GRID-tree

Ou G-tree

Grupo:
Bernardo Sant'Anna Costa
Bruno de Melo Rangel
Henrique Lauar
Juliana Aparecida Borges
Lavínia Fonseca Pereira
Luísa Ribeiro Notaro
Pedro Parentoni de Almeida
Rafael Stylita Duarte

1. Introdução à Grid-tree

A G-tree (ou Grid-tree) é uma estrutura de dados proposta por Kumar em 1994, que seu objetivo é organizar dados espaciais (ou multidimensionais).

Por exemplo: “Busque todos os empregados que estão **entre 30 e 35 anos** e ganham um **salário entre 100 mil e 120 mil**”

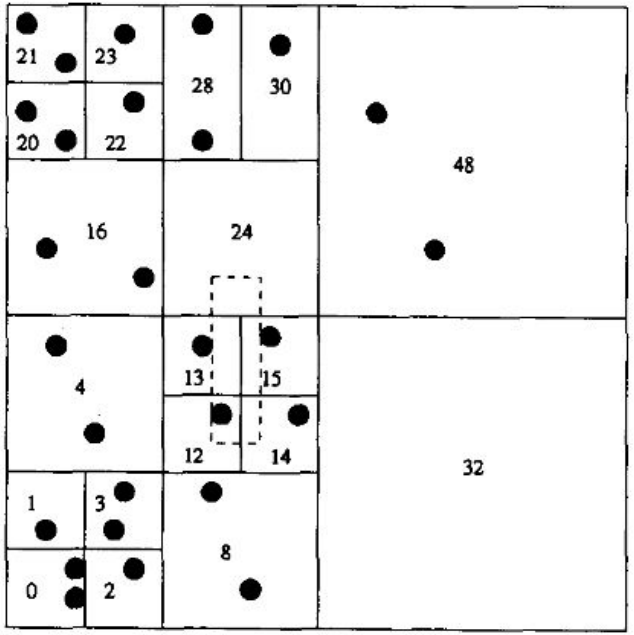
Uma solução seria manter uma árvore-B onde a idade e o salário são usados como chave, e fazendo várias pesquisas para retornar todos os dados nesse intervalo.

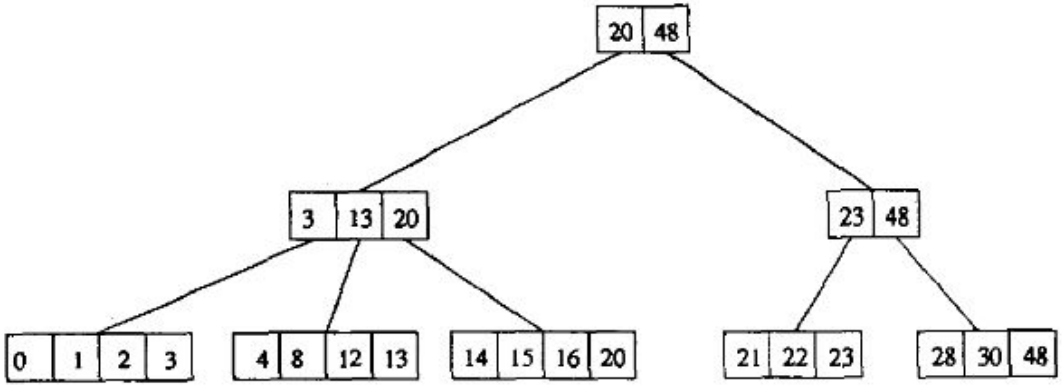
A Grid-tree se adapta bem com frequência alta de inserções e remoções, e a distribuição não uniforme dos dados.

É possível fazer também a pesquisa por intervalo eficientemente nessa estrutura.

1. Introdução à Grid-tree

Essa estrutura de dados combina as características da **grid** e da **Árvore-B** (ou Árvore-B*?)





2. Motivação para a Grid-tree

Em cenários de dados multidimensionais, como **coordenadas espaciais** ou **dados de imagens**, as estruturas tradicionais como árvores-B e árvores KD podem se tornar ineficientes.

