Estruturas de Dados Espaciais

Processamento Geométrico

Bancos de Dados Espaciais

Sistemas de Informações Geográficos (GIS)

Claudio Esperança

Objetivos do Curso

- Estruturas de dados para armazenamento, busca e ordenação de dados espaciais
- Algoritmos geométricos fundamentais
- Arquitetura de bancos de dados espaciais
- Integração com bancos de dados convencionais
- Processamento de consultas (queries) envolvendo dados espaciais e não-espaciais

Dados Espaciais x Dados Escalares

- Multidimensionais x Unidimensionais
- Noção de Forma x pontos ou tuplas
- Ordenação parcial x Ordenação total
- Relações geométricas x Relações sobre grandeza
- Frequentemente, os dois tipos são combinados em:
  - Sistemas de Informação Geográficos
  - CAD
  - Computação Gráfica

Espaço de dados

- Qualquer tipo de dado supõe um espaço onde ele está imerso
- Modelagem de dados requer que se escolha um espaço apropriado
- Frequentemente, mais de uma opção é possível
- Exemplo: Cidade
  - Espaço de cadeias de caracteres
  - Código numérico (ex. CEP)
  - Ponto do planisfério (Latitude e Longitude)
  - Conjunto de pontos (ex. delimitado por um polígono)
- Cada espaço é mais conveniente para um ou outro tipo de processamento

Dimensão

- Qual a dimensão do espaço de cidades?
  - Como cadeia de caracteres: 1 (existe um mapeamento 1 para 1 entre o conjunto de cadeias e o conjunto dos numeros inteiros)
  - Como ponto no planisfério: 2
  - Como conjunto de pontos delimitado por um polígono: 2
- Qual a dimensão do dado cidade?
  - Como cadeia de caracteres: 0
  - Como ponto no planisfério: 0
  - Como conjunto de pontos delimitado por um polígono: 2

Dimensão (cont.)

- Dados escalares (não espaciais) são modelados como pontos em um espaço unidimensional
- Dados espaciais são modelados como pontos ou conjuntos de pontos em espaço multidimensional
- Entre os dois: Conjuntos de pontos em espaço unidimensional (ex., intervalos, séries temporais)

Relações entre dado e espaço

- Localização
  - Existe uma cidade chamada “São Paulo” ?
  - Existe uma cidade em 39°29’30” S, 65°50’20” W ?
- Vizinhança
  - Qual a cidade com nome subsequente a “São Paulo”?
  - Qual a cidade mais próxima de São Paulo?
    - Noção de métrica
- Extensão (Dados Espaciais)
  - Qual o perímetro de Säo Paulo?
  - Qual a área de São Paulo?

Uso de ordenação

- Dados escalares
  - É possível estabelecer uma ordem total
  - Ordenação facilita operações de localização e vizinhança
- Dados espaciais
  - É impossível estabelecer uma ordem total sem romper com relações de vizinhança
  - A imposição de uma ordem total é conhecida como *linearização do espaço*.Exemplo: ordenar um conjunto de pontos lexicograficamente
  - Ordenação parcial, no entanto, pode facilitar diversas operações
- Estruturas de dados espelham ordenação

Estruturas de dados para dados escalares

- Visam essencialmente facilitar operações de localização e de vizinhança
- Exemplos:
  - Tabelas organizadas por cálculo de endereço *(Hash Tables)*
    - Usadas em localização de dados
      - Caso médio: O(1)
      - Pior caso: O(n)
    - Podem ser baseadas em memória ou disco
  - Árvores binárias balanceadas
    - Localização de dados: O(log n)
    - Vizinhança: O(log n)
    - Primariamente baseadas em memória principal

Estruturas de dados para dados escalares (cont.)

- Árvores B e suas variantes
  - Localização de dados: O(log n)
  - Vizinhança: O(log n)
  - Otimizadas para utilização em memória secundária (disco)
  - Asseguram alta taxa de utilização (garantidamente > 50%)

Idéia geral de estruturas de dados espaciais

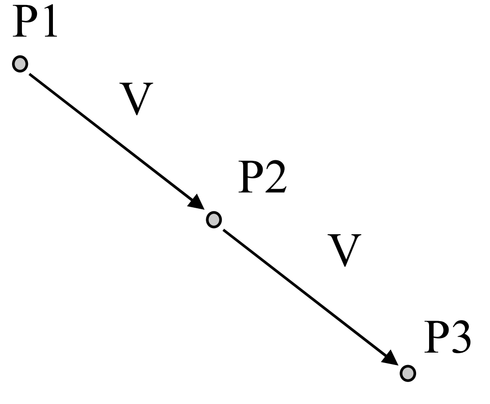
- Precisam suportar grande número de operações
- Não existe estrutura de dados espacial que garantidamente seja eficiente para atender todos os tipos de operararação
- Aplicações em bancos de dados espaciais / GIS:
  - Utiliza-se estruturas de dados gerais que têm eficiencia razoável no caso médio. Ex.: PMR-quadtrees, Grid files, R-trees e suas variantes
- Aplicações em CAD, Computação gráfica:
  - Frequentemente estruturas de dados gerais dão bons resultados
  - Em casos especificos, estruturas de dados especializadas podem ser indicadas.: Ex.: Diagramas de Voronoi

Representação de dados geométricos 2D

- Valores escalares
  - Inteiros de precisão fixa
    - representação exata
    - problema de discretização (quantização)
  - Inteiros de precisão variável
    - utilizam alocação dinâmica de memória
    - manipulação através de biblioteca (lento)
  - Ponto flutuante (simples / dupla precisão)
    - representação inexata de números reais
    - sujeitos a problemas de precisão
  - Números racionais (fração)
    - Frações (espaço = 2 inteiros de precisão fixa/variável)
    - Manipulação através de biblioteca
    - Problema de unicidade (infinidade de frações c/ mesmo valor)

Representação de dados geométricos 2D (cont.)

- Pontos e vetores
  - 2 valores escalares
  - Não confundir os dois
    - Ponto denota posição
    - Vetor denota deslocamento
    - Usar programação geométrica



P1 + V = P2

P2 - P1 = V

P1 + 2V = P3

Representação de dados geométricos 2D (cont.)

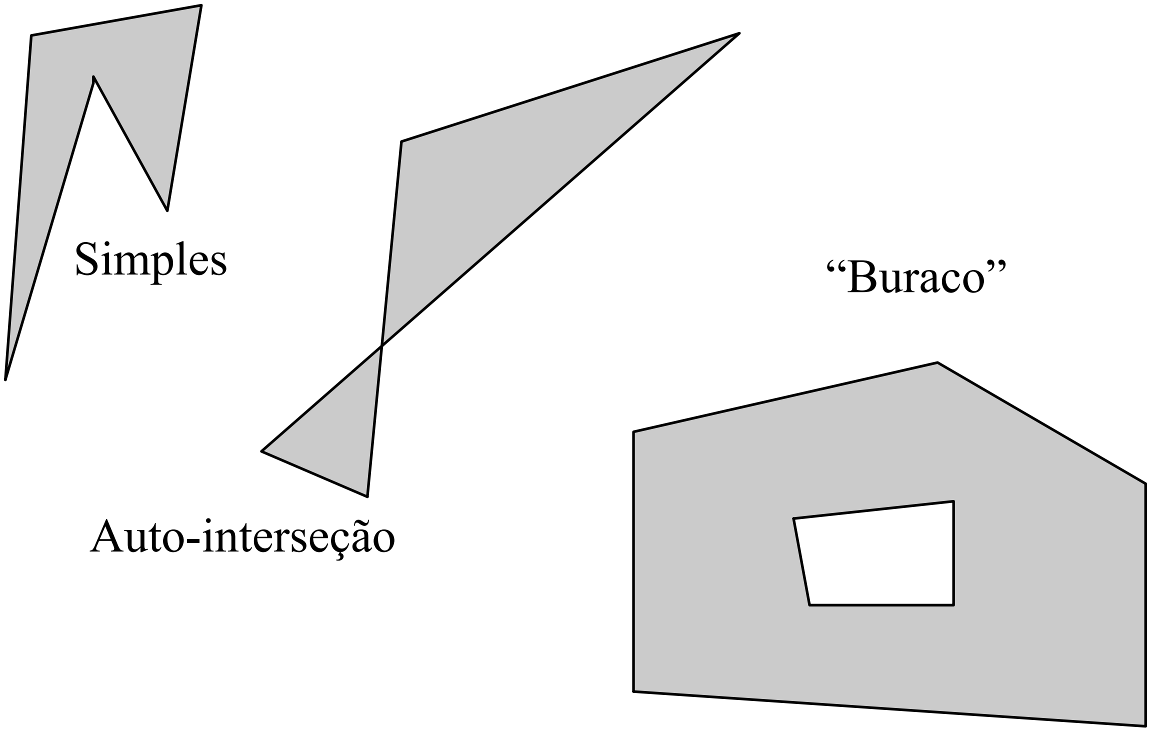
- Retas
  - Representação implícita
    - a x + b y + c = 0
    - 3 valores escalares
    - É possível substituir 1 valor por um *flag,* mas complica a representação
  - Representação paramétrica
    - $(x,y) = P + t\,V$
    - t é o parâmetro que permite "caminhar" sobre a reta
    - 4 valores escalares
    - É possível substituir V por um ângulo, mas complica a representação
  - Conversão entre representações
    - *Exercício*

Representação de dados geométricos 2D (cont.)

