6 Métodos de acesso para dados espaciais

Clodoveu A. Davis Jr.
Gilberto Ribeiro de Queiroz

6.1 Introdução

Este capítulo aborda alguns dos métodos de acesso espacial mais tradicionais, denominados *k-d trees, grid files, quad-trees* e *R-trees*.

Métodos de acesso espacial, ou índices espaciais, são estruturas de dados auxiliares, mas essenciais para o processamento eficiente de consultas espaciais. De fato, normalmente uma consulta espacial envolve apenas uma pequena parcela do banco de dados. Neste caso, percorrer todo o banco pode ser bastante ineficiente. Portanto, um plano de execução para a consulta tipicamente começa com uma fase de filtragem, que identifica quais objetos podem satisfazer a qualificação da consulta. Esta fase depende da existência de índices espaciais, em geral definidos sobre aproximações das geometrias dos objetos.

6.2 Preliminares sobre métodos de acesso

Uma forma comum de indexarmos um conjunto de dados é através do uso de árvores de pesquisa. As mais simples e conhecidas são as árvores binárias, como: AVL, Red Black e Splay Tree (Cormen et al., 1990), que são árvores balanceadas, ou seja, todos os caminhos desde a raiz até as folhas possuem o mesmo comprimento. Essas estruturas permitem que algumas operações, como a localização de um elemento, sejam executadas em tempo logarítmico – $O(\log n)$. A Figura 6.1 ilustra a representação de uma árvore binária balanceada, onde as cidades estão organizadas segundo a ordem alfabética de seus nomes. Esse tipo de estrutura é empregado para uso em memória principal.

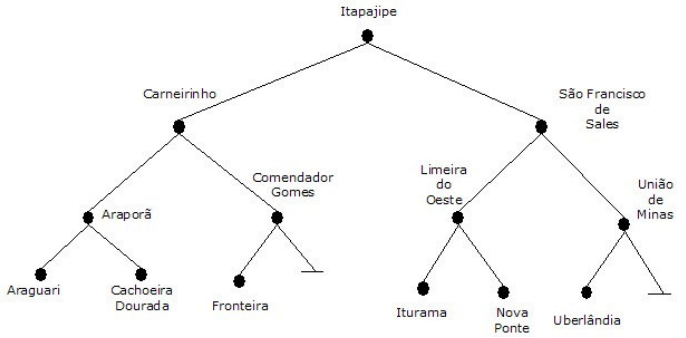

Figura 6.1 – Árvore binária balanceada.

No caso de grandes bancos de dados onde a memória principal pode não ser suficiente para armazenar todos os nós da árvore, é comum armazená-la em disco. Nesse caso utilizamos uma representação que procura minimizar o acesso a disco. A forma mais comum, e mais largamente empregada pelos sistemas comerciais atuais, é a representação do índice através de uma árvore B^+ (Comer, 1979). Conforme podemos observar na Figura 6.2, essa estrutura tenta minimizar o número de acessos a disco durante o processo de pesquisa agrupando várias chaves em um único nó, denominado de página.

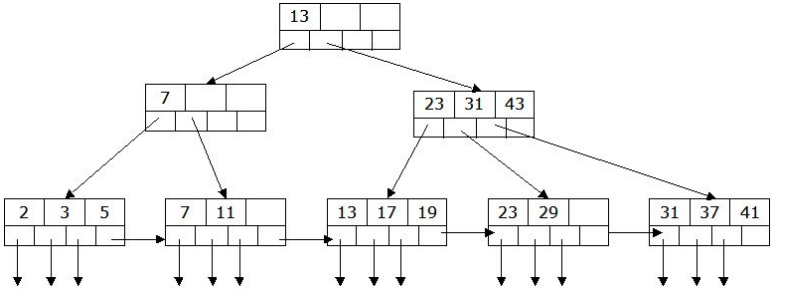

Figura 6.2 – Índice unidimensional B-Tree. Fonte: adaptada de Garcia-Molina et al., (2001).

Em comum, tanto a árvore binária quanto a B^+, são estruturas unidimensionais, isto é, elas pressupõem que a chave de pesquisa seja

formada por apenas um atributo ou pela concatenação de vários atributos. Em sistemas que trabalham com informações multidimensionais (mais de um atributo), como os sistemas de bancos de dados espaciais, estas estruturas não são diretamente aplicáveis.

Para esses sistemas existe uma classe de métodos conhecidos como *métodos de acesso multidimensionais*. No caso dos bancos de dados espaciais, estes métodos estão ligados ao processamento de consultas típicas como: consulta por apontamento (encontrar os objetos que contém um dado ponto), consultas por região (encontrar os objetos dentro de uma janela ou retângulo) e consultas com predicados topológicos (encontrar os objetos vizinhos de um determinado objeto).

