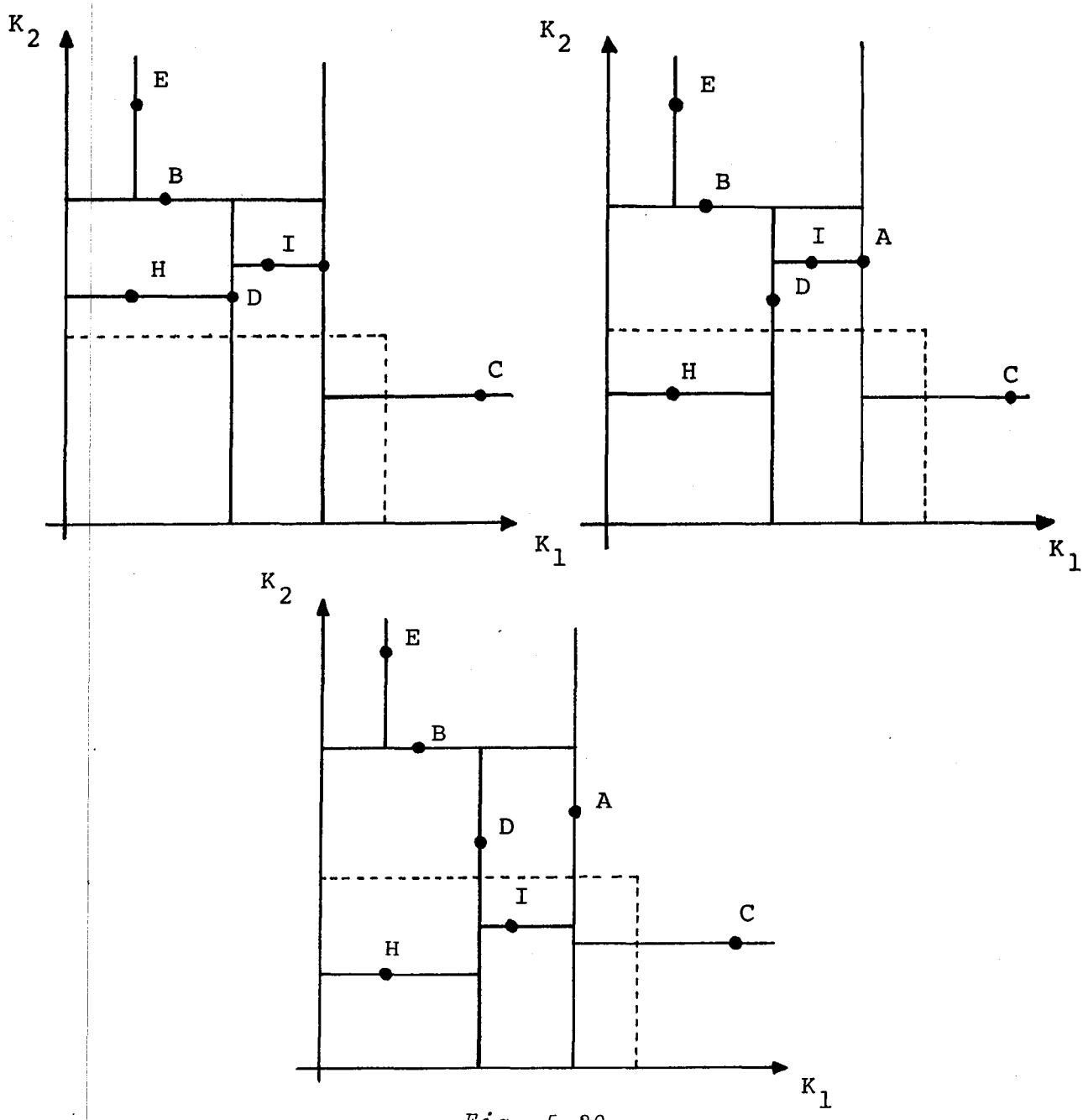


Fig. 5.29

Com relação ao arquivo representado por G, se for feito um movimento de coordenadas para K_1' e K_2' tal que K_k passe por exemplo, por C e K_2' por A, se a posição dos nós for a da figura 5.30, ele será tratado da mesma forma que o arquivo A, com a diferença de que está dois níveis abaixo.

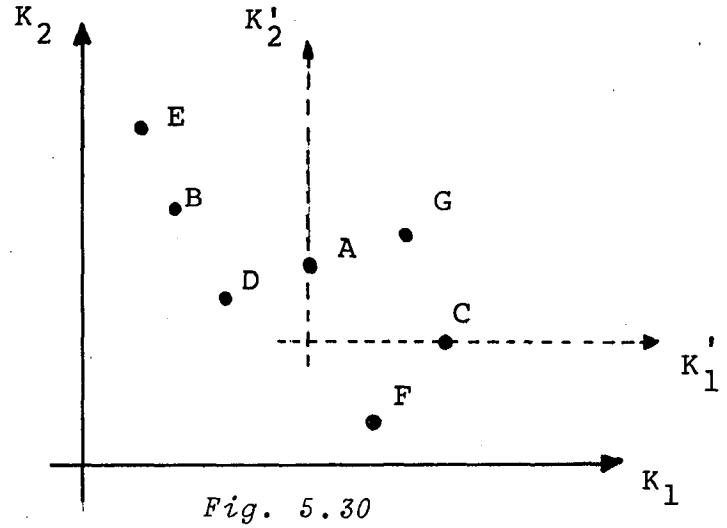


Fig. 5.30

Com estas análises, desenvolvidas para nós até o nível 3, sabe-se o que acontece com todos os nós da árvore. Para exemplificar o que foi dito até agora, considere-se a figura 5.31 que mostra uma região de busca e uma parte de um conjunto de nós dispostos de tal forma que será feito o número máximo de visitas.

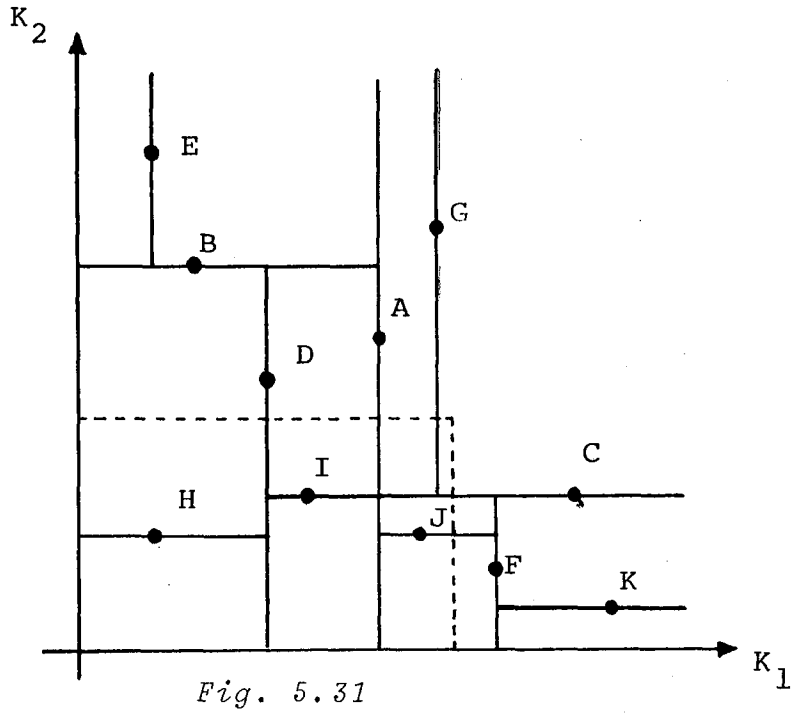


Fig. 5.31

A figura 5.32 mostra a situação da árvore equivalente a essa parte do arquivo da figura 5.31, depois de aplicada a busca. Nós hachurados representam nós que foram eliminados da contagem, por estarem completamente dentro ou completamente fora da região de busca.