- Segmentos de reta
  - Dois pontos (extremidades)
  - Representação paramétrica
    - Assumir intervalo $t = [0,1]$
- Retângulos alinhados com os eixos coordenados
  - Usados extensivamente em estruturas de dados espaciais
    - Partições do espaço
    - Caixas Limitantes (*bounding boxes*)
  - Pontos extremos (min_x, min_y) (max_x, max_y)
    - max_x >= min_x e max_y >= min_y
  - Ponto mínimo e vetor extensão (min_x, min_y)(tam_x,tam_y)
    - tam_x e tam_y >= 0

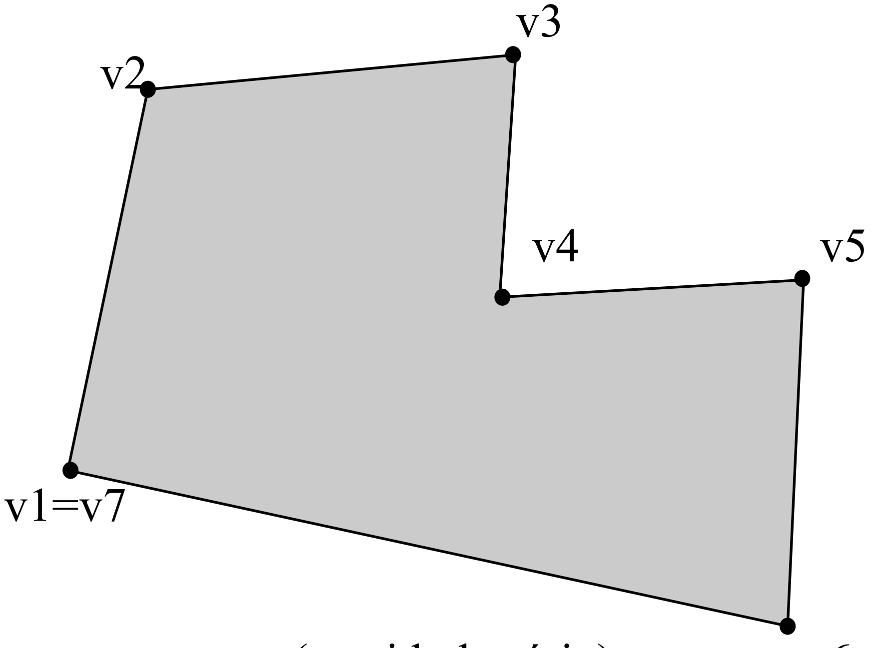
Representação de dados geométricos 2D (cont.)

- Polígonos
  - Usados para aproximar regiões
  - Podem ser simples ou complexos
    - Simples: topologicamente homeomorfos a um disco
    - Complexos: Podem possuir auto-interseções e "buracos"



Representação de dados geométricos 2D (cont.)

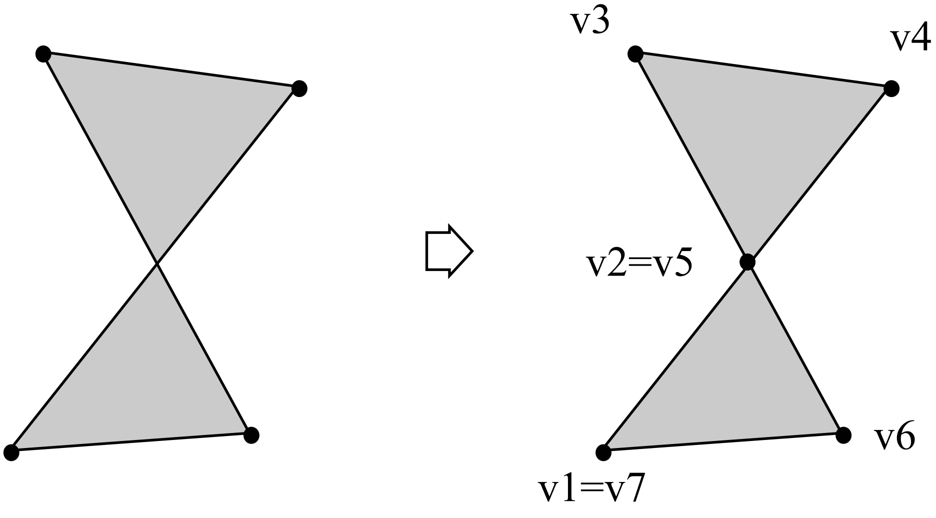
- Polígonos (cont)
  - Lista de vértices (pontos)
    - Muitos algoritmos requerem uma circulação predeterminada (sentido horário ou anti-horário)
    - Importante garantir (implicita ou explicitamente) que primeiro vértice = último vértice

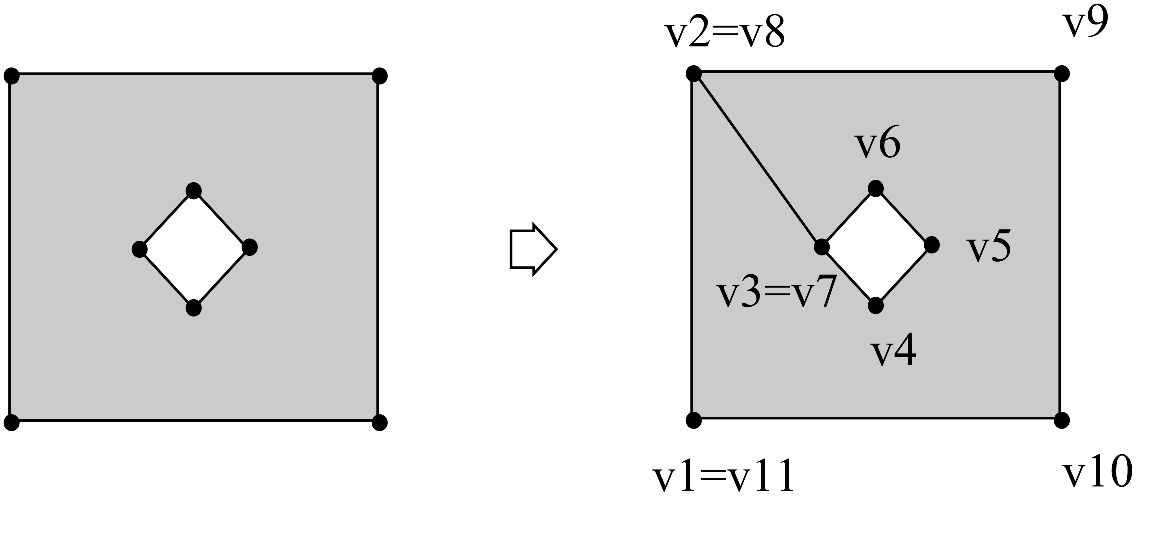


(sentido horário)

Representação de dados geométricos 2D (cont.)

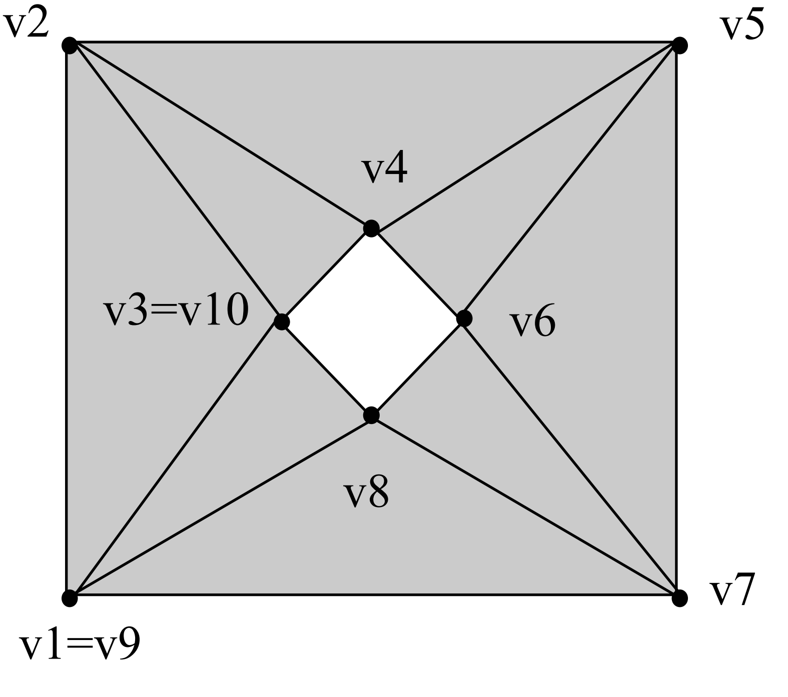
- Polígonos (cont.)
  - Polígonos complexos podem ser representados por listas de vértices inserindo-se novos vértices e/ou arestas





Representação de dados geométricos 2D (cont.)

- Polígonos (cont.)
  - Triangulação
    - Conjunto de triângulos que se interceptam apenas segundo arestas e cuja união é idêntica ao polígono
    - Poliígono c/ N vértices = N-2 triângulos
    - Facilita uma série de algoritmos
    - Polígono pode ser triangulado em O(N)



Operações com dados geométricos

- Sejam *A* e *B* dois dados geométricos (ex.: ponto, segmento de reta, polígono, etc)
- Predicados ($A \times B \rightarrow boolean$)
  - *intercepta(A,B)*: se *A* e *B* têm ao menos um ponto em comum

$$A \cap B \neq \emptyset$$

  - *contém(A,B)*: se *A* contém B

$$A \supseteq B$$

  - *adjacente(A,B)*: se a fronteira de *A* e a fronteira de *B* têm algum ponto em comum, mas o interior de *A* e *B* não se interceptam (*A* e *B* têm que ser conjuntos compactos de dimensão >= 2)

$$\partial A \cap \partial B \neq \emptyset \;\wedge\; \partial A \cap \partial B \supseteq A \cap B$$

  - *próximo(A,B,d)*: se a distância entre *A* e *B* é maior que *d*

$$\delta(A,B) \leq d$$

Operações com dados geométricos (cont.)