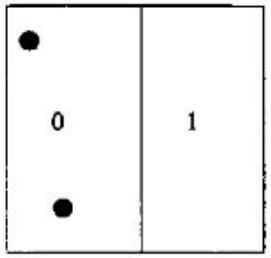
A G-tree foi projetada para resolver algumas dessas limitações, oferecendo uma maneira eficaz de lidar com a inserção, remoção e consulta por intervalo.

Outras estruturas, como a *árvore-BD*, *árvore-KDB* e *árvore-zkdb*, têm desvantagens em termos de eficiência de armazenamento ou complexidade de divisão. A G-tree visa otimizar esses processos, ajustando-se bem a dados **distribuídos de maneira não uniforme** e em **diversas dimensões**.

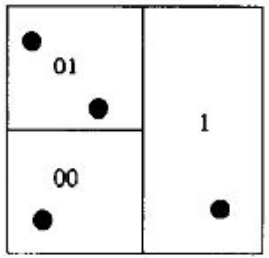
3. Estrutura da Grid-tree

A G-tree é organizada de maneira hierárquica, com duas partes principais:

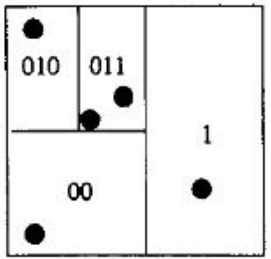
- Partições (buckets):
  - O espaço de dados é dividido em partições, onde cada partição contém um conjunto de pontos de dados.



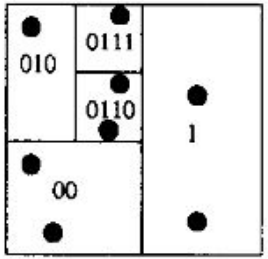
(a)



(b)



(c)

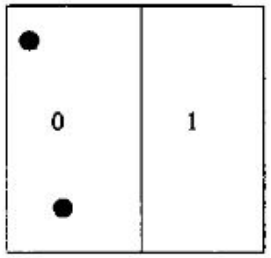


(d)

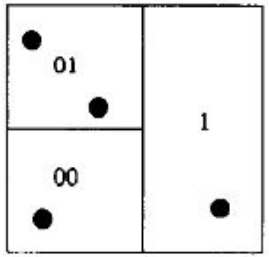
3. Estrutura da Grid-tree

A G-tree é organizada de maneira hierárquica, com duas partes principais:

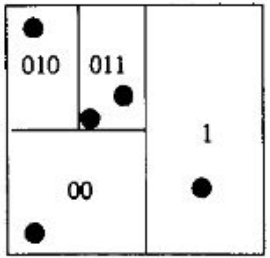
- Partições (buckets):
  - As partições têm uma capacidade máxima, definida por um parâmetro ***max-entries***, que limita o número de pontos de dados em cada partição.



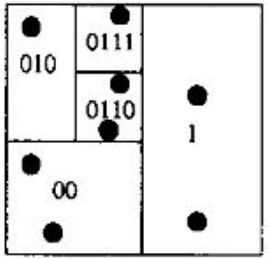
(a)



(b)



(c)

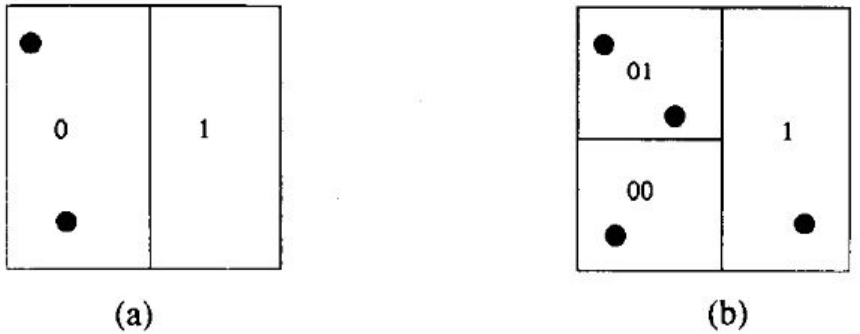


(d)

3. Estrutura da Grid-tree

A G-tree é organizada de maneira hierárquica, com duas partes principais:

- Partições (buckets):
  - Quando uma partição atinge o limite de capacidade, ela é subdividida em duas novas partições menores.