Uma idéia fundamental dos índices espaciais é o uso de aproximações, isto é, a estrutura do índice trabalha com representações mais simples dos objetos, como o menor retângulo envolvente (REM) do objeto. Dessa forma, a maneira mais comum de processar as consultas é dividir o processamento em duas etapas: uma de filtragem e outra de refinamento (Figura 6.3). Na etapa de filtragem são usados métodos de acesso espaciais. O principal objetivo do uso destes métodos é o de reduzir e rapidamente selecionar os possíveis candidatos que satisfaçam a consulta. A redução do espaço de busca é muito importante, pois a etapa seguinte, a de refinamento, envolve a aplicação de algoritmos geométricos computacionalmente complexos e custosos e que são aplicados à geometria exata dos candidatos selecionados na etapa anterior.

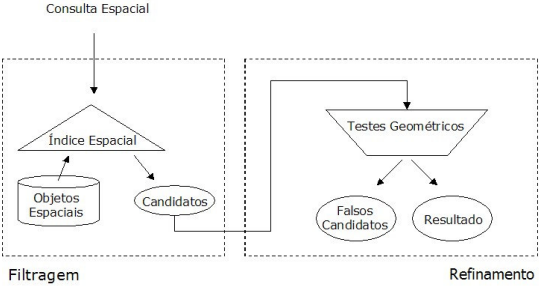

Figura 6.3 – Processamento de consultas espaciais. Fonte: adaptada de Gaede e Günther (1998).

6.3 *k*-d Tree

A *k*-d tree é uma árvore binária, ou seja, cada nó interno possui dois descendentes não importando a dimensionalidade *k* do espaço envolvido. Originalmente, esta árvore é apresentada em Bentley (1975), como uma alternativa a um problema mais genérico do que o geométrico dos bancos de dados espaciais: o de busca baseado em vários atributos.

Em um banco de dados tradicional, uma consulta como: "encontrar todos os empregados nascidos entre 1950 e 1955 que recebem entre R$3.000,00 e R$5.000,00", poderia ser vista como uma pesquisa em duas dimensões, considerando cada atributo, neste caso data de nascimento e salário, como uma das componentes do espaço (Figura 6.4a). No caso dos sistemas de bancos de dados espaciais, essa consulta equivale a encontrar todos os pontos no interior do retângulo definido pelos intervalos (3000:5000)x(1950:1955) (Figura 6.4b).

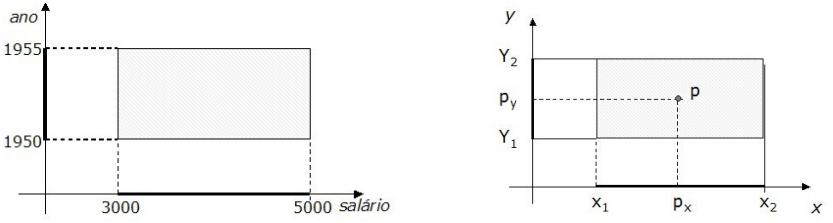

Figura 6.4 – Busca em dois atributos (a) Interpretação geométrica da busca (b).

Dada uma chave formada por *k* atributos (*k* dimensões), onde k_i é o valor da *i-ésima* componente da chave, a idéia básica da *k*-d tree é a seguinte:

- A cada nó da árvore associamos uma chave e um discriminador;
- O discriminador é um valor entre 0 e *k*-1 que representa a dimensão ao longo da qual o espaço será subdividido, ou seja, a componente k_i que será empregada para definição da ordem na árvore;
- Todos os nós do mesmo nível são associados ao mesmo valor de discriminador;

- O nó raiz está associado ao discriminador 0, aos dois descendentes diretos, discriminador 1, e assim por diante, até o k-ésimo nível que está associado ao discriminador k-1;
- A partir do nível k+1 o ciclo começa se ser repetido, sendo associado o discriminador 0 a este nível.
- A ordem imposta por essa árvore é tal que, dado um nó qualquer P de discriminador j, qualquer nó descendente (L) da sub-árvore esquerda possui um valor de chave cuja componente k_j é menor do que a componente k_j do nó P;
- Da mesma forma, qualquer nó descendente (R) da sub-árvore direita possui um valor de chave cuja componente k_j é maior do que a componente k_j do nó P.