- *distância(A,B)* ou δ*(A,B)*
  - depende de uma <u>métrica</u> δ*(a,b)*

$$\delta(A,B) = \min_{a \in A, b \in B} \delta(a,b)$$

  - Uma métrica é uma função que mapeia pares de pontos em números positivos com as propriedades:
    - $\delta(a,b) = 0$ se e somente se $a=b$
    - $\delta(a,b) = \delta(b,a)$
    - $\delta(a,c) \leq \delta(a,b) + \delta(b,c)$
  - métricas mais empregadas:
    - Euclideana:

$$\delta_E(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$$

    - Valor absoluto (ou *Manhattan*)

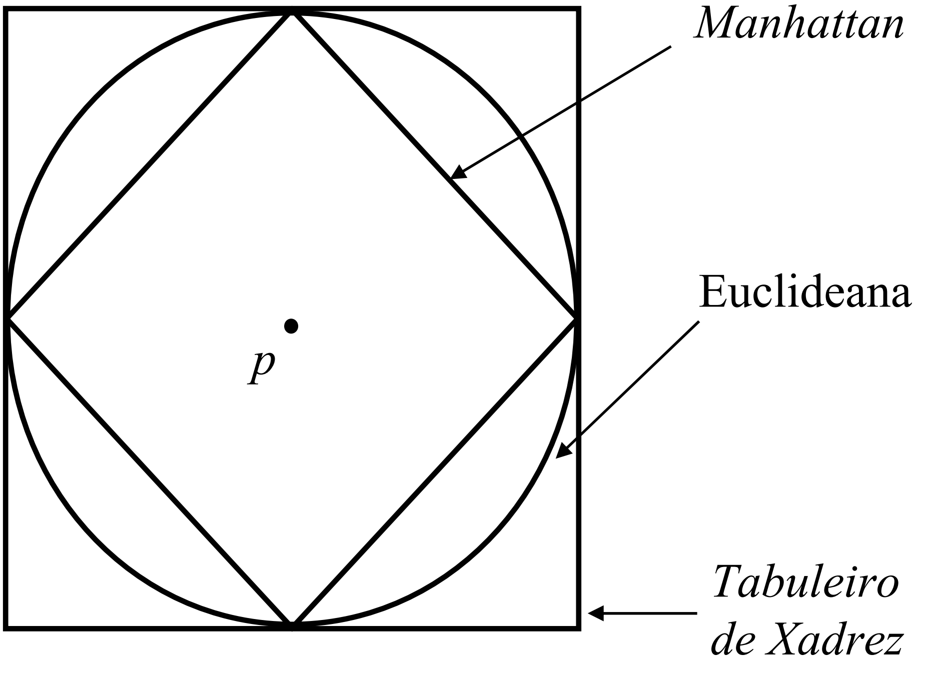
$$\delta_A(a,b) = |a_x - b_x| + |a_y - b_y|$$

    - Valor máximo (ou *Tabuleiro de Xadrez*)

$$\delta_M(a,b) = \max\left\{ |a_x - b_x| \,,\, |a_y - b_y| \right\}$$

Operações envolvendo dados geométricos

- *distância(A,B)* (cont.)
  - lugar geométrico de todos os pontos à mesma distância de um dado ponto *p* é característica da métrica

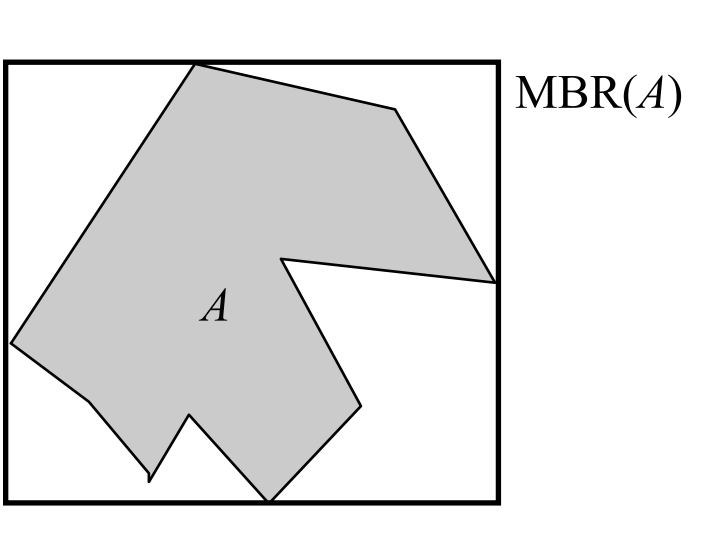


Operações com dados geométricos (cont.)

- Propriedades integrais
  - Perímetro (comprimento da borda)
  - Área
  - Volume (3D)
  - Centro de massa (Centróide)
- Operações morfológicas
  - Dilatação
  - Contração
- Operações booleanas
  - União
  - Interseção
  - Diferença

Caixas Limitantes

- Caixas limitantes (*bounding boxes*) servem como estimativas do lugar geométrico de dados espaciais
- Certas operações podem ser aceleradas através de testes preliminares com caixas limitantes
- O tipo mais comum de caixa limitante é o retângulo com lados alinhados com os eixos coordenados - (*MBR - Minimum Bounding Rectangle*)
- Dado um conjunto A, MBR(A) é um retângulo definido por seus dois vértices extremos, *min* e *max,* tal que, para todo eixo coordenado x
  - $min_x = \min_{a \in A}(a_x)$
  - $max_x = \max_{a \in A}(a_x)$



Propriedades de caixas limitantes

- Interseção
  - $A \cap B \subseteq \text{MBR}(A) \cap \text{MBR}(B)$
  - $\text{MBR}(A) \cap \text{MBR}(B) = \varnothing \Rightarrow A \cap B = \varnothing$
  - $A \cap \text{MBR}(B) = \varnothing \Rightarrow A \cap B = \varnothing$
  - $A \cap B \neq \varnothing \Rightarrow \text{MBR}(A) \cap \text{MBR}(B) \neq \varnothing$
  - $A \cap B \neq \varnothing \Rightarrow A \cap \text{MBR}(B) \neq \varnothing$
- Distância
  - $\delta(A,B) \geq \delta(\text{MBR}(A), \text{MBR}(B))$
  - $\delta(A,B) \leq$

    $$\min_{a \in \text{F}(\text{MBR}(A)),\, b \in \text{F}(\text{MBR}(B))} \{ \delta_{\max}(a,b) \},$$

    onde $\text{F}(\text{MBR}(A))$ e $\text{F}(\text{MBR}(B))$ são faces das caixas limitantes de A e de B e $\delta_{\max}(S,T)$ é a distância máxima entre os conjuntos S e T:

    $$\delta_{\max}(S,T) = \max_{s \in S, t \in T} \{ \delta(s,t) \}$$

    (propriedade ligada ao fato que toda face de uma caixa limitante contém ao menos um ponto do conjunto limitado)

Implementando predicados de interseção

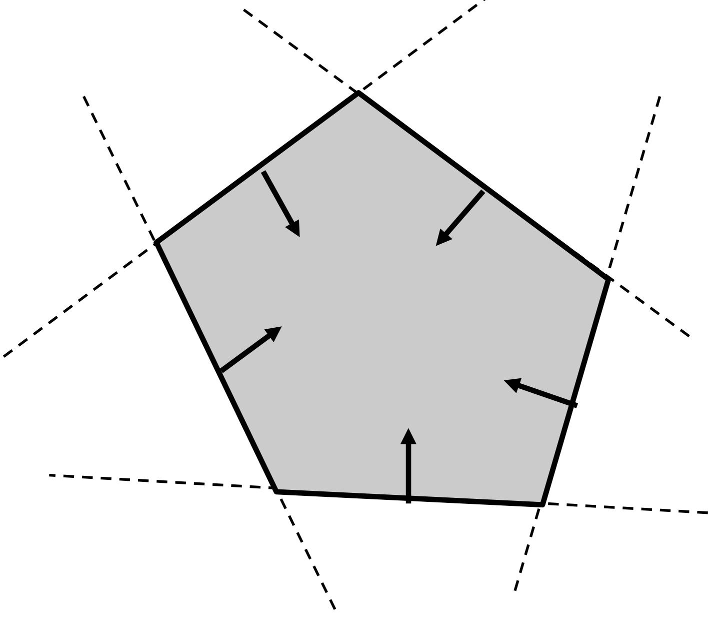
- Regras gerais
  - Usar testes preliminares contra caixas limitantes se esses testes forem mais simples do que os testes de interseção propriamente ditos (ex. testes entre polígonos)
  - Procurar discernir entre interseção com o interior e interseção com a fronteira (predicado *toca*(*A,B*))
  - Usar classificação de ponto contra semi-espaços para diagnóstico precoce de não-interseção
  - Usar aritmética inteira sempre que possível
  - Usar estruturas de dados "complicadas" apenas como último recurso

Testes de interseção com Pontos

- Ponto
  - $P \cap Q \neq \emptyset \;\Leftrightarrow\; P_x = Q_x \wedge P_y = Q_y$
- Reta
  - Seja L dado por a . x + b . y + c = 0
  - $P \cap L \neq \emptyset \;\Leftrightarrow\;$ a . P_x + b . P_y + c = 0
- Segmento de Reta
  - Testar P contra a reta de suporte
  - Seja S dado parametricamente por $Q + tV$, onde $0 \leq t \leq 1$
  - Calcular t_x ou t_y , correspondentes à interseção do segmento com as retas $x = P_x$ ou $y = P_y$
  - t_x ou t_y ? Escolher baseado no maior entre V_x e V_y
- Retângulo alinhado c/ eixos
  - Seja R dado por seus pontos mínimo e máximo
  - $P \cap R \neq \emptyset \;\Leftrightarrow P_x \geq min_x \wedge P_x \leq max_x \wedge$
    $P_y \geq min_y \wedge P_y \leq max_y$
  - Para teste contra o interior de R, substituir “≥“ por “>“ e “≤“ por “<“

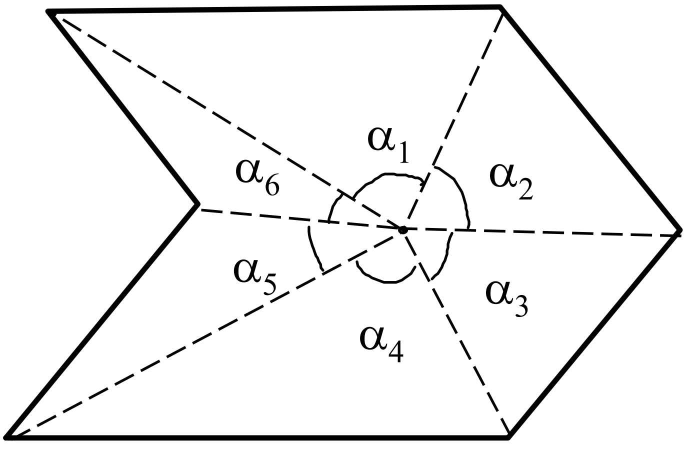
Testes de interseção com Pontos (cont.)