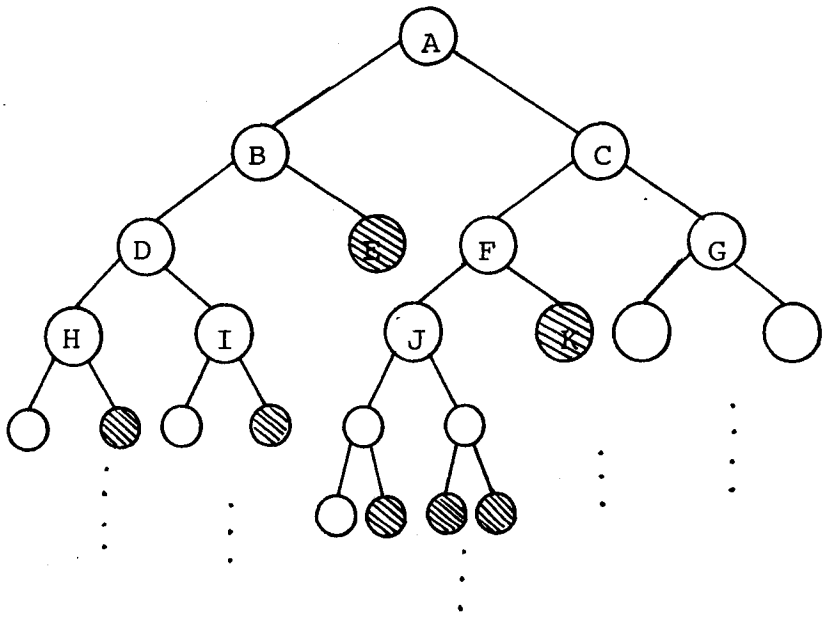


Fig. 5.32

Agora é possível construir as equações de recorrência do número total de nós visitados, com os resultados a que chegamos analisando o que acontece com os nós dos primeiros níveis da árvore.

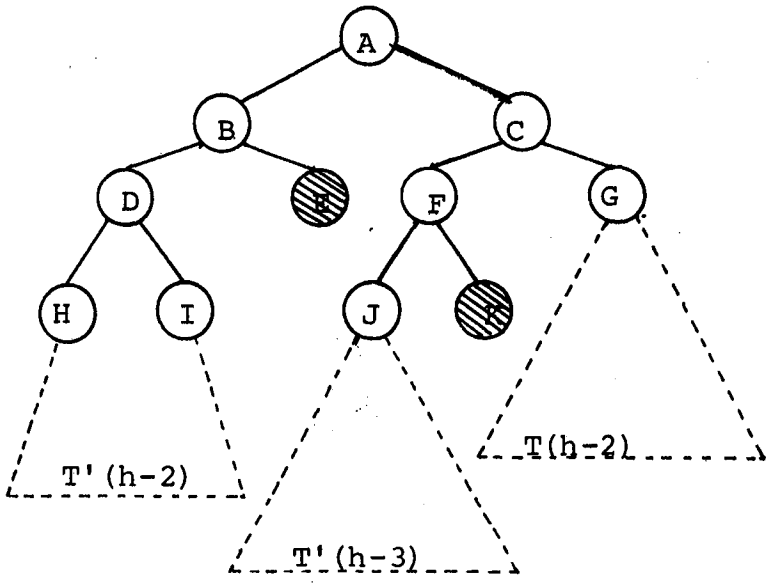
Assim, sendo h a altura da árvore e T(h) o número de nós visitados; sendo h_1 a altura da subárvore D ou E e $T'(h_1)$ o equivalente a T(h) para esta subárvore, as equações de recorrência são:

$T(h) = 1 + 1 + 1 + T(h-2) + T'(h-2) + 1 + T'(h-3)$

$T'(h_1) = 1 + 1 + 1 + 2T'(h_1-2)$

Na figura 5.33 pode-se visualizar essas igualdades para o exemplo da figura 5.31. Nós hachurados representam nós eliminados da contagem.

T(h) - número de nós visitados em toda a árvore.



$T'(h_1)$ - número de nós visitados na subárvore com raiz D.

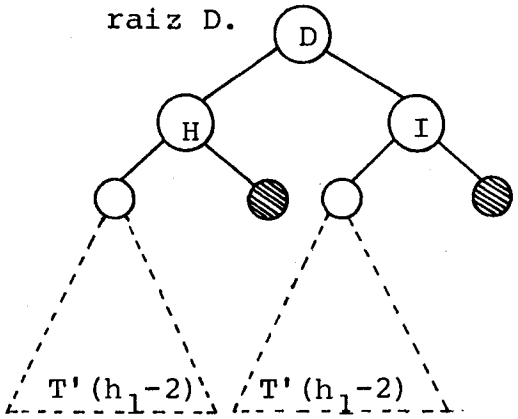


Fig. 5.33

As equações:

$$T(h) = 4+T(h-2)+T'(h-2)+T'(h-3)$$

$$T'(h_1) = 3+2T'(h_1-2)$$

serão resolvidas com as condições iniciais,

$$T'(0) = T(0) = 0$$

$$T'(1) = T(1) = 1$$

Resolvendo-se $T'(h_1)$, vem:

$$T'(h_1) = \begin{cases} 3 \sum\limits_{0 \leq k \leq i-1} 2^k + 2^i & \text{para } h_1 = 2i+1 \quad [5.13] \\ 3 \sum\limits_{0 \leq k \leq i-1} 2^k & \text{para } h_1 = 2i \quad [5.14] \end{cases}$$

efetuando as somatórias [5.13] e [5.14] tem-se:

$$T'(h_1) = \begin{cases} 4.2^i-3 & \text{para } h_1 = 2i+1 \quad [5.15] \\ 3.2^i-3 & \text{para } h_1 = 2i \quad [5.16] \end{cases}$$

substituindo [5.15] e [5.16] em T(h) obtem-se:

$$T(h) = \begin{cases} 4+T(h-2)+10.2^{i-2}-6 & \text{para } h = 2i \\ 4+T(h-2)+7.2^{i-1}-6 & \text{para } h = 2i+1 \end{cases}$$

Resolvendo-se agora para T(h), vem:

$$T(h) = \begin{cases} \sum\limits_{0 \leq k \leq i-1} (-2+5.2^k) & \text{para } h = 2i \quad [5.17] \\ \sum\limits_{0 \leq k \leq i-1} (-2+7.2^k) & \text{para } h = 2i+1 \quad [5.18] \end{cases}$$

efetuando as somatórias [5.17] e [5.18], resulta:

$$T(h) = \begin{cases} 5(2^i-1)-2i & \text{para } h=2i \\ 7(2^i-1)-(2i+1)+2 & \text{para } h=2i+1 \end{cases}$$

então:

$$T(h) = \begin{cases} 5(2^{h/2}-1)-h & \text{para } h=2i \\ 7(2^{\lfloor h/2 \rfloor}-1)-h+2 & \text{para } h=2i+1 \end{cases}$$

Como

$$n = 2^h-1$$

pode-se reescrever T(h) da seguinte forma:

$$T(h) = O(2.n^{1/2})$$

isto é,

$$T(h) = O(k.n^{1-(1/k)}) \qquad \text{para } k = 2$$

Para analisar o caso k-dimensional, a idéia básica é contar o número de regiões "limitadas", pois sempre que uma região limitada é encontrada, pode-se ignorar uma das subárvores na contagem.

Por exemplo, para k=2, no conjunto de nós considerado na figura 5.34, depois de visitar B tem-se uma região limitada associada a D, pois cada um de seus quatro lados tem como limite os próprios eixos ou as linhas paralelas aos eixos que particionam o conjunto de nós.

Se essa região se sobrepõe, mas não está totalmente dentro da região de busca, o subarquivo associado com E pode ser eliminado. Se ela está completamente dentro da região de

busca o subarquivo representado por D não será mais considerado na contagem. Assim, para cada região limitada que é gerada, uma subárvore é eliminada da contagem. Ao se contar o número de regiões limitadas que se sobrepõem à região de busca pode-se conhecer o número total de nós visitados, que será igual ao número de regiões não limitadas. Conta-se primeiro o número de nós visitados em um nível da árvore e depois soma-se esse valor para todos os níveis.