Para o caso de pontos no espaço bidimensional, como o da Figura 6.5, a k-d tree ou 2-d tree que corresponde à decomposição do espaço ao longo das dimensões x e y, compara os valores da componente "x" nos níveis pares da árvore e da componente "y" nos níveis ímpares.

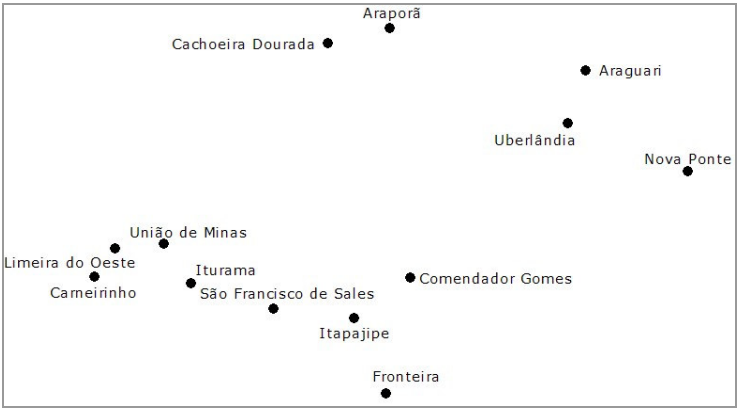

Figura 6.5 – Sedes de alguns municípios do Estado de Minas Gerais.

Assim, se adotarmos uma ordem arbitrária de entrada dos pontos correspondentes às sedes municipais, podemos construir a árvore da Figura 6.6 da seguinte forma:

- Considerando Araguari a primeira cidade a ser inserida na árvore, esta ocupará a raiz;
- Sendo Araporã, a próxima, como ela possui um valor de coordenada "*x*" menor do que a de Araguari, esta é inserida à esquerda. Aqui, como a raiz ocupa um nível par (0), comparamos os valores da componente "*x*" das coordenadas das sedes municipais.
- Nova Ponte possui um valor "*x*" maior, e, portanto, é inserida à direita.
- No caso de Cachoeira Dourada, ela possui um valor de "*x*" menor do que Araguari (comparação em um nível par), e um valor de "y" menor do que Araporã (comparação no nível ímpar – 1), e por isso é inserida à esquerda desta.
- Esse processo é realizado até que todas as cidades estejam armazenadas na árvore.

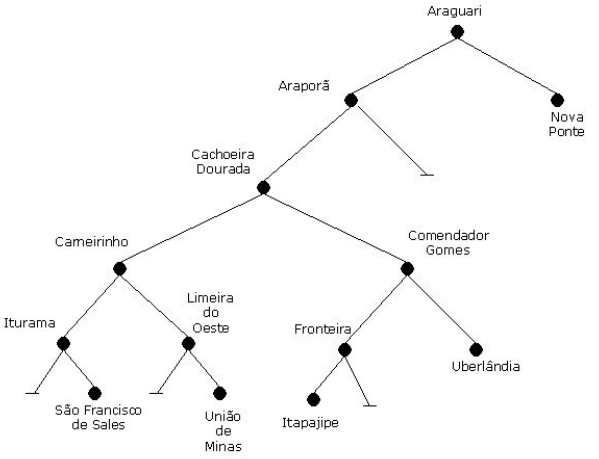

Figura 6.6 – Representação em forma de árvore para as sedes da Figura 6.5.

A partição do espaço representada pela *k*-d tree do exemplo acima pode ser vista na Figura 6.7.

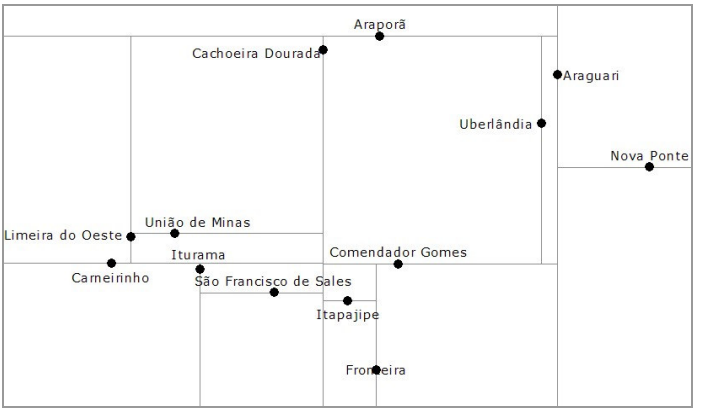

Figura 6.7 – Representação planar para as sedes da Figura 6.5.