- Polígono convexo
  - Seja G dado por uma lista de vértices $g_1, g_2, \ldots g_N$ enumerados segundo uma circulação horária da fronteira de G
  - Seja a linha de suporte da aresta g_i-g_{i+1} dada por $\mathrm{a}_i \cdot x + \mathrm{b}_i \cdot y + \mathrm{c}_i = 0$, de tal forma que para todos os pontos Q do interior do polígono $\mathrm{a}_i \cdot Q_x + \mathrm{b}_i \cdot Q_y + \mathrm{c}_i > 0$
  - $P \cap R \neq \varnothing \iff \forall i=1 \ldots N,\ \mathrm{a}_i \cdot P_x + \mathrm{b}_i \cdot P_y + \mathrm{c}_i \geq 0$

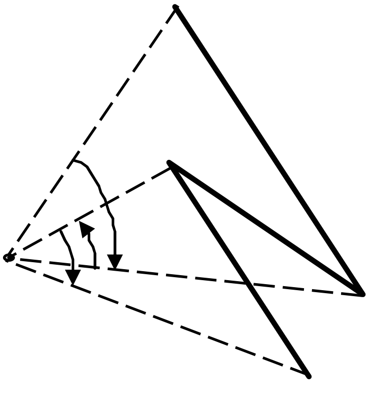


Testes de interseção com Pontos (cont.)

- Polígono qualquer
  - Teste da soma de ângulos
    - A soma dos angulos formados entre *P* e dois vértices consecutivos *gi* e *gi+1* de G deve ser de 360 °

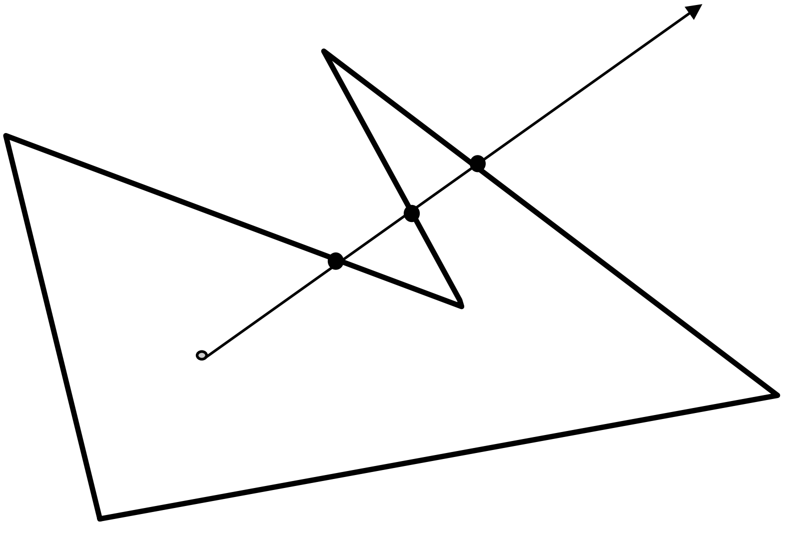


Ângulos tomados no mesmo sentido (sentido contrário) da circulação do polígono são positivos (negativos)



Testes de interseção com Pontos (cont.)

- Polígono qualquer (cont.)
  - Teste de paridade
    - Reta que atravessa o polígono “entra/sai” um número par de vezes
    - Semi-reta ancorada em *P* “sai” vez mais do que “entra”



Testes de interseção com Segmentos de reta

- Seja *S* um segmento de reta
- Interseção com outro segmento de reta *T*
  - Computar o ponto de interseção *P* entre as retas de suporte de *S* e *T*
    - forma implícita ax+by+c=0 normalizada de tal forma que o primeiro coeficiente nao nulo seja igual a 1
    - Se as retas têm a e b idêntico, são paralelas (*S* e *T* não se interceptam)
    - Se as retas têm a, b e c idênticos, são coincidentes (*S* e *T* se interceptam se e somente se MBR(*S*) e MBT(*T*) se interceptam)
  - Testar interseção de *P* com *S* e *T*

Testes de interseção com Segmentos de reta

- Retângulo *R* alinhado com eixos coordenados
  - Sejam *A* e *B* os dois pontos extremos de *S*
  - Classificar *A* e *B* para determinar em qual das 9 regiões relativas a *R* eles se encontram:

1	2	3
4	0	5
6	7	8

B \ *A*	0	1	2	3	4	5	6	7	8
0	●	●	●	●	●	●	●	●	●
1	●	○	○	○	○		○		
2	●	○	○	○				●	
3	●	○	○	○		○			○
4	●	○			○	●	○		
5	●			○	●	○			○
6	●	○			○		○	○	○
7	●		●				○	○	○
8	●			○		○	○	○	○

○ $S \cap R = \emptyset$

● $S \cap R \neq \emptyset$

Testes de interseção com Segmentos de reta (cont)

  - Se a classificação das extremidades de *S* com relação a *R* for insuficiente para determinar se *S* intercepta *R*, testar a interseção de *S* com cada face de *R*
- Polígono
  - Testar *S* com cada aresta do polígono *G*
  - Se *S* não intercepta a fronteira de *G*, então *S* intercepta *G* se e somente se algum ponto de *S* está dentro de *G*
    - Teste de interseção de uma das extremidades de *S* contra *G*

Testes de interseção com Retângulos

- Teste de *R* contra outro retângulo *Q*
  - *R* intercepta *Q* se e somente se as projeções de *R* e *Q* em ambos os eixos coordenados se interceptam
    - $\max\{\min_{Rx}, \min_{Qx}\} \geq \min\{\max_{Rx}, \max_{Qx}\}$
    - $\max\{\min_{Ry}, \min_{Qy}\} \geq \min\{\max_{Ry}, \max_{Qy}\}$
- Teste de *R* contra polígono *G*
  - Testar todas as arestas de *G* contra *R*
  - Se a fronteira de *G* não intercepta *R*, então há duas hipóteses
    - *R* está contido em *G*
    - *R* não intercepta *G*
  - Testar um ponto de *R* contra *G*

Teste de interseção entre dois polígonos

- Algoritmo "ingênuo" tem péssimo desempenho, pois envolve testar cada aresta de um polígono (*G*) contra todas as arestas do outro polígono (*H*)
  - $O\ (|G|\ .\ |H|)$
- Representações alternativas para polígonos podem facilitar a tarefa
  - Exemplo: Triangulações
  - Exemplo: BSP-trees
- Algoritmos mais eficientes são conhecidos na literatura de Geometria Computacional
  - Exemplo: Algoritmo de varredura

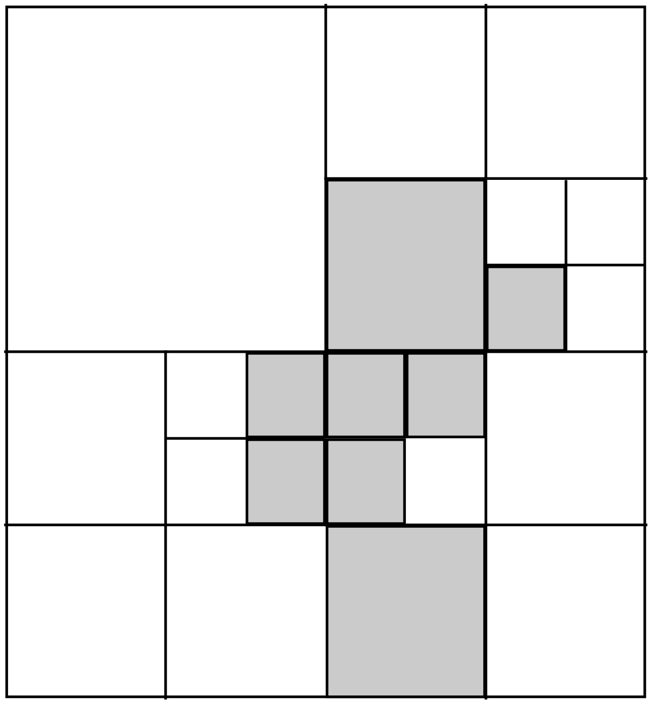
Métodos de decomposição do espaço

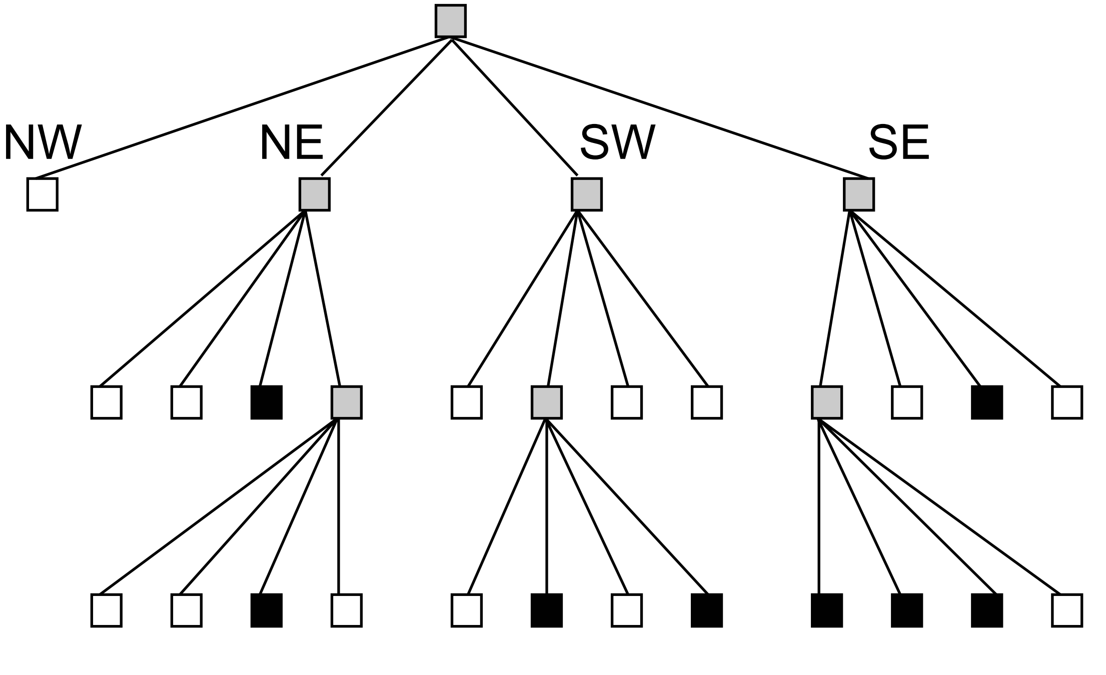
- Visam lidar com a complexidade de distribuições espaciais dividindo o espaço em subregiões. Cada subregião é menos complexa que o todo
- Contraposição com métodos de decomposição ou agrupamento de objetos com base na proximidade entre esses objetos
- Classificação quanto a hierarquia
  - Metodos hierárquicos (Quadtrees, k-d-trees)
  - Métodos não hierárquicos (grades regulares, Grid File, EXCELL)
- Classificação quanto ao posicionamento das subdivisões
  - Regular (divisão do espaço em partes iguais): bintrees, quadtrees de região,
  - Não regular ou adaptativos (divisão não necessariamente resulta em subdivisões idênticas): k-d trees, quadtrees de pontos, BSP-trees
- Classificação quanto a forma das subdivisões
  - Quadriláteros alinhados com os eixos (quadtrees, bintrees, k-d trees)
  - Polígonos convexos quaisquer (BSP-trees)
  - Outros (triangulações, sector trees, etc)