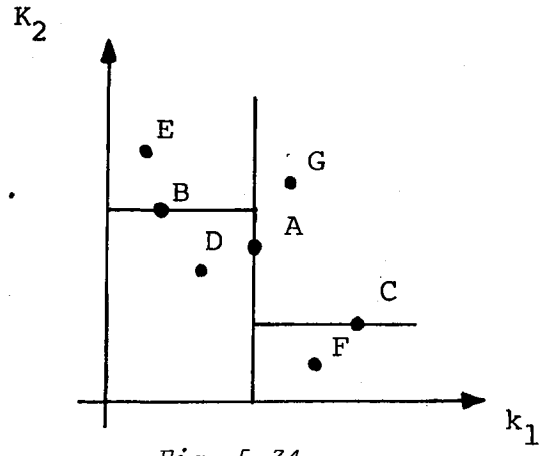


Fig. 5.34

Para ilustrar o raciocínio desenvolvido na análise considere-se ainda o caso bi-dimensional, onde se usa linhas perpendiculares aos eixos K_i para particionar as regiões existentes. Essas linhas também servem como limites de novas regiões. No nível zero, tem-se apenas uma região não limitada. No nível 1 é traçada a primeira linha perpendicular a K_1, o que é chamado de primeira "visita" ao eixo K_1, e obtem-se com isso duas regiões não limitadas. No nível 2 são traçadas li-

nhas perpendiculares a K_2 o que é primeira visita a K_2 e faz surgir a primeira região limitada (figura 5.35).

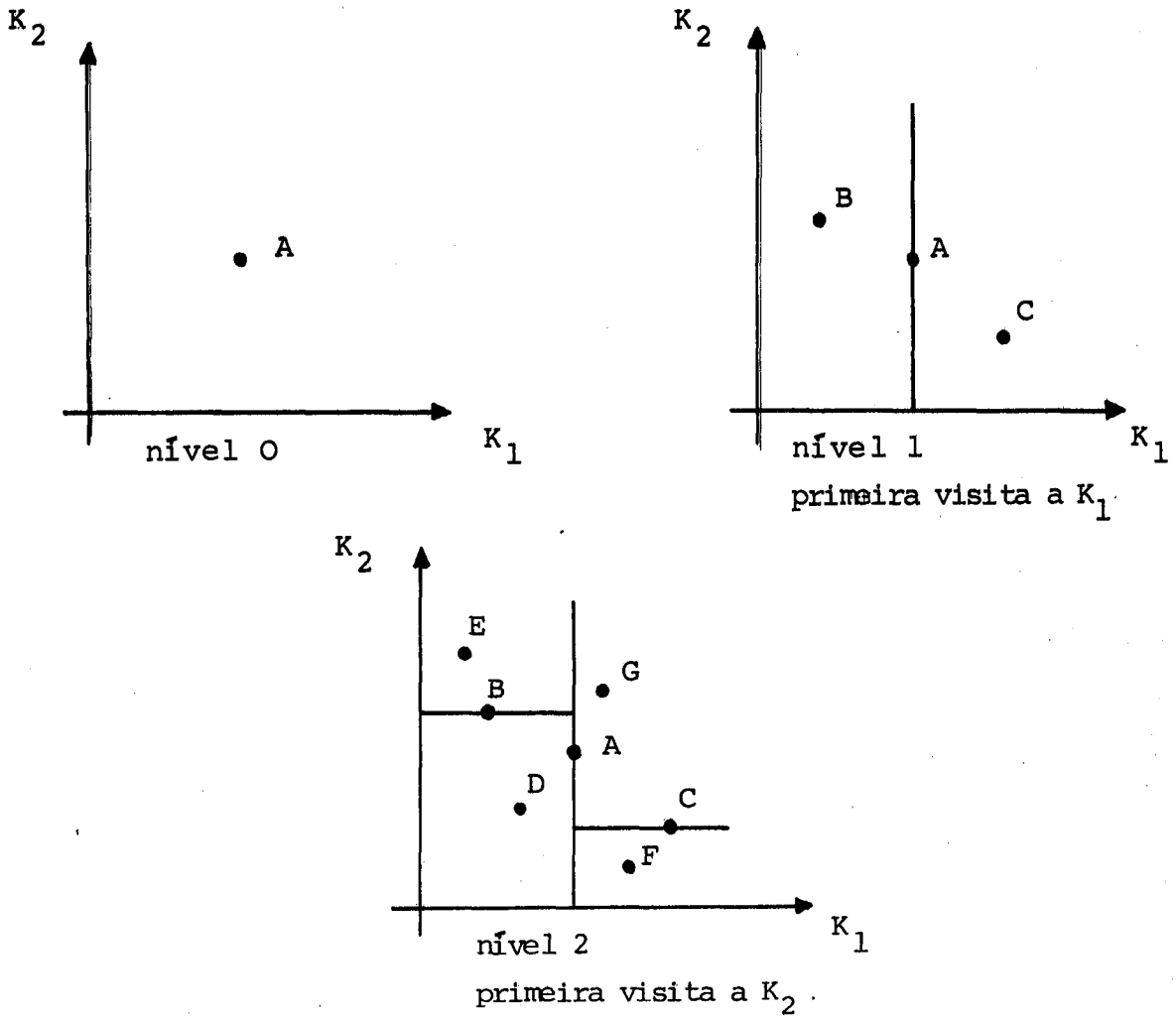


Fig. 5.35

Na segunda visita a K_1, são traçadas duas linhas perpendiculares ao eixo K_1, em cada uma das regiões geradas na primeira visita. Isto resulta quatro regiões ao longo de K_1, isto é, regiões interceptadas por linhas paralelas a K_1 ,

das quais apenas uma não é limitada na direção de K_1 (figura 5.36).

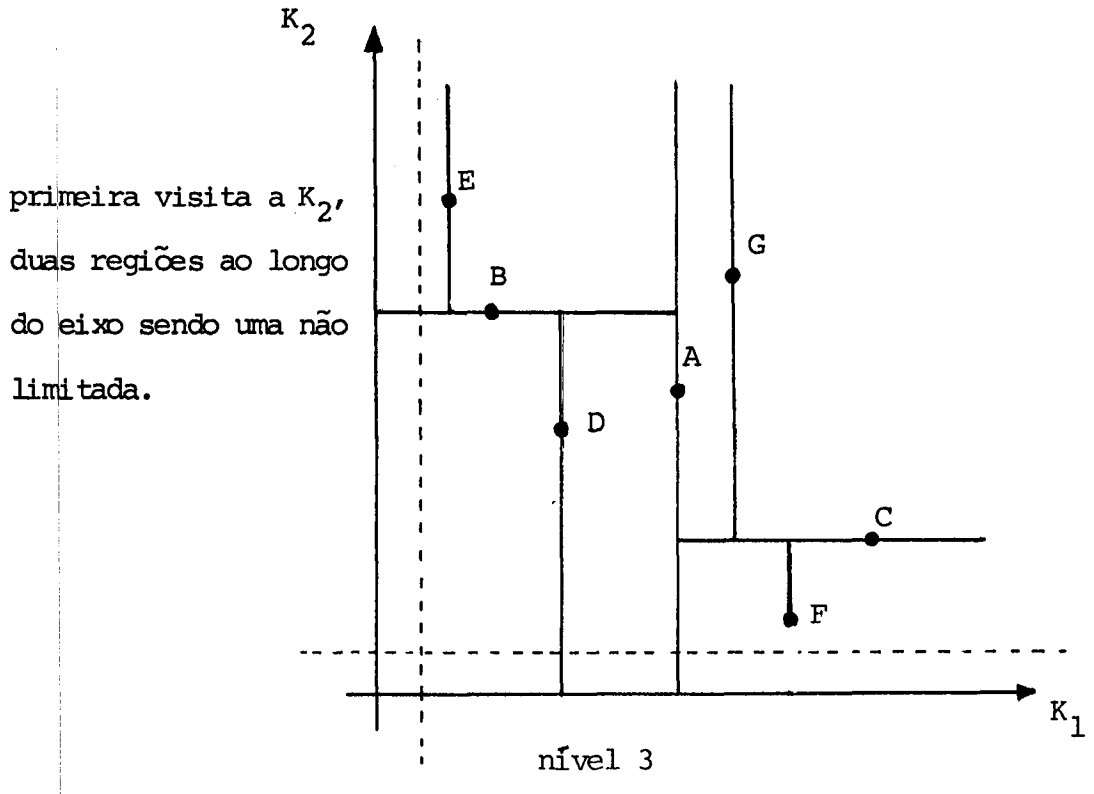


Fig. 5.36

No próximo nível faz-se a segunda visita a K_2, e assim por diante, de tal forma que, no nível i, para k=2, o número de visitas feitas ao j-ésimo eixo é:

$$1 + \left\lfloor \frac{i-j}{2} \right\rfloor$$

e o número de regiões geradas por essas visitas, é:

$$2^{1+\lfloor (i-j)/2 \rfloor}$$

Todas exceto uma são limitadas por duas linhas perpendiculares ao eixo K_j, e assim pode-se calcular o número total de regiões limitadas no nível i, que é:

$$(2^{1+\lfloor (i-1)/2 \rfloor} - 1)\ (2^{1+\lfloor (i-2)/2 \rfloor} - 1)$$

Com isso pode-se calcular o número total de regiões não limitadas em cada nível, que é o número de nós visitados nesse nível. Depois, somando-se esse número para todos os níveis, têm-se o número total de nós visitados, considerada a ressalva inicial.

Para o caso geral de registros com k chaves, assume-se que o número total de nós é $n = 2^{mk}-1$ e que as chaves tem valores positivos. Os discriminantes também são escolhidos ciclicamente e hiperplanos perpendiculares aos eixos K_1, K_2, ...,K_k particionam e limitam regiões.

No nível zero tem-se apenas uma região não limitada. No nível 1 tem-se um hiperplano perpendicular a K_1, o que é a primeira visita a K_1. Faz-se o mesmo para $K_2,\ldots,K_k$. Na segunda visita a K_1 tem-se 2 hiperplanos perpendiculares a K_1, um em cada uma das duas regiões geradas na primeira visita, e o número resultante de regiões ao longo de K_1 é quatro. Faz-se o mesmo para $K_2,\ldots,K_k$ e assim por diante. No nível $i < k$ as regiões geradas são não limitadas, e no nível $i = k$ a primeira região limitada aparece. Em geral, no nível i o número de visitas ao eixo K_i é:

$$1 + \left\lfloor \frac{i-j}{K} \right\rfloor$$

o que é possível verificar considerando que, até o nível k fez-se uma visita a cada eixo, até o nível $2k$ fez-se duas visitas, e até o nível $\ell_1 k$ fez-se ℓ_1 visitas a cada eixo. No nível $\ell_1 K+\ell_2$, para ℓ_1 e ℓ_2 inteiros não negativos, se $j \leq \ell_2$ fez-se ℓ_1+1 visitas ao j-ésimo eixo e se $j > \ell_2$ fez-se ℓ_1 visitas ao j-ésimo eixo. Isso é dado por:

$$1 + \left\lfloor \frac{i-j}{k} \right\rfloor$$

O número de regiões geradas por essas visitas ao longo do eixo K_j é:

$$2^{1+\lfloor (i-j)/k \rfloor}$$

e todas elas, exceto uma, estão limitadas na direção K_j por dois hiperplanos perpendiculares ao eixo K_j. Assim o número total de regiões limitadas no nível i é dado por:

$$B(i) = \prod_{1 \leq j \leq k} (2^{1+\lfloor (i-j)/k \rfloor} - 1)$$

e o número total de regiões não limitadas no nível i é:

$$S(i) = 2^i - B(i)$$

Isso significa que, de acôrdo com o raciocínio empregado nesta análise, o número total de nós visitados é a soma desse valor para todos os níveis da árvore, ou seja:

$$T(h) = \sum_{0 \leq i \leq h-1} S(i)$$

$$= \sum_{0 \leq i \leq h-1} \left[2^i - \prod_{1 \leq j \leq k} (2^{1+\lfloor (i-j)/k \rfloor} - 1) \right] \quad [5.19]$$

como considerou-se que:

$$n = 2^{mk} - 1$$

tem-se h=mk

Assim,

$$\sum_{0 \leq i \leq h-1} 2^i = 2^{mk} - 1 \qquad [5.20]$$

Considerando que:

$$A = \prod_{1 \leq j \leq k} (2^{1+\lfloor (i-j)/k \rfloor} - 1)$$

vem:

$$A = (2^{1+\lfloor (i-1)/k \rfloor} - 1)\,(2^{1+\lfloor (i-2)/k \rfloor} - 1)\,(2^{1+\lfloor (i-3)/k \rfloor} - 1) \ldots (2^{1+\lfloor (i-k)/k \rfloor} - 1)$$

ou seja,

$$\sum_{0 \leq i \leq h-1} A = \sum_{0 \leq i \leq k-1} A + \sum_{k \leq i \leq 2k-1} A + \sum_{2k \leq i \leq 3k-1} A + \ldots + \sum_{(m-1)k \leq i \leq mk-1} A$$

Desenvolvendo as somatórias verifica-se que:

$$\sum_{0 \leq i \leq h-1} \prod_{1 \leq j \leq k} (2^{1+\lfloor (i-k)/k \rfloor} - 1) =$$

$$0 + (2^2-1)^0 (2^1-1)^k + (2^2-1)^1 (2^1-1)^{k-1} + \ldots\ldots + (2^2-1)^{k-1} (2-1)^1$$

$$+ (2^3-1)^0 (2^2-1)^k + (2^3-1)^1 (2^2-1)^{k-1} + \ldots\ldots + (2^3-1)^{k-1} (2^2-1)^1$$

$$+ \ldots\ldots\ldots\ldots\ldots\ldots$$

$$+ (2^m-1)^0 (2^{m-1}-1)^k + (2^m-1)^1 (2^{m-1}-1)^{k-1} + \ldots + (2^m-1)^{k-1} (2^{m-1}-1)^1$$

$$= \sum_{1 \leq i \leq m-1} (2^i-1)^k + \sum_{1 \leq t \leq m-1} \sum_{1 \leq i \leq k-1} (2^t-1)^i (2^{t+1}-1)^{k-i} \qquad [5.21]$$

chamando-se de G o somatório:

$$G = \sum_{1 \leq i \leq k-1} (2^t-1)^i (2^{t+1}-1)^{k-i}$$

e desenvolvendo, vem:

$$G = \sum_{1 \leq i \leq k-1} (2^t-1)^i (2^{t+1}-1)^{k-i} \frac{(2^{t+1}-1)^i}{(2^{t+1}-1)^i}$$

$$= (2^{t+1}-1)^k \sum_{1 \leq i \leq k-1} \frac{(2^t-1)^i}{(2^{t+1}-1)^i} \qquad [5.22]$$

Resolvendo-se na equação [5.22] o somatório, resulta:

$$\sum_{1 \leq i \leq k-1} \left(\frac{2^t-1}{2^{t+1}-1}\right)^i = \frac{((2^t-1)/(2^{t+1}-1))^k - 1}{((2^t-1)/(2^{t+1}-1))-1} - 1$$

$$= \frac{(2^t-1)^k(2^{t+1}-1) - (2^{t+1}-1)^k(2^t-1)}{(-2^t)(2^{t+1}-1)^k} \qquad [5.23]$$

Substituindo-se [5.23] em [5.22] e simplificando, obtem-se:

$$\sum_{1 \leq j \leq k-1} (2^t-1)^i (2^{t+1}-1)^{k-i} = \frac{(2^{t+1}-1)^k(2^t-1) - (2^t-1)^k(2^{t+1}-1)}{2^t} \qquad [5.24]$$