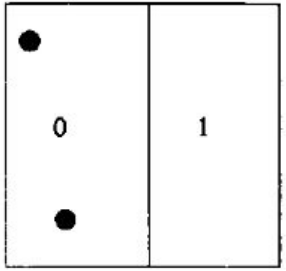


1. A partição 0 foi dividida em duas.
2. As novas partições recebem os bits da partição pai (0).
3. A de baixo recebe mais um bit 0 e a de cima recebe um bit 1

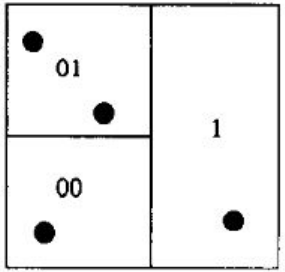
3. Estrutura da Grid-tree

A G-tree é organizada de maneira hierárquica, com duas partes principais:

- Partições (buckets):
  - As partições são identificadas em binário. Neste caso consideramos uma Grid-tree com partições de até **6 bits**.



(a)



(b)

1. As partições com 1 bit recebem cinco 0's à direita

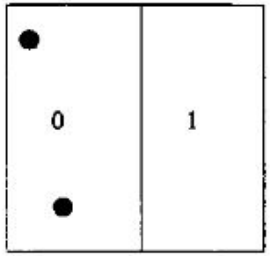
2. As partições com 2 bits recebem quatro 0's à direita, assim por diante

3. A partição 1 seria 100000 = 32

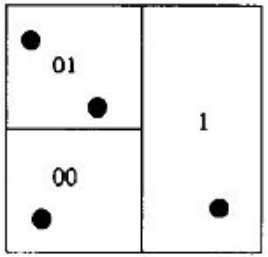
3. Estrutura da Grid-tree

A G-tree é organizada de maneira hierárquica, com duas partes principais:

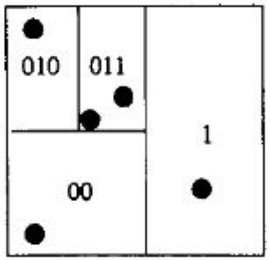
- Partições (buckets):
  - A divisão acontece sempre de maneira alternada entre os eixos X e Y (horizontal e vertical)



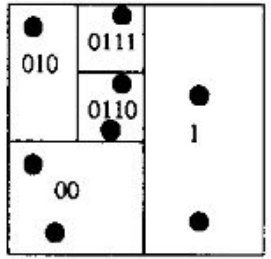
(a)



(b)



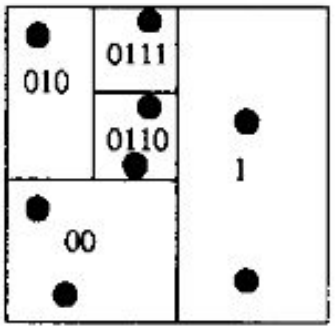
(c)



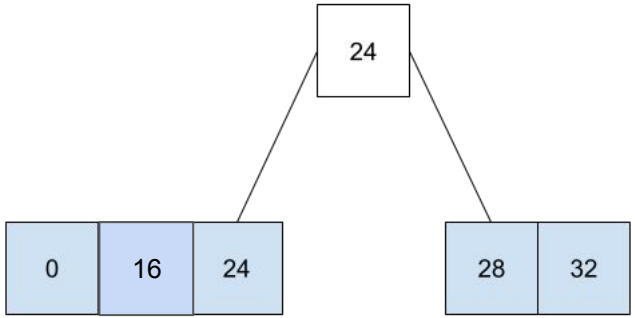
(d)

3. Estrutura da Grid-tree

- Árvore B (B-tree):
  - As partições são organizadas em uma **única árvore B ou árvore B***, que ajuda a manter a ordenação e facilita operações eficientes de busca e recuperação de partições.