Quadtrees de região

- Utilizado inicialmente para imagens binárias (preto e branco)
- Assume uma imagem quadrada de lado igual a uma potência de 2 ($2^n \times 2^n$)
- Cada divisão de um quadrado $2^i \times 2^i$ resulta em 4 quadrados $2^{i-1} \times 2^{i-1}$
- Critério de subdivisão: dividir enquanto subregião contiver pixels brancos e pretos; não subdividir se subregião for uniformemente preta ou branca
- Nós folha são rotulados brancos e pretos; nós internos são chamados de “cinza”

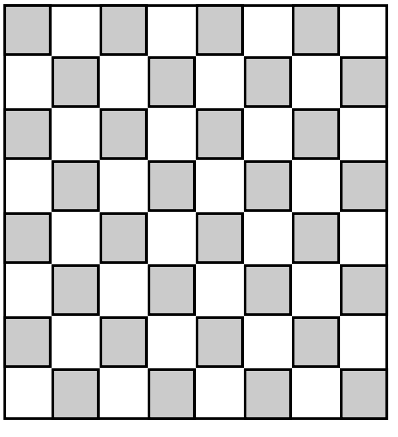
Quadtrees de Região (cont.)





Propriedades da Quadtree de Região

- Espaço de armazenamento:
  - Se numa árvore completa há K nós-folha, há um total de
    $K + K/4 + K/16 + \ldots = \text{floor}(4K/3)$
  - O resultado vale para qualquer árvore com K nós folha
  - Pior caso (máximo número de nós-folha): Imagem tipo tabuleiro de xadrez (todos os quadrantes têm que ser divididos até o nível de pixel)



Propriedades da Quadtree de Região (cont.)

- Espaço de armazenamento (cont):
  - Overhead dos nós internos pode ser aliviado usando representações sem ponteiros
    - Usar uma lista onde nós são enumerados em ordem de visita (ex. pos-ordem NW,NE,SW,SE)
    - Usar uma lista de códigos posicionais das folhas pretas
      - Cada nó é representado por uma sequência de dígitos em base 4 acompanhados por um indicador do nível do nó na árvore.
  - Espaço depende da posição dos pixels pretos dentro do sistema de referência da quadtree
    - Translადando uma mesma imagem pode-se obter mais ou menos blocos
  - Espaço é <u>linearmente</u> proporcional à resolução (n) e ao perímetro da imagem (p)

$$\#\text{Blocos} \leq 24 \cdot n - 19 + 24 \cdot p$$

Propriedades da Quadtree de Região

- Complexidade de algoritmos
  - Localização de ponto (Cor de um pixel): *O(log n)*
  - Conversão de imagem matricial: *O (n)*
  - Vizinhança: *O(log n)*

Conversão de Raster em Quadtree de Região

- Algoritmo "ingênuo"
  - Criar uma quadtree trivial (imagem com todos os pixels brancos)
  - Rotina "PintaPixel":
    - Dada a posição (x,y) de um pixel da imagem e sua cor (c), modificar a quadtree para refletir a mudança
  - Chamar a rotina "PintaPixel" para todos os pixels da imagem
  - Independente da ordem em que os pixels são pintados
  - Complexidade
    - Para imagem $n \times n$: $O(n^2 \log n)$
    - Subárvores podem ser criadas para ser logo depois deletadas
    - Desejável: $O(n^2)$

Conversão de Raster em Quadtree de Região (cont.)

- Rotina “PintaPixel” (detalhes)
  - Descer na árvore até localizar o nó-folha correspondente ao pixel
  - Se o nó-folha tem cor igual ao pixel, não faça nada
  - Caso o nó-folha seja de tamanho 1x1 (pixel), troque sua cor
  - Caso o nó-folha corresponda a mais de 1 pixel, subdivida-o até o nível de pixel, localize o nó-folha correspondente ao pixel a ser pintado e troque sua cor.
  - Subir do nó-folha à raiz juntando (“merging”) sempre que necessário, isto é, se os três irmãos do nó folha têm agora a mesma cor “c”

Conversão de Raster em Quadtree de Região (cont.)

- Assumir que toda a imagem cabe na memória principal
- Algoritmo ótimo pode criar um nó da quadtree apenas se este for necessário
  - O caso de quatro nós-folha irmãos com a mesma cor pode ser eliminado se os pixels da imagem forem consultados
- Rotina “ConstróiNó”
  - Recebe como parâmetro o conjunto de pixels correspondente à região do nó
  - Se é uma região 1x1, retornar a cor do pixel
  - Se é uma região contendo mais de um pixel, chamar ConstróiNó recursivamente para as 4 subregiões
    - Se todas as 4 subregiões são da mesma cor c (não cinza) retornar c
    - Senão
      - Construir os nós-folha (filhos)
      - Construir o nó cinza (pai)
      - Retornar cinza

Conversão de Raster em Quadtree de Região (cont.)

- Detalhes de implementação
  - Se a imagem não é um quadrado cujo lado é uma potência de 2,
    - Assumir a menor potência de 2 que contém a imagem
    - Ao acessar um pixel fora da imagem, assumir cor branca
  - Ao criar uma representação quadtree com ponteiros em memória
    - não é necessário criar tantos nós brancos e pretos quantos sejam o número de pixels da imagem, bastam um nó branco e um nó preto
    - O valor de retorno de ConstroiNó é um ponteiro
    - ConstróiNó só "constroi" nós cinza
- Complexidade
  - ConstroiNo é chamado apenas uma vez por nó
  - $O(n^2)$

Representação sem ponteiros de quadtrees

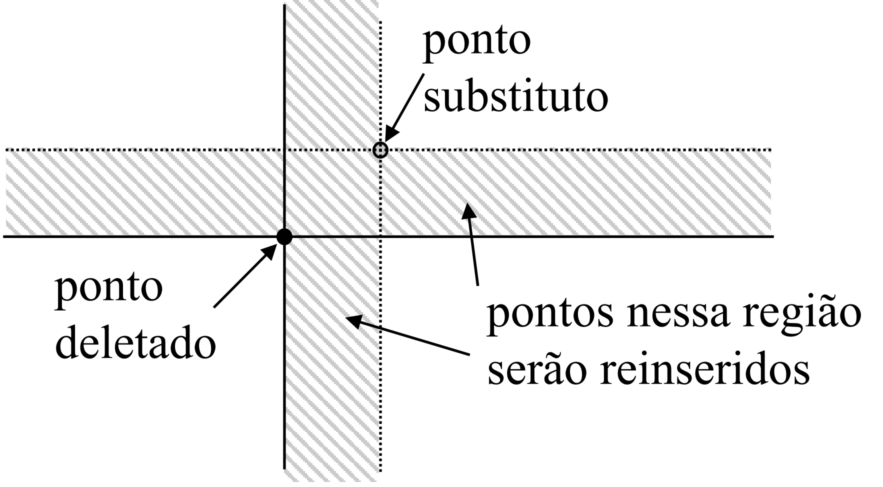
- Somente os nós folha são representados
- Cada nó-folha é representado por um número em base 4 (*P*) e um indicador de altura na árvore (*h*).
  - Indicam o caminho desde a raiz até a folha
  - $P=<p_n, p_{n-1}, \ldots, p_1>$, onde apenas os *h* primeiros dígitos (da esquerda para a direita) são significativos
- Se a imagem é binária (branco e preto), nós brancos não precisam ser representados
- Representação apropriada para armazenamento em disco
  - Usar uma estrutura para acesso sequencial indexado (e.g., B-tree)
  - Ordem das chaves definida por *P*
  - Não há necessidade de testar *h*
    - Um nó preto não pode ser pai de outro
  - Ordem corresponde à curva "Z" que passa por todos os nós pretos da quadtree

Quadtree de pontos

- Propriedades gerais
  - Extensão multidimensional da árvore de busca binária
  - Pontos são armazenados em nós internos
  - Depende da ordem de inserção dos pontos
  - Para *N* pontos inseridos segundo uma distribuição randômica uniforme, a altura esperada da árvore é *O(log N)*
  - Estrutura própria para armazenamento em memória
- Inserção de pontos
  - Algoritmo "ingênuo": Semelhante à inserção em árvores binárias
  - Problema de balanceamento: Assegurar altura logaritmica
  - Se os pontos são conhecidos de antemão, ordená-los segundo *x* ou *y* e escolher as medianas como raízes das subárvores

Quadtrees de pontos (cont.)

- Inserção de pontos (cont.)
  - Versão dinâmica de inserção balanceada
    - rebalancear a árvore sempre que um critério de balanceamento falhar
    - utiliza a idéia do balanceamento estático apenas na subárvore que infringiu o critério
- Deleção de pontos
  - Idéia do algoritmo análogo em árvores binárias não pode ser usada: nem sempre existem nós-folha que podem substituir o nó sendo deletado
  - Solução “ingênua”: reinserir todos os pontos da subárvore cuja raiz é o nó deletado
  - Solução melhorada: descobrir um “bom” nó-folha candidato e reinserir apenas os nós que tornariam a quadtree inválida

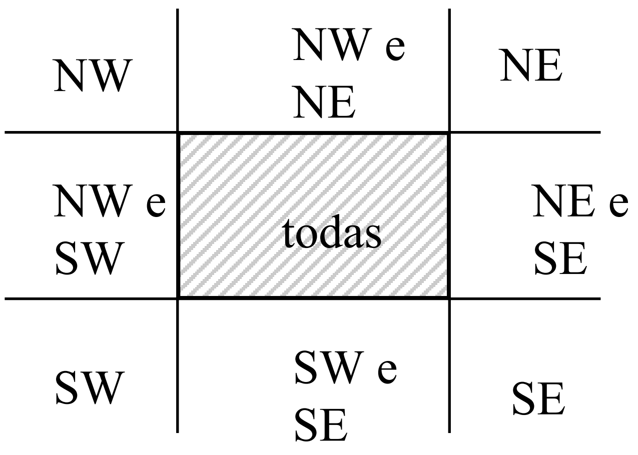


Quadtrees de pontos (cont.)