Substituindo-se [5.24] em [5.21] obtem-se:

$$\sum_{0 \leq i \leq h-1} \prod_{1 \leq j \leq k} (2^{1+\lfloor (i-j)/k \rfloor}-1) =$$

$$= \sum_{1 \leq i \leq m-1} (2^i-1)^k + \sum_{1<t<m-1} \frac{(2^{t+1}-1)^k(2^t-1) - (2^t-1)^k(2^{t+1}-1)}{2^t} \qquad [5.25]$$

e substituindo-se [5.25] e [5.20] em [5.19] vem:

$$T(h) = 2^{mk} - 1 - \sum_{1\leq i\leq m-1} (2^i-1)^k$$

$$- \sum_{1\leq t\leq m-1} \frac{(2^{t+1}-1)^k\,(2^t-1) - (2^t-1)^k\,(2^{t+1}-1)}{2^t}$$

$$= 2^{mk}-1-\left(\sum_{1\leq i\leq m-1} (2^i-1)^k + \sum_{1\leq t\leq m-1} \frac{(2^{t+1}-1)^k(2^t-1) - (2^t-1)^k(2^{t+1}-1)}{2^t} \right) \quad [5.26]$$

Chamando de C a expressão entre parênteses de [5.26] e resolvendo resulta:

$$C = \sum_{1\leq i\leq m-1} (2^i-1)^k + \sum_{1\leq t\leq m-1} \frac{(2^{t+1}-1)^k(2^t-1) - (2^t-1)^k(2^{t+1}-1)}{2^t}$$

$$= \sum_{1\leq i\leq m-1} (2^i-1)^k + \sum_{1\leq i\leq m-1} (2^{i+1}-1)^k - \frac{(2^{i+1}-1)^k}{2^i} - (2^i-1)^k.2 + \frac{(2^i-1)^k}{2^i}$$

$$= \sum_{1\leq i\leq m-1} \left[(2^{i+1}-1)^k-(2^i-1)^k- \frac{(2^{i+1}-1)^k}{2^i} + \frac{(2^i-1)^k}{2^i}\right] \quad [5.27]$$

Resolvendo-se em partes a equação [5.27] tem-se que:

$$\sum_{1\leq i\leq m-1} (2^{i+1}-1)^k - (2^i-1)^k = (2^m-1)^k-1 \quad [5.28]$$

e

$$\sum_{1\leq i\leq m-1} \left[\frac{(2^i-1)^k}{2^i} - \frac{(2^{i+1}-1)^k}{2^i}\right] = \frac{(2^1-1)^k}{2} - \frac{(2^2-1)^k}{2^2} - \frac{(2^3-1)^k}{2^3} - \cdots$$

$$\cdots - \frac{2^{m-1}-1}{2^{m-1}} - \frac{(2^m-1)^k}{2^{m-1}} + \frac{2^1-1}{2^2} - \frac{2^1-1}{2^2} =$$

$$= \frac{(2^1-1)^k}{2} + \frac{(2^1-1)^k}{2} - \frac{(2^m-1)^k}{2^{m-1}} - \sum_{1\leq i\leq m-1} \frac{(2^i-1)^k}{2^i} \quad [5.29]$$

Substituindo-se [5.28] e [5.29] em [5.27] vem:

$$C = (2^m-1)^k - 1 + 1 - \frac{(2^m-1)^k}{2^{m-1}} - \sum_{1 \leq i \leq m-1} \frac{(2^i-1)^k}{2^i} \qquad [5.30]$$

Substituindo-se [5.30] em [5.26] resulta:

$$T(h) = 2^{mk} - 1 - (2^m-1)^k + \frac{(2^m-1)^k}{2^{m-1}} + \sum_{1 \leq i \leq m-1} \frac{(2^i-1)^k}{2^i} \qquad [5.31]$$

Usando a expansão binominal,

$$(2^i-1)^k = \sum_{0 \leq j \leq k} \binom{k}{j} (-1)^j (2^i)^{k-j}$$

obtem-se :

$$\sum_{1 \leq i \leq m-1} \frac{(2^i-1)^k}{2^i} = \sum_{1 \leq i \leq m-1} \frac{1}{2^i} \sum_{0 \leq j \leq k} \binom{k}{j} (-1)^j (2^i)^{k-j}$$

$$= \frac{2^{(k-1)m}-1}{2^{k-1}-1} + O\ (2^{(k-2)m}) \qquad [5.32]$$

Desenvolvendo-se os termos da equação [5.31] tem-se:

$$(2^m-1)^k = 2^{mk} - k2^{m(k-1)} + O\,(2^{m(k-2)}) \qquad [5.33]$$

$$\frac{(2^m-1)^k}{2^{m-1}} \approx \frac{2^{mk}}{2^{m-1}} + O\ (2^{m(k-2)}) \qquad [5.34]$$

Finalmente, substituindo-se [5.32] , [5.33] e [5.34] em [5.31] obtem-se a expressão:

$$T(h) = 2^{mk} - 1 - (2^{mk} - k2^{m(k-1)} + O(2^{m(k-2)}) + \frac{2^{mk}}{2^{m-1}} + O(2^{m(k-2)}) +$$

$$+ \frac{2^{(k-1)m}-1}{2^{k-1}-1} + O(2^{m(k-2)})$$

$$\simeq k2^{m(k-1)} + 2^{m(k-1)+1} + \frac{2^{(k-1).m}-1}{2^{k-1}-1}$$

$$\simeq \left[(k+2) + \frac{1}{2^{k-1}-1}\right] 2^{m(k-1)}$$

$$= O(k.2^{mk-m})$$

$$= O(k.n^{1-(1/k)})$$

5.5 *Árvore k-d não Homogênea*

Assim como as árvores binárias de busca com uma chave, as árvores k-d podem ser implementadas de maneiras diferentes. A maneira mais comum de se representar uma árvore k-d é a que foi adotada até agora, chamada representação "homogênea", na qual estão baseados os algoritmos de inserção e busca apresentados. Na árvore homogênea todos os nós têm a mesma estrutura e a mesma função: cada nó tem a finalidade de armazenar um registro e dirigir a busca, pela chave que é o discriminante do nó. Os registros contêm campos para k chaves, um campo para o discriminante, campos de ponteiros e campos para outros atributos que eventualmente possam existir.

Uma outra forma de representação de árvores k-d é a "não-homogênea", apresentada por Bentley em [BENTLEY-79b]. Na árvore, não-homogênea existem dois tipos diferentes de nós:

internos e externos ou terminais. Os nós internos tem apenas a finalidade de dirigir a busca e não armazenam nenhum registro. Eles contêm um campo de discriminante, um campo de chave que é o valor do discriminante e campos de ponteiros. Os registros completos estão nos nós externos chamados "buckets", que são as folhas da árvore e podem conter um ou mais registros.

Por exemplo, considere-se um conjunto de registros de duas chaves, com discriminantes escolhidos ciclicamente , representado na forma homogênea como mostra a figura [5.37] , onde a ligação terra (⏚) representa ponteiro nulo.