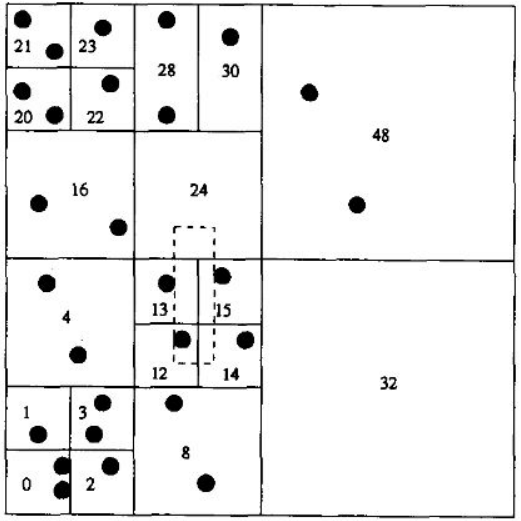
(d)

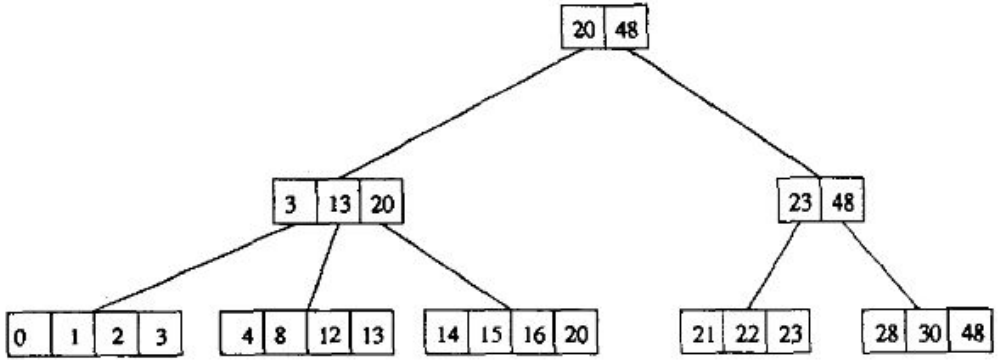


Árvore B* para a figura (d)

3. Estrutura da Grid-tree

- Árvore B ou Árvore B*:
  - A árvore B ou B* armazena as referências às partições. A numeração das partições segue uma ordem binária, refletindo a posição relativa das partições no espaço de dados multidimensional.





3. Estrutura da Grid-tree

Exemplo:

- Inicialmente, o espaço de dados inteiro é tratado como uma única partição.
- Conforme novos dados são inseridos, a partição original pode ser dividida ao longo de uma dimensão (por exemplo, o eixo X), criando duas subpartições.
- Se essas subpartições também ficarem cheias, o processo continua, alternando entre as dimensões (por exemplo, eixo Y).
- A árvore B ou B* gerencia essas partições de maneira eficiente, mantendo a árvore balanceada e garantindo que as operações de busca sejam rápidas.

4. Operações na G-tree

Agora que você entende a estrutura básica da G-tree, vamos falar sobre como ela executa suas operações principais:

- ❏ Inserção
- ❏ Remoção
- ❏ Consulta por Intervalo (Range Queries)

4.1 Inserção

A inserção de um ponto multidimensional na G-tree segue os seguintes passos:

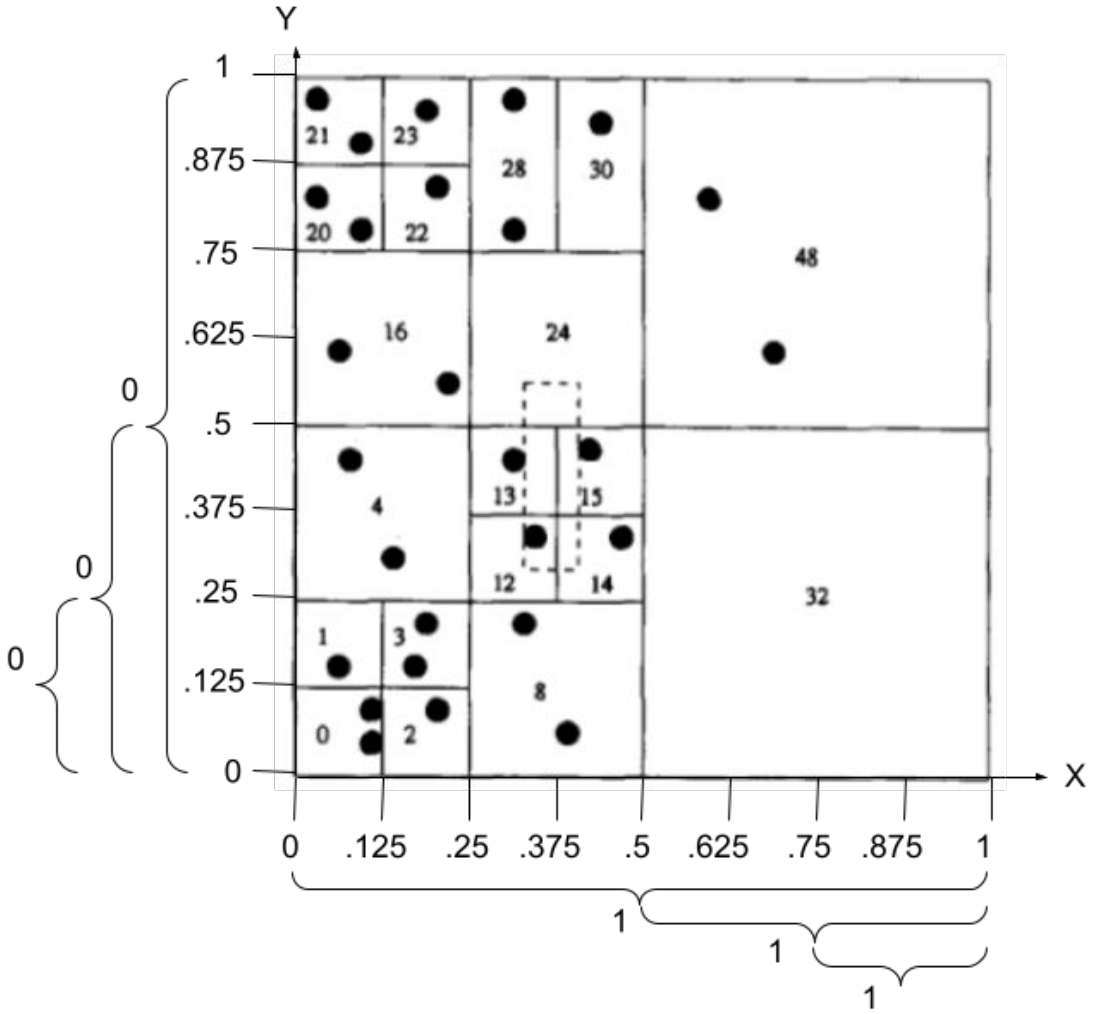
Exemplo, inserindo o ponto (0.9, 0.1)

1. Identificação da partição: O sistema localiza a menor partição que contém o intervalo de coordenadas onde o ponto deve ser inserido.
2. Verificação de capacidade: Se a partição tiver espaço disponível (menos que max-entries), o ponto é simplesmente adicionado.