Um dos problemas que podemos observar na *k*-d tree da Figura 6.6 é que ela depende da ordem em que os nós são inseridos. No pior caso, podemos acabar obtendo uma árvore completamente desbalanceada, que apresenta a mesma profundidade do número de elementos inseridos. Uma forma de contornar este problema é adotar um método que otimize a construção da árvore. Uma forma de construir a árvore mais balanceada é escolher o elemento que ocupa a posição média ao longo da componente em divisão. A Figura 6.8 ilustra uma árvore construída tomando-se o elemento central em relação à componente em cada nível da divisão.

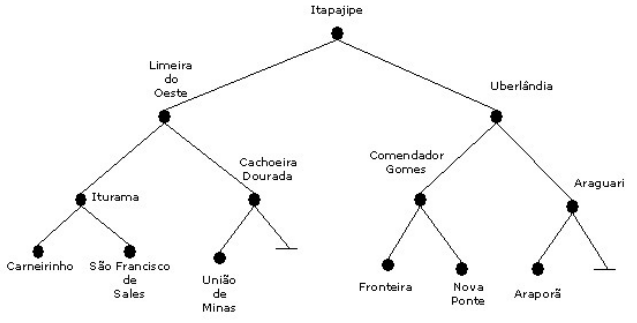

Figura 6.8 – Representação em árvore (balanceada) para as sedes.

Na Figura 6.9, podemos observar que Itapajipe possui o mesmo número de elementos do lado esquerdo e direito e por isso foi tomada como o primeiro elemento a ser inserido na árvore da Figura 6.8. Limeira do Oeste pode ser tomada como elemento médio entre os elementos à esquerda de Itapajipe quando comparamos ao longo do eixo *y*.

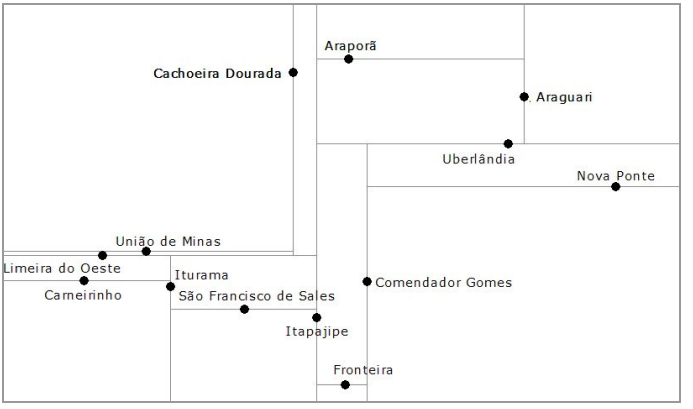

Figura 6.9 – Representação planar para as sedes da Figura 6.5.

A pesquisa por intervalos na 2-d tree do exemplo anterior pode ser realizada da seguinte forma:

1. Seja um retângulo de coordenadas inferior esquerda (x_1:y_1) e superior direita (x_2:y_2) contendo o intervalo de pesquisa, partindo da raiz, temos que verificar:
2. Se o nó corrente encontra-se no intervalo ele é reportado;
3. Se ele for um nó em um nível par (caso do nó raiz):
    a. Se a coordenada "x_1" do retângulo de pesquisa for menor do que a coordenada "x" do nó, aplicamos recursivamente os passos a partir do passo 2, à sub-árvore esquerda;
    b. Se a coordenada "x_2" do retângulo de pesquisa for maior do que a coordenada "x", aplicamos recursivamente os passos a partir do passo 2, à sub-árvore direita;

4. Se ele for um nó em um nível ímpar:
    a. Se a coordenada "y_1" do retângulo de pesquisa for menor do que a coordenada "y" do nó, aplicamos recursivamente os passos a partir do passo 2, à sub-árvore esquerda;
    b. Se a coordenada "y_2" do retângulo de pesquisa for maior do que a coordenada "y", aplicamos recursivamente os passos a partir do passo 2, à sub-árvore direita;

Na literatura, encontramos diversas variantes dessa estrutura. Friedman et al. (1977) apresentam uma variante em que os elementos (registros ou pontos) são armazenados apenas nos nós folha, sendo que os nós internos contém apenas os valores usados para particionar o espaço. Robinson (1981), combinando a B-Tree, adaptou esta estrutura para armazenamento em memória secundária, chamando esta nova estrutura de *k*dB-Tree.