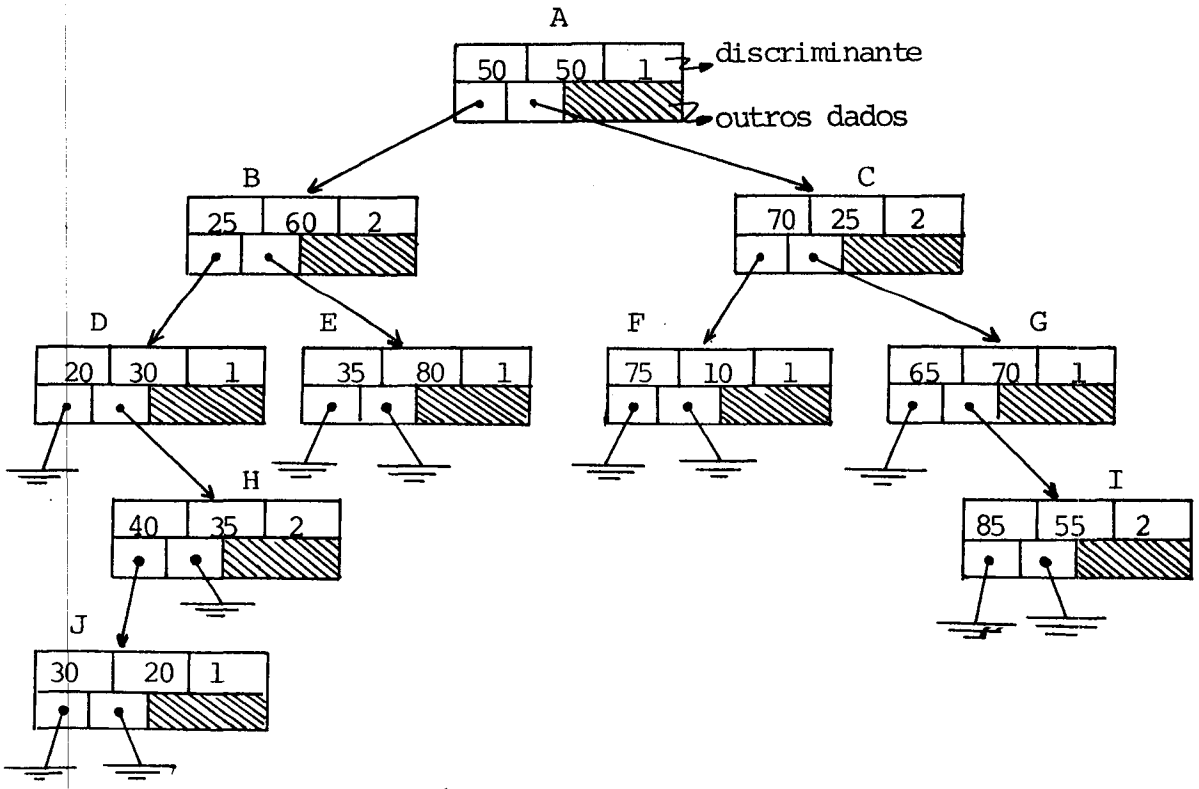


Fig. 5.37

Este mesmo conjunto, na representação não homogênea, assumindo que "buckets" podem armazenar até 3 registros, teria a forma mostrada na figura 5.38

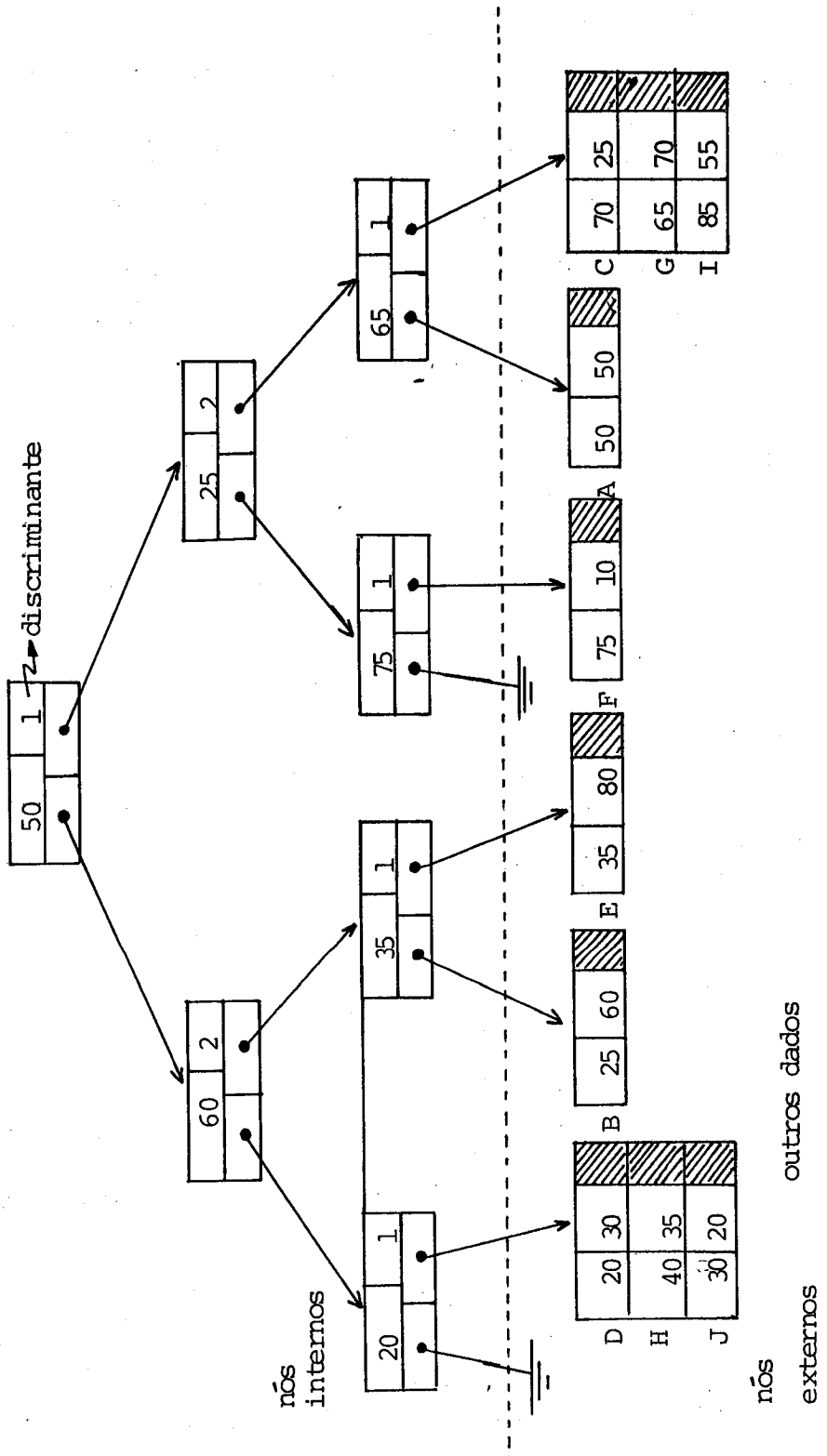


Fig. 5.38

Sendo usada esta representação, os algoritmos apresentados devem evidentemente sofrer algumas modificações.

A representação não homogênea é visivelmente superior quando os registros devem ser armazenados em memória secundária. Neste caso, os registros completos não tem que ser trazidos para a memória principal para fazer as comparações que decidem por qual subárvore o processo deve seguir . Pode-se agrupar os nós internos que estão próximos, na mesma página de disco e até mesmo compactá-los, como Knuth faz para árvores de uma dimensão [KNUTH-73], o que reduz o número de acessos a disco.

Ao contrário das árvores k-d, as árvores Quad - não tem a possibilidade de ser implementadas na forma não homogênea. Pela própria definição da estrutura, todos os nós precisam conter todos os campos de chaves, pois o espaço é particionado não só por uma chave, mas por k chaves, gerando 2^k novas regiões para cada região particionada.

5.6 *Conclusões*

As análises de busca por intervalo para árvores - k-d consideradas até agora levam à alguns resultados que permitem expressar os custos de pré-processamento, armazenamento de busca, resumindos a seguir.

- Pré-processamento:

Se uma árvore k-d é construída aleatóriamente, seu

comportamento é o mesmo de uma árvore binária unidimensional, portanto o custo dessa construção será em média $O(n\log_2 n)$, onde n é o número de nós da árvore. Se a árvore é construída com o algoritmo OTIMIZAR, para se obter uma árvore k-d ótima o custo de processamento de OTIMIZAR é $O(n\log_2 n)$ no pior caso. Logo, o custo de pré-processamento é dado por:

$$P_{kd}(n,k) = O(n\log_2 n)$$

- Armazenamento:

A árvore k-d é uma estrutura utilizada para o armazenamento de dados k-dimensionais, ou seja, registros com k chaves. Como são necessários apenas dois ponteiros por registro além das chaves, ela exige um espaço de armazenamento proporcional a kn. Portanto o custo de armazenamento de um arquivo com n registros é:

$$S_{kd}(n,k) = O(kn)$$

- Busca:

Na análise da busca por intervalo para árvores k-d balanceadas, com hiperplanos equidistantes e registros uniformemente distribuídos no hipercubo $[0,1]^k$ chega-se às seguintes conclusões:

- para registros com duas chaves, o número médio de nós visitados para uma região de busca com lados x_1 e x_2 é dado pela expressão:

$$E(n,x_1,x_2) = x_1x_2n+(3x_1+2x_2)\ ((n+1)^{1/2}-1) + \log_2(n+1)$$

- para registros com k chaves e o hiper-retângulo da busca determinado por intervalos $x_1,x_2,\ldots x_k$ esse número médio é dado por:

$$E(i,x_1,x_2,\ldots,x_k) = \sum_{0\leq i\leq h-1} \prod_{1\leq j\leq k} (x_j.2^{\lceil (i-j+1)/k \rceil}+1)$$

onde h é a altura da árvore.