- Deleção de pontos (cont.)
  - A escolha do ponto substituto
    - 4 candidatos naturais (1 em cada quadrante)
    - Para achar o candidato do quadrante NW de *P*, caminhar sempre para SE do filho NW de *P*
    - Para escolher *Q,* o melhor dos 4 candidatos:
      - Critério 1: escolhendo *Q* nenhum dos outros 3 candidatos precisariam ser reinseridos
      - Critério 2: *Q* minimiza a área sombreada (métrica Manhattan entre *P* e *Q)*. Neste caso, no máximo 1 outro candidato estaria na área sombreada
  - Problema de deleção pode ser aliviado com o uso de uma pseudo-quadtree
    - Pontos são armazenados nas folhas
    - Nós internos são pontos que não fazem parte da massa de dados

Quadtrees de Pontos (cont.)

- Busca
  - Estrutura é apropriada para consultas envolvendo proximidade
  - Exemplo: todos os pontos que intersectam um retângulo *R*
    - As subárvores c/ raiz em filhos de *P* a serem pesquisadas dependem da posição de *P* com relação a *R:*



  - Exercício: Como realizar a busca do ponto mais próximo de um dado ponto de consulta *Q*?

K-D trees

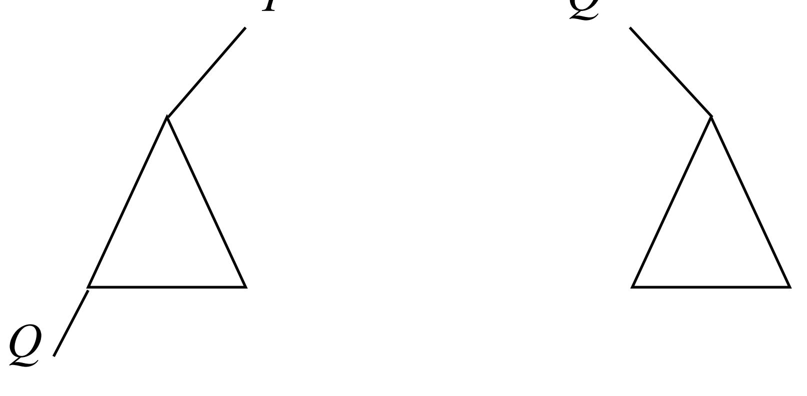
- Propriedades gerais
  - K refere-se à dimensão do espaço de pontos
  - Cada ponto inserido corta o espaço em duas subregiões segundo um hiperplano perpendicular ao *i'*ésimo eixo coordenado (discriminante)
  - O eixo discriminante varia alternadamente à medida que se desce na árvore (*x, y, x, y,* etc)
  - Cada nó só necessita de *k* valores para as as coordenadas do ponto e mais dois ponteiros para as subárvores. Não é necessário armazenar o discriminante
  - Tem características semelhantes à quadtree de pontos
    - Apenas uma comparação é necessária para cada nó da árvore
    - Desvantajoso em arquiteturas paralelas (pode-se testar K valores simultaneamente)

K-D trees (cont.)

- Inserção de pontos
  - Algoritmo “ingênuo” análogo ao das árvores de busca binária e ao das quadtrees de pontos
  - Problema de balanceamento também solucionado de forma análoga
  - K-D tree adaptativa é análoga à pseudo-quadtree de pontos (massa de dados estática)
    - Pontos são guardados nas folhas
    - Nós internos correspondem a hiperplanos que divide os pontos em subconjuntos de cardinalidade aproximadamente igual
    - Nós internos ocupam espaço invariante em relação a K
    - Relaxa-se a alternância de discriminantes em favor de uma melhor distribuição espacial entre os pontos
      - ex.: escolhe-se o hiperplano perpendicular ao eixo de maior dimensão do MBR de todos os pontos

K-D trees (cont.)

- Deleção de pontos
  - Semelhante à deleção em árvores binárias
  - Se o discriminante do nó *P* a ser deletado é *x*, devemos substituir *P* pelo nó *Q* à direita de *P* tal que Q_x é mínimo e, finalmente, deletar *P*
  - Porque não o nó *Q* à esquerda de *P* com valor máximo Q_x ? Pode haver mais de um, e a convenção é que nós à esquerda de *P* têm valores estritamente menores que P_x
  - Se a subárvore à direita de *P* é vazia, escolhe-se o nó *Q* à esquerda de *P* com Q_x mínimo, troca-se *P* por *Q* com a subárvore à esquerda de *P* agora posicionada à direita de *Q* e deleta-se *Q* de sua antiga posição recursivamente

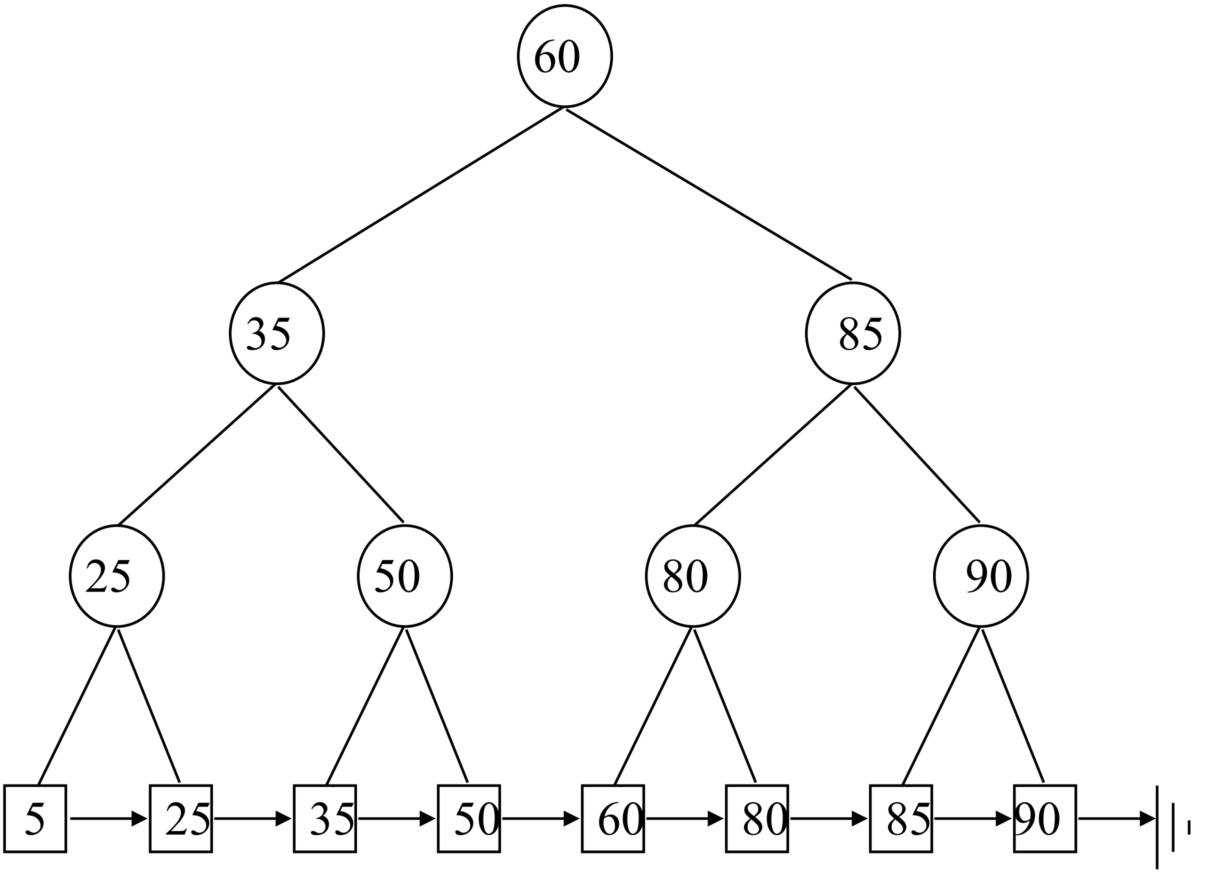


K-D trees (cont.)

- Deleção de pontos (cont.)
  - Cuidado na busca de um ponto substituto: ele pode estar somente em um dos ramos de uma subárvore se o discriminante desta for *x*, mas pode estar em qualquer ramo se o discriminante for *y*
  - Complexidade de buscar um substituto: $O(N^{1-1/K})$
- Busca
  - Algoritmos semelhantes à quadtree de pontos
  - Range search tem complexidade $O(K \cdot N^{1-1/K})$ (não óbvio)
    - Análise não leva em conta o custo de reportar cada ponto, mas apenas o custo de descer nas subárvores que não estão inteiramente contidas na região sendo buscada