6.4 Grid File

Uma das técnicas mais interessantes para facilitar a pesquisa em dados convencionais é o *hashing*. Este método, chamado também de "transformação de chave", consiste em criar uma série de receptáculos (chamados habitualmente de *buckets*), que receberão os identificadores dos itens que se quer pesquisar. Estes *buckets* são numerados seqüencialmente de 1 a *n*, e portanto sua implementação mais natural é sob a forma de arranjo (Figura 6.10). Cada identificador que chega, seja para ser inserido, seja para ser pesquisado, é transformado em um número de 1 a n, identificando o bucket correspondente a ele. Então, só se precisa inserí-lo no *bucket* (no caso de inserção) ou procurar um elemento com identificador igual que já esteja dentro do *bucket*. Quanto maior for o valor de *n*, menos itens existirão em média dentro de cada *bucket*, e o resultado da pesquisa será mais rápido. No entanto, a escolha do valor de *n* e da função de transformação ideal é problemática, pois não se deseja provocar uma distribuição irregular dos itens pelos *buckets*. Em geral, recomenda-se utilizar *n* primo, e uma função de transformação que (1) seja simples de ser computada, e (2) produza uma distribuição uniforme de saídas entre 1 e *n*.

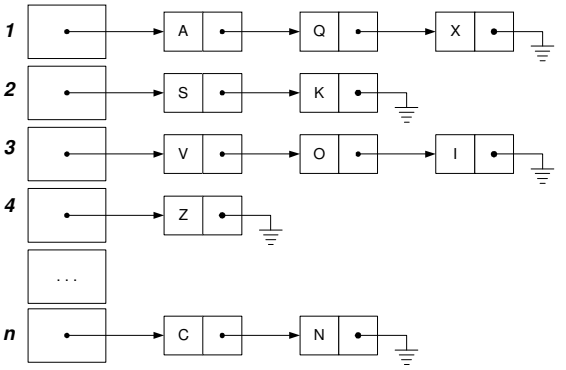

Figura 6.10 – *Hashing.*

O método de indexação chamado *grid file* (Nievergelt et al., 1984). estende o conceito de *hashing* para duas dimensões. Ou seja, em vez de *n buckets*, teremos uma grade regular que cobre todo o espaço de pesquisa. O método foi originalmente concebido para possibilitar pesquisas convencionais, como no caso do *hashing*, porém utilizando duas variáveis distintas. No caso geográfico, é natural imaginar quadrados sobre a superfície da Terra, que funcionam como os *buckets* do *hashing* (Figura 6.11). A partir de cada elemento da grade (que pode ser modelada na memória com o uso de uma matriz no lugar do vetor utilizado no *hashing*) são organizadas listas lineares contendo os identificadores dos objetos que estão localizados naquela área.

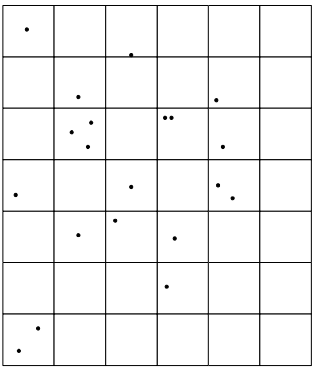

Figura 6.11 – *Grid File.*

Alguns tipos de pesquisas podem ser feitas com muita facilidade utilizando *grid files*. Por exemplo, deseja-se saber quais pontos estão a menos de *M* metros de um determinado ponto *P*. Para isto, basta calcular as coordenadas de dois pontos auxiliares, um deles subtraindo *M* de ambas as coordenadas, e no outro somando:

$$x_1 = x_P - M$$
$$y_1 = y_P - M$$
$$x_2 = x_P - M$$
$$y_2 = y_P - M$$

Depois, determina-se qual é a linha e a coluna dos quadrados que contêm os pontos x_1 e x_2. Todos os pontos procurados estarão nos quadrados compreendidos entre os limites expressos pelas linhas e colunas encontradas. Naturalmente, deve-se calcular a distância de cada ponto encontrado a *P*, para verificar o atendimento à restrição de distância, pois a pesquisa na realidade foi feita usando um quadrilátero, e não com um círculo.

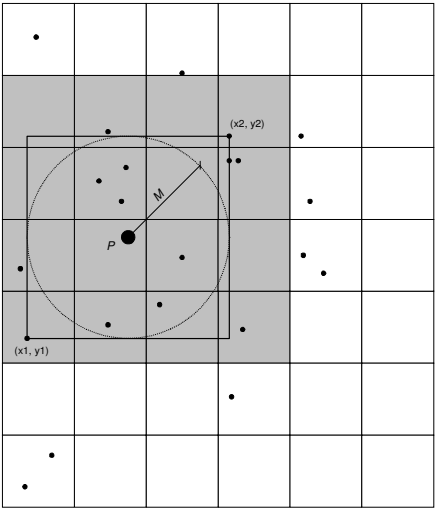

Figura 6.12 – Localização de pontos por proximidade.