4.1 Inserção

Inserindo o ponto *p*(x:0.9, y:0.1)

101010 = 42 (decimal)



4.1 Inserção

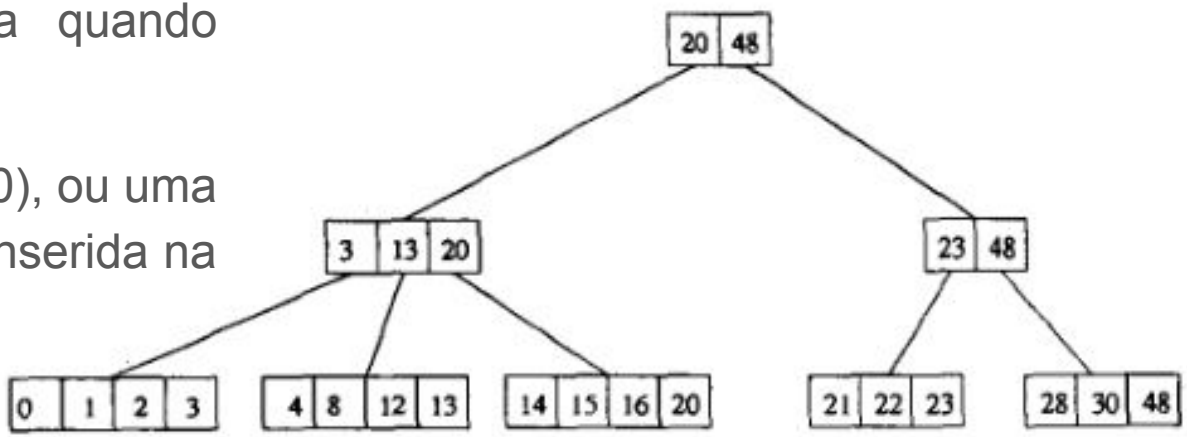
Inserindo o ponto *p* (x:0.9, y:0.1)

101010 = 42 (decimal)

É feita uma pesquisa na árvore pela partição 42, a busca falha quando encontra a partição 48

Ou seja, a partição 42(101010), ou uma partição **ancestral** deve ser inserida na árvore.

A maior partição **ancestral** de 42 é a partição 10 (32). Ela é inserida na árvore, e o ponto *p* pertence a 32.



4.1 Inserção

1. Subdivisão de partições: Se a partição já estiver cheia, ela é dividida em duas novas partições menores. A divisão acontece ao longo de uma das dimensões (X, Y, etc.). Essa divisão ocorre **alternadamente** entre as dimensões para **manter um equilíbrio espacial**.

2. Redistribuição: Os pontos são redistribuídos entre as novas partições com base nas coordenadas.

4.1 Inserção

Outro caso de inserção, considerando o ponto *p*(0.1, 0.6)

A menor partição que o ponto pertence é a 010000 = 16

Busca na árvore a partição 16, mas está cheia, então a partição 0100 é dividida nas partições 01000 e 01001.

Os pontos que já existiam precisam ser realocados para a nova partição, e o novo ponto *p* é inserido na partição 01000.

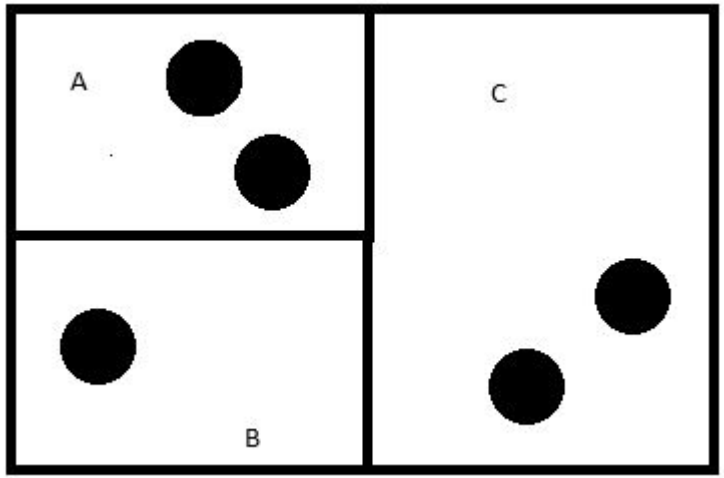
4.2 Remoção

A remoção de um ponto segue uma abordagem semelhante à inserção:

- ❏ Identificação da partição: O sistema localiza a partição onde o ponto está armazenado.
- ❏ Remoção do ponto: O ponto é removido da partição.
- ❏ Mesclagem de partições: Se após a remoção, o número de pontos em duas partições complementares for menor ou igual ao limite ***max-entries***, as partições são mescladas novamente para formar uma única partição.