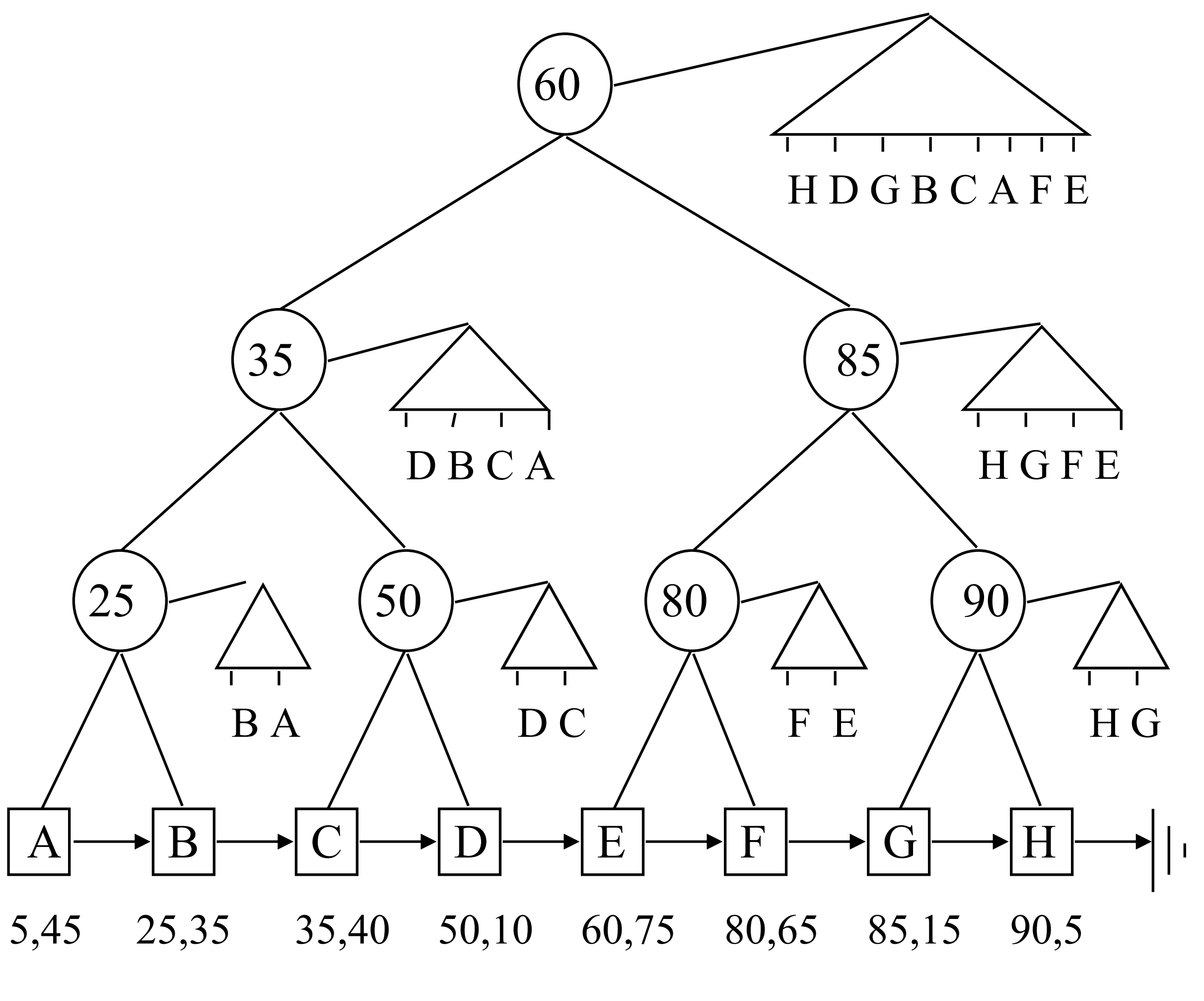
Range Trees

- Visam solucionar em tempo ótimo o problema de busca de pontos incluidos num intervalo N-dimensional (Um retângulo em 2-D)
- Uma Range Tree em 1-D é simplesmente uma árvore binária balanceada com os pontos armazenados nas folhas e encadeados entre si formando uma lista



Range Trees (cont.)

- Uma Range Tree em 2D é uma Range tree de Range trees. Cada nó interno da RT para (x) contém uma subárvore que é uma RT para (y)

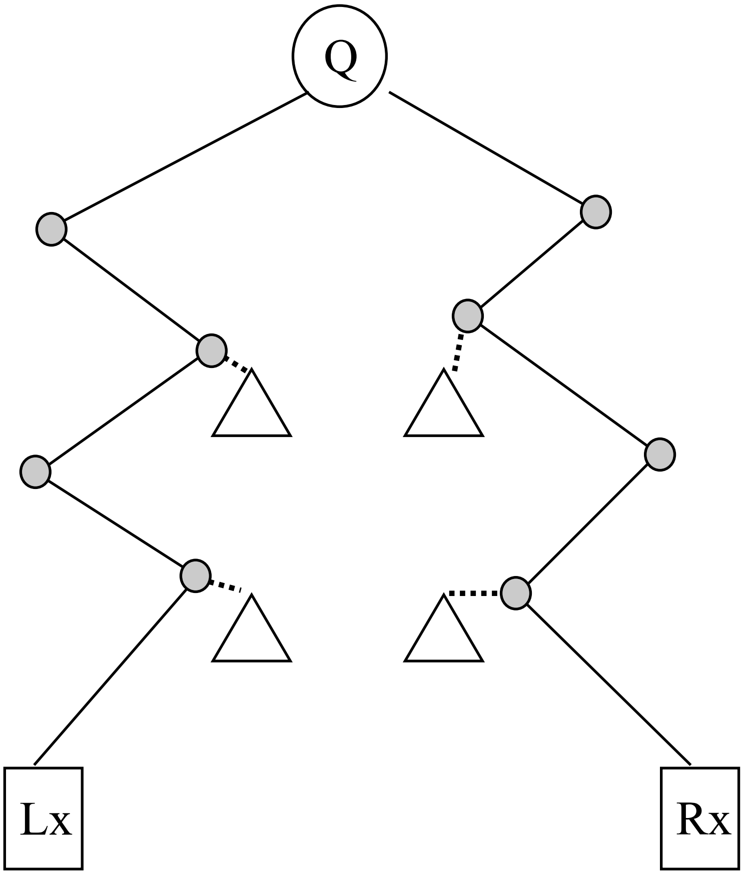


Range Trees (cont)

- Algoritmo p/ busca de um intervalo $[L_x , R_x][L_y , R_y]$
  - Buscar Na RT (x)
    - LXP: o menor nó c/ $x \geq L_x$
    - RXP: o maior nó c/ $x \leq R_x$
  - Se LXP e/ou RXP estão no intervalo, reportar
  - Encontrar Q, o ancestral comum mais próximo
  - PATH_LXP é o conjunto de nós internos desde Q (exclusive) até LXP
  - Analogamente para PATH_RXP
  - Para todo nó P em PATH_LXP tal que LEFT(P) também está em PATH_LXP
    - Buscar na RT(y) de RIGHT(P) os nós cujos y estão no intervalo $[L_y , R_y]$ e reportar
  - Para todo nó P em PATH_RXP tal que RIGHT(P) também está em PATH_RXP
    - Buscar na RT(y) de LEFT(P) os nós cujos y estão no intervalo $[L_y , R_y]$ e reportar

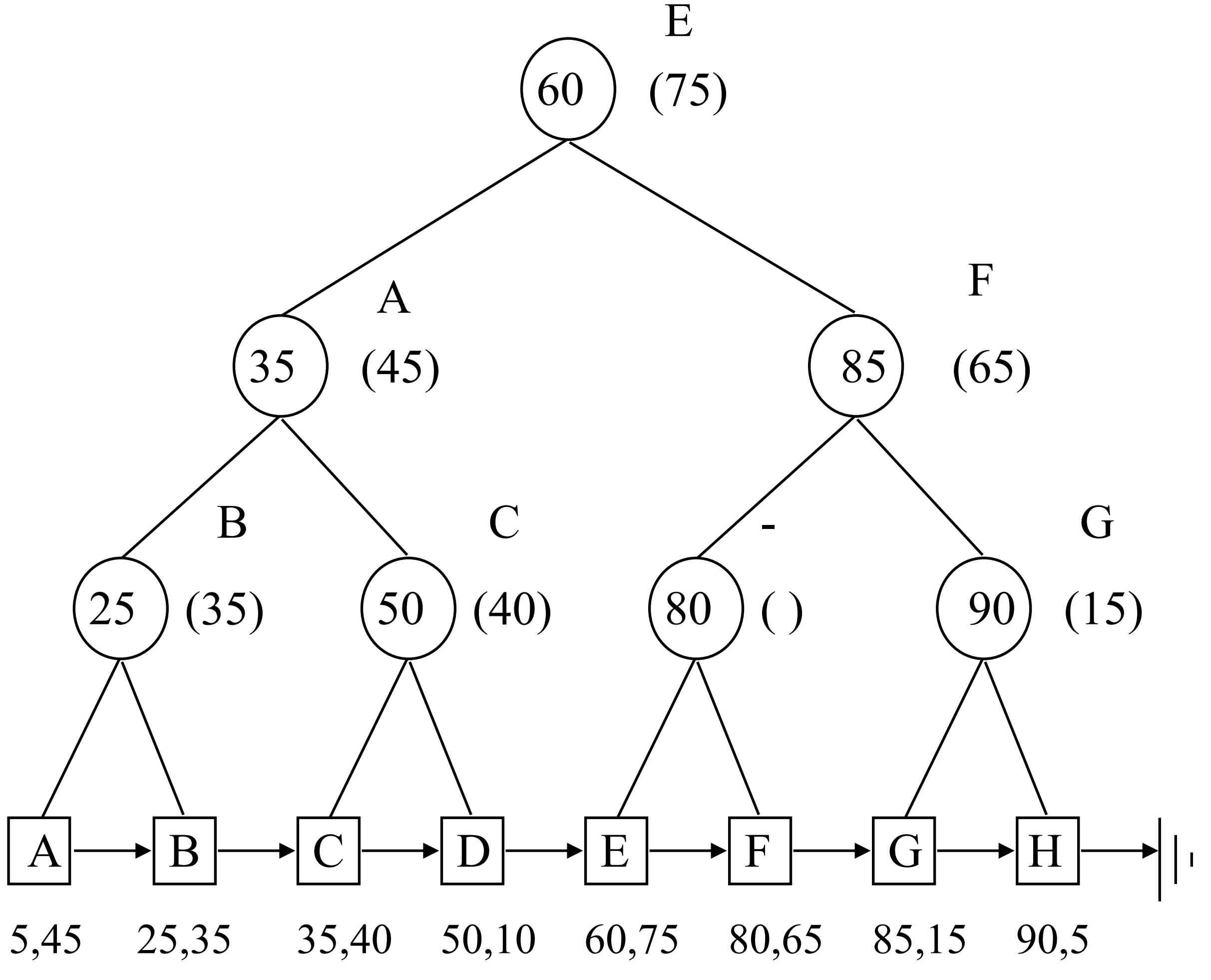
Range Tree (cont.)