A principal limitação deste método, superada pelos métodos que serão apresentados a seguir, é o fato de somente poder lidar com pontos, ou com objetos que caibam inteiramente em um quadrado. Não é um método adequado para lidar com linhas ou polígonos, mas é bastante eficiente e simples de implementar para o tratamento de dados puntuais.

6.5 Quad-tree

A quad-tree é um tipo de estrutura de dados organizada em árvore, em que cada "nó" ou "tronco" gera sempre (e exatamente) quatro "folhas", como mostrado na Figura 6.13 (Samet, 1984). O SIG, utilizando esta estrutura de dados para realizar a indexação espacial, considerará que cada nó corresponderá a uma região quadrada do espaço. Esta região será subdividida, novamente em quatro partes, gerando mais um nível na árvore, e assim sucessivamente, até que se chegue a ter um ou nenhum objeto geográfico dentro dos quadrados resultantes da subdivisão.

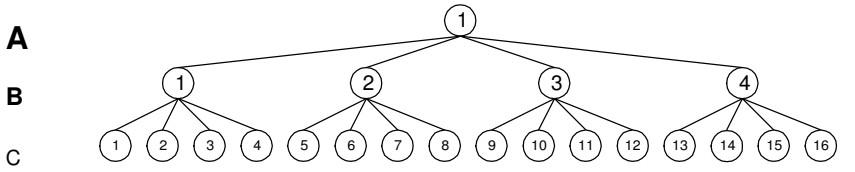

Figura 6.13 – Quad-tree.

Na Figura 6.13 existem três níveis (A, B e C) na quad-tree. A cada um destes níveis corresponderá um quadrado geográfico, conforme apresentado na Figura 6.14.

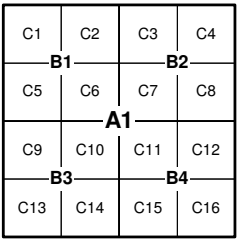

Figura 6.14 – Subdivisão do espaço por uma quad-tree.

Esta subdivisão poderá continuar indefinidamente, de forma recursiva: basta considerar um quadrado do nível C como sendo o quadrado A1, e reaplicar o processo. Observe-se que o resultado final será uma árvore que terá um número variável de níveis, de acordo com a região geográfica: regiões mais densas formarão mais níveis, e regiões menos densas terão menos níveis.

O processamento de consultas por região (janela ou retângulo) nessa estrutura inicia-se a partir da raiz, selecionando apenas as ramificações cujos quadrados estiverem em contato com o retângulo da região. Se nenhuma estiver, todas as ramificações abaixo daquele nó são descartadas. Se alguma estiver, toda a ramificação abaixo daquele ponto é selecionada, e sucessivamente testada, até que se alcance os objetos geográficos.

6.6 Tiling

Este processo é semelhante ao anterior, com a diferença de que as subdivisões não prosseguem indefinidamente.

O processo consiste em, conhecendo-se a priori os limites geográficos da base de dados, criar camadas de "tiles", ou ladrilhos, quadrados, de dimensões fixas. O número de camadas e a largura dos ladrilhos de cada camada são determinados pelo usuário, de modo a corresponder da melhor maneira possível à variedade de dimensões dos objetos geográficos que serão encontrados na base de dados. O número total de ladrilhos endereçáveis é limitado pelo sistema, para melhorar sua performance.

Cada objeto será associado a um ladrilho, que será o menor ladrilho que contiver inteiramente o objeto. Assim, a cada ladrilho, de cada camada, serão associados diversos objetos, formando uma lista. No momento do display, testa-se os limites da janela de visualização contra a lista de ladrilhos, selecionando aqueles que tenham pelo menos uma extremidade contida no retângulo. Os objetos associados a cada um destes ladrilhos serão, então, recuperados na base de dados para apresentação.

Com estes limites previamente calculados, evita-se percorrer freqüentemente uma estrutura de dados como a quad-tree, que geralmente tem que ser mantida em disco.

A Figura 6.15 ilustra a estrutura básica do *tiling*.

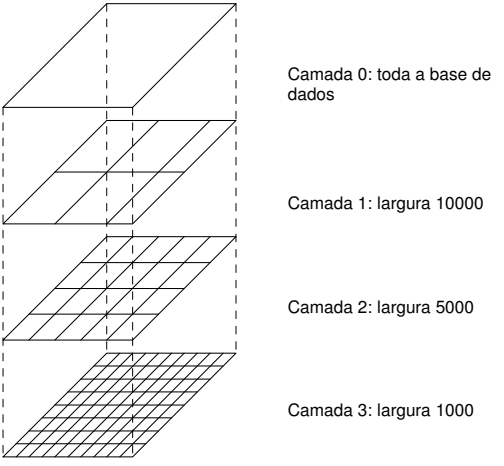

Figura 6.15 – Tiling.