Desenvolvendo-se esse valor, obtém-se:

$$E(n,x_1,\ldots,x_k) = n. \prod_{1\leq j\leq k} x_j + O(n^{1-(n/k)})$$

Então, para árvores k-d balanceadas, com hiperplanos equidistantes e registros uniformemente distribuidos no hipercubo $[0,1]^k$ o custo da busca por intervalo será em média dado por:

$$Q_{kd}(n,k) = n. \prod_{1\leq j\leq k} x_j + O(n^{1-(1/k)})$$

A análise do caso médio para árvores k-d aleatórias ainda é um problema em aberto.

Analisando-se o pior caso de busca por intervalo em uma árvore k-d balanceada com $n=2^{mk}-1$ nós onde $h=mk$ é a altura da árvore, para m inteiro, obtem-se uma expressão, assintótica do número de nós visitados (T(h)). Considera-se que visitas a nós que representam subespaços que estão completamente dentro da região de busca não são contadas. Então T(h) é da

-do por:

$$T(h) = O(k.n^{1-(1/k)})$$

Portanto, nestas condições, a busca por intervalo terá, no pior caso, um custo de:

$$Q_{kd}(n,k) = O(k.n^{1-(1/k)}).$$

6. OUTRAS ESTRUTURAS

Até agora foram investigadas algumas estruturas apropriadas para a busca por intervalo, que se apresentam como possíveis soluções práticas para esse problema. Outras estruturas semelhantes foram estudas por alguns autores, embora de interesse principalmente teórico em vez de prático.

Uma dessas estruturas é a "árvore intervalo" introduzida por Bentley [BENTLEY-79a]. Ela apresenta o melhor tempo de busca para o pior caso entre todas as estruturas já vistas mas tem custos de pré-processamento e armazenamento relativamente altos. A árvore intervalo é uma estrutura que tem a interessante propriedade da recursividade na dimensão, isto é, a estrutura k-dimensional é definida em termos da estrutura (k-1) dimensional. Apresenta o seguinte comportamento:

$$P_{ai}\ (n,k) = O(n \log_2^{k-1} n)$$

$$S_{ai}\ (n,k) = O(n \log_2^{k-1} n)$$

$$Q_{ai}\ (n,k) = O(\log_2^{k} n + F')$$

onde F' é o número de registros encontrados na região de busca.

Para a maioria das aplicações, o gasto necessário com memória torna-a uma estrutura inadequada limitando seu uso aos casos em que k = 2 ou 3, mas trata-se de uma estrutura muito interessante do ponto de vista teórico.

Uma outra estrutura é o "intervalo-k" introduzida

por Bentley e Maurer [BENTLEY-80b], que também comporta-se eficientemente no pior caso de busca por intervalo. Eles desenvolveram dois tipos de intervalo-k, "overlapping" e "nonoverlapping". Como a estrutura intervalo-k é bastante complicada para se descrever, e é de interesse fundamentalmente teórico, vamos apenas mencionar seu comportamento. A estrutura intervalo-k do tipo overlapping tem os custos:

$$P_O(n,k) = O(n^{1+\varepsilon})$$

$$S_O(n,k) = O(n^{1+\varepsilon})$$

$$Q_O(n,k) = O(n \log_2 n + F')$$

onde F' é o número de registros encontrados na região de busca, para qualquer $\varepsilon > 0$, o que mostra que o tempo de busca é muito eficiente, mas os custos de pré-processamento e armazenamento são altos.

A estrutura intervalo-k nonoverlapping possui os custos:

$$P_n(n,k) = O(n \log_2 n)$$

$$S_n(n,k) = O(n)$$

$$Q_n(n,k) = O(n^{\varepsilon})$$

para qualquer $\varepsilon > 0$ fixo. É mais eficiente com relação ao pré-processamento e armazenamento mas tem um tempo de busca maior.

Outras estruturas, ou variações dessas que foram apresentadas têm sido propostas como solução para o problema

de busca por intervalo como a "Árvore Quintary" de [LEE -80] e a "Árvore k-d-b" introduzida por [ROBINSON-81] . Outros trabalhos sobre problemas de busca por chaves multiplas semelhantes à busca por intervalo foram feitos por Dobkin e Lipton - [DOBKIN-76] e por Bentley [BENTLEY-80a] . Para discussão de métodos de busca eficientes no contexto de sistemas de base de dados as referencias são [LION-77], [SHNEIDERMAN-77], [YANG-77] e [YANG-78] .

As estruturas analisadas no decorrer deste trabalho e as introduzidas neste capítulo parecem refletir satisfatóriamente a situação em que se encontram as pesquisas no assunto.

7. CONCLUSÕES FINAIS E TRABALHO ADICIONAL

Nos capítulos anteriores foram discutidas algumas estruturas para busca por intervalo. As cinco primeiras estruturas foram apresentadas como possíveis soluções práticas para o problema, e para cada uma delas óbviamente existem situações nas quais ela é claramente superior e outras situações nas quais ela é inadequada. As outras três estruturas considerando-se os tipos "overlapping" e "nonoverlapping" de Intervalo-k são de interesse teórico. O comportamento dessas oito estruturas pode ser visto na tabela 7a que mostra os custos de pré-processamento, armazenamento e busca encontrados nas análises das estruturas. Estes custos são apresentados para o pior caso, a menos que esteja explicitado o contrário. Condições especiais dos dados assumidas na análise da busca são observadas abaixo da tabela e F' significa o número de registros encontrados na busca.

A discussão das estruturas de dados apresentada está em um nível conceitual bastante abstrato e têm sido ignorados muitos dos problemas que surgem nas aplicações reais de busca por intervalo. De um modo geral, do ponto de vista prático, é importante que essas estruturas sejam implementadas em sistemas de base de dados reais para ver como a teoria se relaciona com a prática. Os exemplos de aplicação citados são bem simples e servem apenas para ilustrar o tipo de consulta que pode ser feito com busca por intervalo. Um exemplo concreto de aplicação de árvores k-d é apresentado em [BENTLEY-79b],

e situações onde elas poderiam ser aplicadas são mostradas em [BENTLEY-75a].

Tabela 7a

Comportamento de Estrutura de Dados para Busca por Intervalo

Estrutura	p(n,k)	S(n,k)	Q(n,k)	
Busca Sequencial	$0(n)$	$0(n)$	$0(n)$	
Projeção	$0(n\log_2 n)$	$0(n)$	$0(n^{1-1/k}+F')$	(*1)
Células	$0(n)$	$0(n)$	$0(F')$	(*2)
Árvores Quad	$0(n\log_2 n)$	$0(n)$	$0(k.n^{1-1/k})$	
Árvores k-d	$0(n\log_2 n)$	$0(n)$	$0(k.n^{1-1/k})$	
Intervalo k "nonoverlapping"	$0(n\log_2 n)$	$0(n)$	$0(r^{\epsilon}+F')$	
Árvore intervalo	$0(n\log_2^{k-1} n)$	$0(n\log_2^{k-1} n)$	$0(\log_2^k n+F')$	
Intervalo k "overlapping"	$0(n^{1+\epsilon})$	$0(n^{1+\epsilon})$	$0(n\log_2 n+F')$	

(*1) Análise do caso médio, para conjunto de dados regular e região de busca pequena.

(*2) Análise do caso médio para qualquer conjunto de dados e tamanho de célula igual ao tamanho da região de busca.

Um outro ponto a ser ressaltado é que os arquivos considerados nas discussões dos processos e análises de busca são estáticos, ou seja não foi considerado o problema de como manter as estruturas quando as operações de inserção, eliminação e busca podem ser feitas alternadamente.