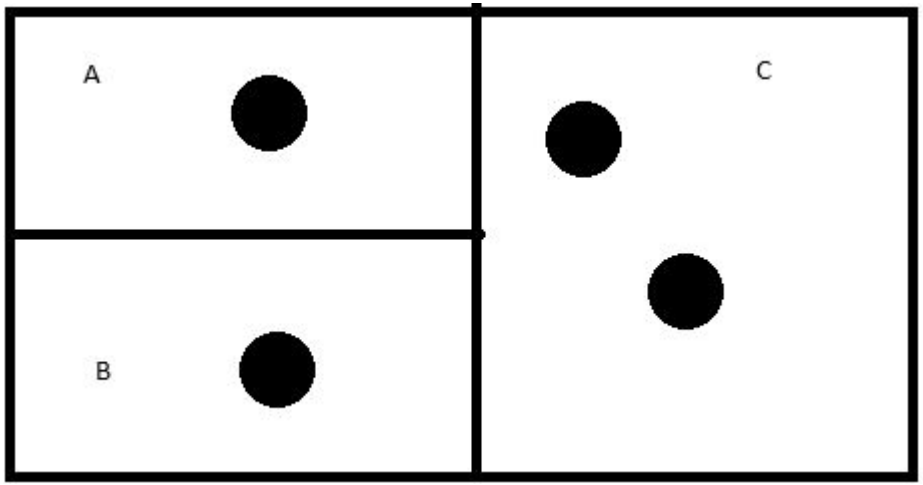
4.2 Remoção

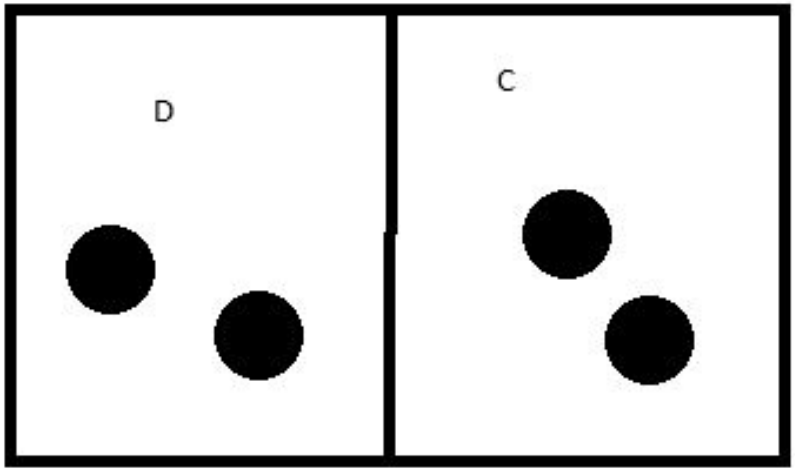
Por exemplo: Supondo o seguinte caso onde ***max-entries*** = 2.



4.2 Remoção

Ao se remover um dos itens de A são feitas comparações com B e C para verificar a possibilidade de mesclar os blocos. Como B possui apenas 1 elemento e ***max-entries*** é igual a 2 é possível fazer a fusão dos dois blocos e gerando no lugar o bloco D.





4.3 Consulta por Intervalo (Range Queries)

As consultas por intervalo são operações muito comuns em bancos de dados multidimensionais, como consultas espaciais para encontrar todos os pontos dentro de uma determinada região. Na G-tree:

- Busca por partições relevantes: A árvore B é usada para identificar as partições que se sobrepõem à região de consulta.
- Verificação de pontos: Para cada partição, os pontos são verificados. Se a partição inteira está dentro do intervalo de consulta, todos os seus pontos são retornados; se apenas parte da partição está dentro do intervalo, cada ponto é verificado individualmente.