6.7 R-tree

A *R-tree*, ou árvore-R, é uma estrutura de dados hierárquica derivada da árvore-B. As diferenças estão centradas na natureza das chaves: valores numéricos ou alfanuméricos simples, no caso das árvores-B, e pontos extremos de retângulos, no caso das árvores-R (Guttman, 1984).

O que a árvore-R busca organizar não é exatamente o contorno ou a forma gráfica do objeto, e sim o seu *retângulo envolvente mínimo* (*minimum bounding rectangle*, MBR). Este retângulo é formado a partir da observação dos limites geométricos mínimo e máximo do contorno do objeto, e é expresso pelas coordenadas dos seus pontos inferior esquerdo e superior direito (Figura 6.16). No caso de objetos puntuais, estes extremos vão coincidir com as coordenadas do próprio objeto (portanto formando um retângulo nulo), mas isto não compromete o método.

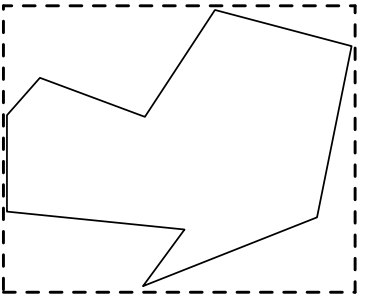

Figura 6.16 – Objeto poligonal e seu retângulo envolvente mínimo.

A árvore-R possui parâmetros para determinar a quantidade de chaves (retângulos) que poderão ocorrer em cada bloco de armazenamento, analogamente à árvore-B, em função do tamanho da página de armazenamento em disco.

As regras básicas para formação de uma árvore-R são semelhantes às da árvore-B. Todas as folhas aparecem sempre no mesmo nível da árvore. Nós internos contêm a delimitação de retângulos que englobam todos os retângulos dos nós nos níveis inferiores (Figura 6.17). Uma árvore-R de ordem (m, M) conterá entre m e M entradas em cada nó da árvore ($m \leq M / 2$), exceto a raiz, que terá pelo menos dois nós (a menos que a árvore só tenha uma entrada).

Um problema com as árvores-R é que a ordem de inserção dos objetos interfere na forma final da árvore, e portanto vai interferir também com o resultado das operações de subdivisão dos nós para manter o balanceamento. Existem algumas técnicas para tentar otimizar este comportamento da árvore-R, mas sempre com algum custo adicional em termos de processamento.

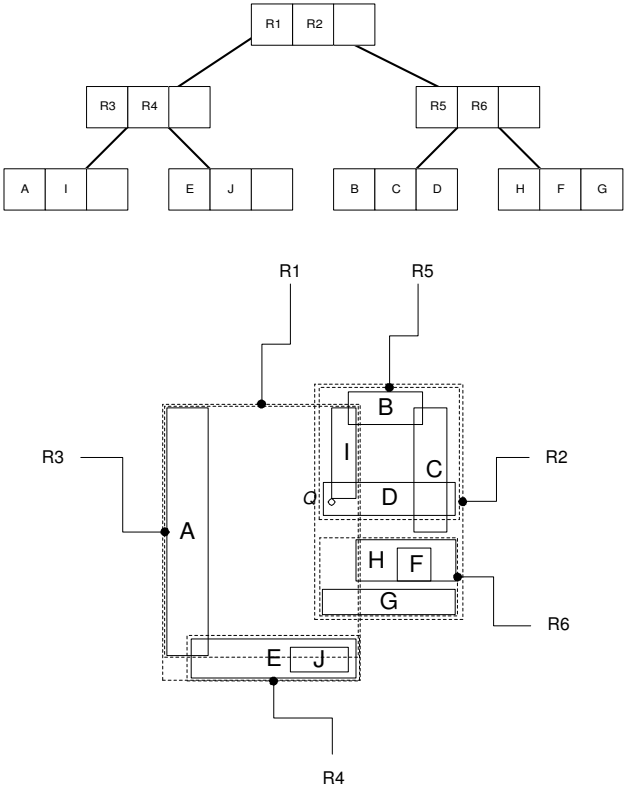

Figura 6.17 – Árvore-R e retângulos correspondentes.