Nas estruturas de busca sequencial, projeção e células alterações são feitas de forma razoávelmente simples. Pa

ra árvores Quad e k-d porém, problemas como determinar o ponto de reorganização da árvore depois que um número de atualiza - ções tenha sido realizado, permanecem ainda em aberto. Árvores k-d dinâmicas são discutidas por Bentley em [BENTLEY-79b].

As discussões também estão todas voltadas para o caso em que as estruturas estão implementadas em memória principal. Quando é necessário usar memória secundária como discos e fitas, todas as estruturas podem ser eficientemente implementadas nesses meios.

As análises de Busca Sequencial, Projeção e Células podem ser feitas facilmente, mas para árvores Quad e k-d os resultados encontrados são parciais, como no caso da busca por intervalo em que se considera um tipo muito particular de árvore, como a árvore perfeita. Para o pior caso, encontra-se um valor assintótico para essas estruturas considerando árvores balanceadas e nós uniformemente distribuídos. O problema do caso médio da busca por intervalo para árvores Quad ou K-d aleatórias ainda é um problema em aberto.

Outro problema relativo a árvores k-d que merece ser mais estudado é o método de escolha do discriminante. Silva Filho apresenta uma maneira ótima de escolha discriminantes - por nível [SILVA-78b] mas o problema de escolher discriminantes por nó permanece em aberto.

Consideráveis pesquisas ainda podem ser feitas no desenvolvimento de heurísticas para melhorar os métodos de busca que têm sido vistos. Técnicas descritas por Bentley e - Burkard [BENTLEY-76] podem ser úteis neste tipo de pesquisa.

Um problema interessante que pode ter valor práti

co é provar limites inferiores na complexidade do problema de busca por intervalo.

Outra questão interessante a ser resolvida colocada por Bentley e Maurer [BENTLEY-80b] é a seguinte: existem outras estruturas com uma relação melhor entre P(n,k), S(n,k) e Q(n,k)? Em particular, para um total de $2k \log_2 n$ comparações, $O(n^{2k-1})$ é ótimo para o armazenamento? Pode o produto $P(n,k) \cdot S(n,k) \cdot Q(n,k)$ ser reduzido a $O(n^2 \log_2^2 n)$? Se não, pode-se mostrar limites inferiores no produto acima, indicando como tempo e espaço se relacionam?

Como pode ser visto, existem vários problemas em aberto relativos à busca por intervalo que merecem ser investigados. Os trabalhos futuros devem dirigir-se básicamente no sentido de melhorar as análises formais das estruturas existentes, buscar novas estruturas com comportamentos melhores e desenvolver aplicações práticas para que se obtenha resultados mais concretos.

Este trabalho discute estruturas que apareceram como uma primeira tentativa de resolução do problema de busca por intervalo. Algumas delas mostram que podem ser úteis na implementação de problemas reais em diversas áreas. Outras mostram resultados teóricos que sugerem ser possível encontrar métodos para se resolver o problema de busca por intervalo eficientemente do ponto de vista global de seu comportamento, combinando-se os custos de pré-processamento, armazenamento e busca.

REFERÊNCIAS

[BENTLEY-75a] BENTLEY, J.L.: "Multidimensional binary search trees used for associative searching". *Communications of* ACM, 18(9): 509-517, set.1975.

[BENTLEY-75b] BENTLEY, J.L. e STANAT, D.F.: "Analysis of range searches in quad trees". *Information Processing Letters*, 3(6): 170-173, jul.1975.

[BENTLEY-76] BENTLEY, J.L. e BURKHARD, W.A.: "Heuristics for parcial match retrieval data base design". *Information Processing Letters*, 4(5): 132-135, fev.1976.

[BENTLEY-77] BENTLEY, J.L., STANAT, D.F., e WILLIAMS, E.H. Jr.: "The complexity of fixed-radius near neighbor searching". *Information Processing Letters*, 6(6):209-212, dez.1977.

[BENTLEY-79a] BENTLEY, J.L.: "Decomposable searching problems". *Information processing letters*, 8(5):133-136, jun.1979.

[BENTLEY-79b] BENTLEY, J.L.: "Multidimensional binary search trees in database applications". *IEEE Trans. Softw. Eng.*, SE-5(4):333-340, jul.1979.

[BENTLEY-79c] BENTLEY, J.L. e FRIEDMAN, J.H.: "Data Structures for Range Searching". *Computing Surveys*, 11(4): 397-409, dez.1979.

[BENTLEY-80a] BENTLEY, J.L.: "Multidimensional divide-and-conquer". *Communications of the ACM*, 23(4):214-229, abr.1980.

[BENTLEY-80b] BENTLEY, J.L. e MAURER, H.A.: "Efficient worst-case data structures for range searching". *Acta Informática*, 13:155-168, 1980.

[DOBKIN-76] DOBKIN, D. e LIPTON, R.J.: "Multidimensional searching problems". *SIAM J.Comput.* 5(2):181-186, jun.1976.

[FINKEL-74] FINKEL, R.A. e BENTLEY, J.L.: "Quad trees-a data structure for retrieval on composite keys". *Acta Informatica*, 4(1):1-9, 1974.

[FRIEDMAN-75] FRIEDMAN, J.H., BASKETT, F. e SHUSTEK, L.J.: "An algorithm for finding nearest neighbors". *IEEE Trans.Comput.*, C-24(10):1000-1006, out.1975.

[FRIEDMAN-77] FRIEDMAN, J.H., BENTLEY, J.L., e FINKEL, R.A.: "An algorithm for finding best matches in logarithmic expected time". *ACM Trans. Math. Softw.*, 3(3):209-226, set.1977.

|GOTILIEB-75| GOTLIEB , C.C. and GOTLIEB, L.R., *Data types and Structures*, Prentice-Hall, Englewoord Chiffs, N.J, 357-363,1975.

[KNUTH-68] KNUTH, D.E., *The art of computer programming*, vol. 1: *Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.

[KNUTH-73] KNUTH, D.E., *The art of computer programming*, vol.3: *Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

[LEE-76] LEE, R.C.T., CHIN, Y.H., and CHANG, S.C.: "Application of principal component analysis to multi-key searching". *IEEE Trans.Softw. Eng.*, SE-2(3):185-193, set.1976.

[LEE-78] LEE, D.T. and WONG, C.K.: "Worst-case analysis for region and parcial region searches in multidimensional binary search trees and quad trees". *Acta Informatica*, 9(1):23-29, 1978.

[LEE-80] LEE, D.T., and WONG, C.K.: "Quintary trees: a file structure for multidimensional database systems", *ACM Trans. Database Syst.*, 5(3):339-453, set.1980.

[LIOU-77] LIOU, J.H. and YAO, S.B.: "Multi-dimensional clustering for data base organization". *Inf.Syst.*,

2:187-198, 1977.

[ROBINSON-81] ROBINSON, J.T.: "The K-D-B-Tree: a search structure for large multidimensional dynamic indexes. "Department of Computer Science, Carnegie-Mellon University, fev.1981

[SCARPELLI-83] SCARPELLI, H.A.C., e MONARD, M.C.: "Pré-processamento para o problema do vizinho mais próximo". Trabalho apresentado no VI Congresso Nacional de Matemática Aplicada e Computacional, set.1983.

[SHNEIDERMAN-77] SHNEIDERMAN, B.: "Reduced combined indexes for efficiente multiple attribute retrieval". *Inf.Syst.*, 2:149-154, 1977.

[SILVA-79] SILVA-FILHO, Y.V.: "Average case analysis of region search in balanced k-d trees". *Information Processing Letters*, 8(5):219-223, jun.1979.

[SILVA-81] SILVA-FILHO, Y.V.: "Optimal Choice of discriminators in a balanced k-d binary search tree". *Information Processing Letters*, 67-70, nov.1981.

[YANG-77] YANG, C. "Avoiding redundant record accesses in unsorted multilist file organizations" *Inf.Syst.*, 2:155-158, 1977.

[YANG-78] YANG, C.: "A class of hibrid list file organizations".

Inf.Syst., 3:49-58, 1978.

[YUVAL-75] YUVAL, G.: "Finding near neigbors in k-dimensional space". *Information Processing Letters*, 3(4): 113-114, mar. 1975.