5. Vantagens da G-tree

Agora que entendemos como a G-tree funciona, é importante destacar suas principais vantagens:

- **Eficiência Espacial:** Como a G-tree só armazena partições não vazias, ela economiza espaço de armazenamento, especialmente em cenários onde os dados são esparsos.
- **Balanceamento Automático:** A subdivisão das partições e o uso da árvore B garantem que a estrutura esteja balanceada, permitindo uma eficiência consistente em operações de inserção, deleção e consulta.

5. Vantagens da G-tree

- **Ajuste Dinâmico às Distribuições de Dados:** A G-tree se ajusta bem a dados distribuídos de maneira não uniforme, dividindo o espaço de forma adaptativa conforme as inserções ocorrem.
- **Fácil Manipulação com Algoritmos:** Muitos algoritmos, especialmente em gráficos e computação de caminhos, operam sobre grids de maneira eficiente. Algoritmos como busca em largura (BFS), busca em profundidade (DFS), e algoritmos de caminho mínimo (Dijkstra) são frequentemente aplicados em grids para encontrar caminhos ou verificar conexões entre células.
- **Simplicidade:** A G-tree é relativamente simples de implementar em comparação com outras estruturas mais complexas, como KDB-trees ou zkdb-trees, que têm regras de divisão mais complicadas.

6. Comparação com Outras Estruturas

6.1. BD-Tree vs. Grid

- BD-Tree: Divide o espaço de forma adaptativa para lidar com alta densidade de dados.
- Desvantagem: Complexidade aumentada e ineficácia em baixa densidade
- Vantagem da Grid: Estrutura simples e estática, fácil de implementar e gerenciar em cenários uniformes.

6.2. k-d Tree vs. Grid

- k-d Tree: Divide o espaço em partições k-dimensionais para consultas eficientes.
- Desvantagem: Desempenho degrada em alta dimensionalidade devido à complexidade das divisões.
- Vantagem da Grid: Trata cada célula de forma independente, mantendo simplicidade e rapidez independentemente da dimensionalidade.

6.3. z-kdb Tree vs. Grid

- z-kdb Tree: Utiliza mapeamento Z-order para preservar a proximidade espacial em consultas multidimensionais.
- Desvantagem: Implementação complexa e manutenção exigente, especialmente para grandes conjuntos de dados.
- Vantagem da Grid: Implementação direta e simples, sem necessidade de transformações complexas, adequada para aplicações sem requisitos específicos de otimização de localidade.

7. Exemplos e Aplicações

Para visualizar a aplicação da G-tree, imagine que você está desenvolvendo um sistema para armazenar coordenadas geoespaciais de locais em uma cidade (longitude, latitude). À medida que novas coordenadas são inseridas, a G-tree cria partições eficientes e flexíveis para organizar os dados. Quando você faz uma consulta para encontrar todos os locais dentro de um determinado bairro, a G-tree rapidamente identifica as partições relevantes e retorna os resultados, sem precisar vasculhar todos os dados.

8. Conclusão

A G-tree é uma solução inovadora para organizar dados multidimensionais. Sua combinação de partições dinâmicas e gerenciamento eficiente com uma árvore B a torna uma estrutura ideal para cenários com grandes volumes de dados espaciais ou multidimensionais. Ela se destaca pela simplicidade, eficiência de armazenamento e adaptação a distribuições de dados complexas.

Esses são os conceitos principais sobre a G-tree, que é uma ferramenta útil para manipular dados espaciais e multidimensionais de forma eficaz!

Referências

- Volker Gaede and Oliver Günther. 1998. Multidimensional access methods. ACM Comput. Surv. 30, 2 (June 1998), 170–231. https://doi.org/10.1145/280277.280279
- Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, Lizhu Zhou, and Zhiguo Gong. 2015. G-tree: An efficient and scalable index for spatial search on road networks. IEEE Transactions on Knowledge and Data Engineering 27, 8 (2015), 2175–2189.