Pesquisas na árvore-R são relativamente simples de serem executadas. O único problema é que um grande número de nós pode ter que ser examinado, pois um retângulo pode estar contido em vários outros, mesmo que o objeto esteja contido em apenas um nó folha. Por exemplo, na Figura 6.17 a identificação do retângulo que contém o ponto *Q* deverá varrer a árvore nível a nível. Inicialmente, inspecionamos a raiz e constatamos que *Q* pode estar tanto em *R1* quanto em *R2*, e portanto devemos prosseguir a pesquisa em ambas as direções no segundo nível. Pesquisando os filhos de *R1*, verificamos que *Q* só pode estar contido em *R3*. Continuando a pesquisa, não chegamos a conclusão alguma, uma vez que *Q* está dentro do retângulo *D*, que tem apenas uma parte dentro de *R3*, e não é acessado por ele. Continuando a pesquisa a partir de *R2*,

verificamos que *Q* somente poderia estar dentro de *R5*. Pesquisando R5, encontramos *D*, o único MBR que contém *Q* efetivamente. Observe-se, no entanto, que *Q* pode não estar dentro do *objeto* cujo MBR é *D*: esta constatação depende ainda de um teste adicional, que verificará se o ponto pertence ao polígono do objeto, agora sim levando em consideração a geometria deste.

Existem diversas variações das árvores-R, cada uma tentando aperfeiçoar um aspecto diferente. No entanto, muitas vezes estas variações introduzem desvantagens, ou uma maior complexidade de implementação, fazendo com que a árvore-R original acabe sendo a opção mais usual.

6.8 Leituras suplementares

Uma fonte importante de leitura complementar são os trabalhos de Gaede e Günther (1998) e, mais recentemente, Vitter (2001), que apresentam uma revisão detalhada da maioria dos índices existentes.

Há inúmeras variantes de R-trees, sendo a R*-tree a mais popular (Beckmann et al., 1990). Uma comparação entre os métodos de acesso espacial mais comum pode ser encontrada em (Kriegel et al., 1990).

Por fim, as TV-trees (Lin et al., 1994) e as X-trees (Berchtold et al., 1996) são exemplos de métodos de acesso para dados de dimensões superiores a dois.

Referências

BECKMANN, N.; KRIEGEL, H. P.; SCHNEIDER, R.; SEEGER, B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: **ACM SIGMOD**, p. 322-331, 1990.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, v. 18, n. 9, p. 509-517, 1975.

BERCHTOLD, S.; KEIM, D. A.; KREIGEL, H.-P. "The X-Tree: An Index Structure for High-Dimensional Data". In Proc. 22nd Int. Conf. on Very Large Databases (VLDB'96), 1996.

COMER, D. The ubiquitous B-Tree. **ACM Computing Surveys**, v. 11, n. 2, p. 121-137, 1979.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to algorithms**. E.U.A.: McGraw-Hill, 1990.

FRIEDMAN, J. H.; BENTLEY, J. L.; FINKEL, R. A. An algorithm for finding best matches in logarithmic expected time. **ACM Transactions on Mathematical Software**, v. 3, n. 3, p. 209-226, 1977.

GAEDE, V.; GÜNTHER, O. Multidimensional access methods. **ACM Computing Surveys**, v. 30, p. 170-231, 1998.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Implementação de sistemas de bancos de Dados**. Rio de Janeiro: Campus, 2001.

GUTTMAN, A. R-Trees: A Dynamic Index Structure for Spatial Searching. In: Annual Meeting ACM SIGMOD. Boston, MA, 1984. p. 47-57.

KRIEGEL, H.-P.; SCHIWIETZ, M.; SCHNEIDER, R.; SEEGER, B. Performance comparison of point and spatial access methods. **Proceedings of the first symposium on Design and implementation of large spatial databases**. Santa Barbara, California, United States. p. 89 – 114, 1990 .

LIN, K.-I.; JAGADISH, H. V.; FALOUTSOS, C. "The TV-tree an index structure for highdimensional data". VLDB Journal: Very Large Data Bases, v.3, n.4, p.517–542, 1994.

NIEVERGELT, J.; HINTERBERGER, H.; SEVCIK, K. The GRID FILE: An Adaptable, Symmetric Multi-Key File Structure. **ACM Transactions on Database Systems (TODS)**, v. 9, n.1, p. 38-71, 1984.

ROBINSON, J. T. Physical storage structures: The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In: 1981 ACM

SIGMOD international conference on Management of data. ACM Press, Washington, 1981. p.

SAMET, H., 1984. The Quadtree and Related Hierarchical Data Structures. **ACM Computing Surveys**, v. 16, p. 187-260.

VITTER, J.S. External Memory Algorithms and Data Structures. ACM **Computing Surveys**, v.33, n. 2, p. 209-271, 2001.