- Subárvores RT(y) pesquisadas para um dado intervalo [Lx,Rx],[Ly,Ry]



Priority Search Tree

- Semelhante à árvore de busca normal em 1D, exceto que cada nó interno também armazena um ponteiro p/ nó folha c/ valor máximo de y entre seus descendentes, desde que esse valor não tenha aparecido em um de seus ascendentes
- Própria para buscas em intervalos $[L_x , R_x][L_y , \infty)$



Priority Search Tree

- Algoritmo de busca p/ intervalo semi-infinito $[L_x , R_x][L_y , \infty)$
  - Descer na árvore buscando Q, ancestral comum mais próximo de LXP e RXP
    - Seja P o nó folha associado com o nó sendo examinado (T)
    - Terminar se Py< L_y pois não há descendentes de T com y > L_y
    - Terminar se P é nulo, pois subárvore já foi toda examinada e/ou reportada
    - Caso contrário, reporte P se Px contido no intervalo $[L_x , R_x]$
  - Uma vez encontrado Q, determine recursivamente onde continuar a busca
    - RIGHT(T) se T e RIGHT(T) no caminho à direita de Q até LXP
    - LEFT(T) se T e LEFT(T) no caminho à esquerda de Q até RXP
    - Senão, em ambos RIGHT(T) e LEFT(T)
  - O (log2 N + F) tempo para buscar N itens e F respostas

Quadtree de Região e métodos afim

- Dividem o espaço em regiões previamente preestabelecidas e não estabelecidas por pontos previamente inseridos à la Point Quadtree
- Variantes:
  - MX-Quadtree:
    - Divisão tipo quadtree de região
    - Assume domínio discreto (coordenadas inteiras entre 0 e 2^n) e pontos distintos
    - Pontos são sempre armazenados em nós-folha de nível mais baixo possível
    - Não há nós brancos, apenas cinzas e nós folha contendo pontos
    - Inserção pode requerer até *n* subdivisões
    - "Collapsing" acontece apenas em quadrantes de onde foi deletado o último ponto
    - Boa para matrizes esparsas
    - Ruim quando o domínio está muito populado por pontos (um grid é melhor)

Quadtrees de região e métodos afim (cont.)

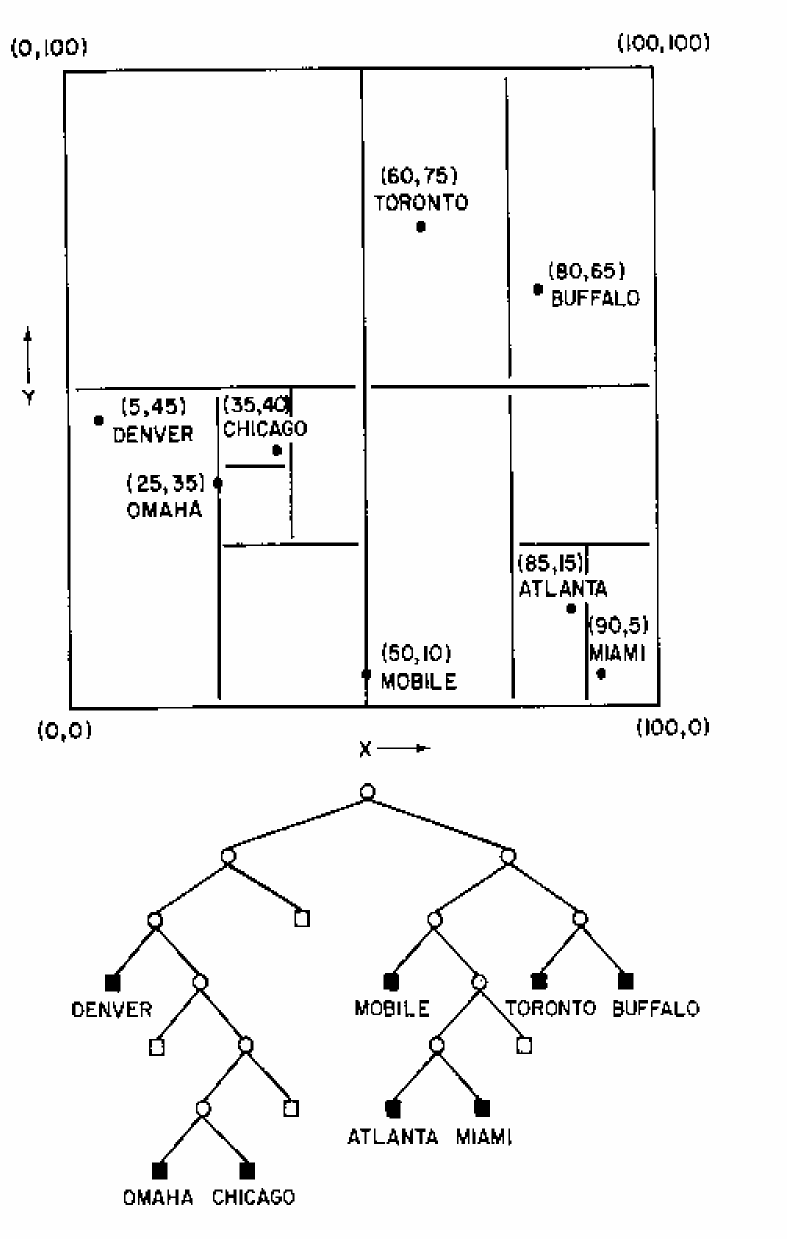
- Exemplo: Algoritmo para achar a transposta de uma MX-quadtree:
  - Transpose (MX)
    - If MX é nó folha, retornar
    - Trocar ponteiro NE com ponteiro SW
    - Transpose (filho NW)
    - Transpose (filho NE)
    - Transpose (filho SW)
    - Transpose (filho SE)

Quadtree de Região e métodos afim (cont.)

- Variantes (cont.)
  - PR-quadtree
    - Idêntica à Region quadtree exceto que não há nós brancos e nós pretos correspondem a pontos
    - Subdivisão ocorre sempre que mais de um ponto aparece num mesmo quadrante
  - PR-bintree (também chamada PR-k-d-tree)
    - Semelhante a PR-quadtree, mas divisão ocorre de forma binária alternando entre os eixos
  - BD-tree (ou BANG file)
    - PR-quadtree onde nós cinza com apenas um filho são "comprimidos"
    - Semelhantemente, uma BD-k-d tree usa compressão em uma PR-bintree (PR-k-d-tree)

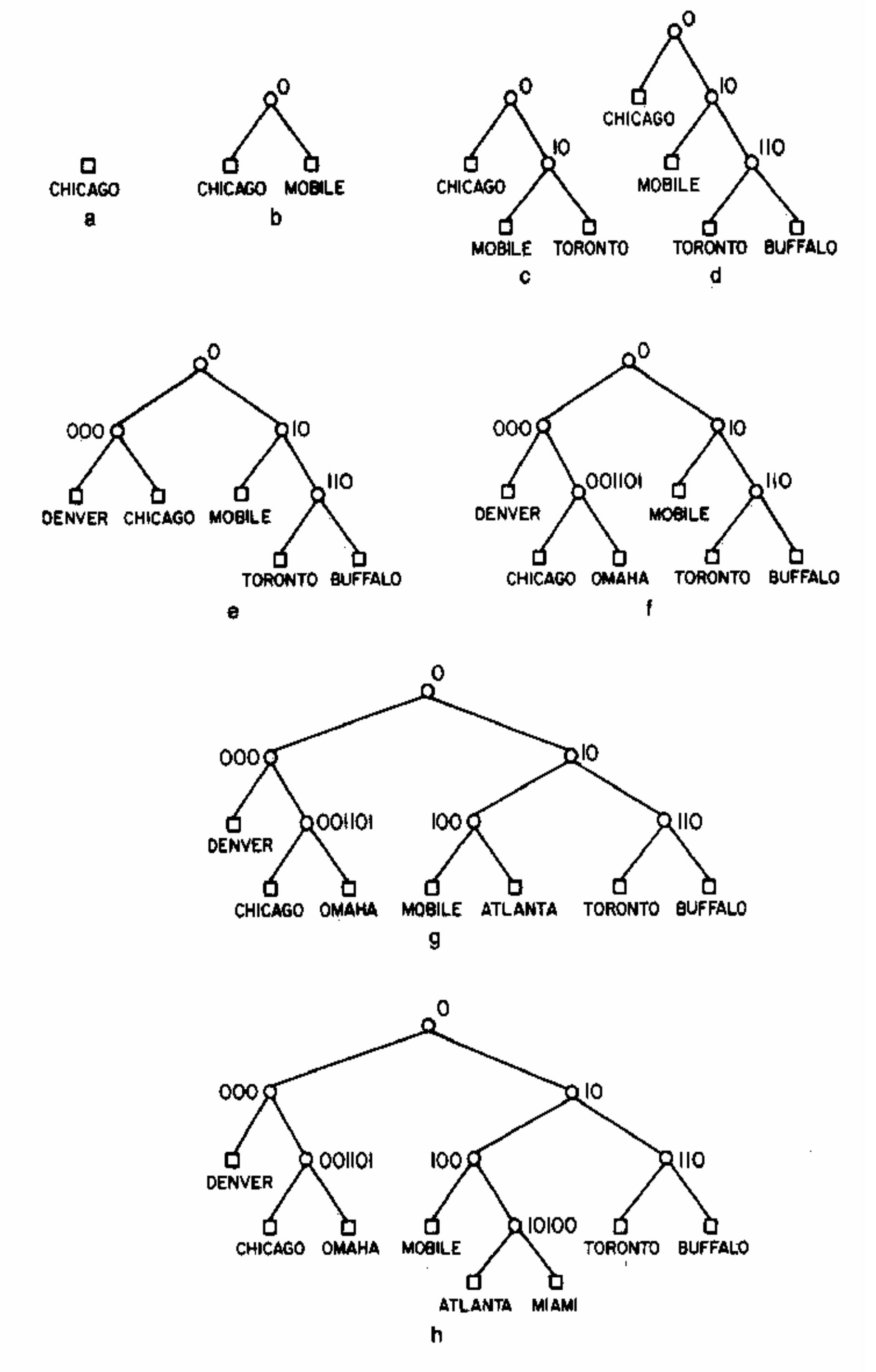
Quadtree de Região e métodos afim (cont.)

- Exemplo: PR-bintree (PR-k-d-tree)



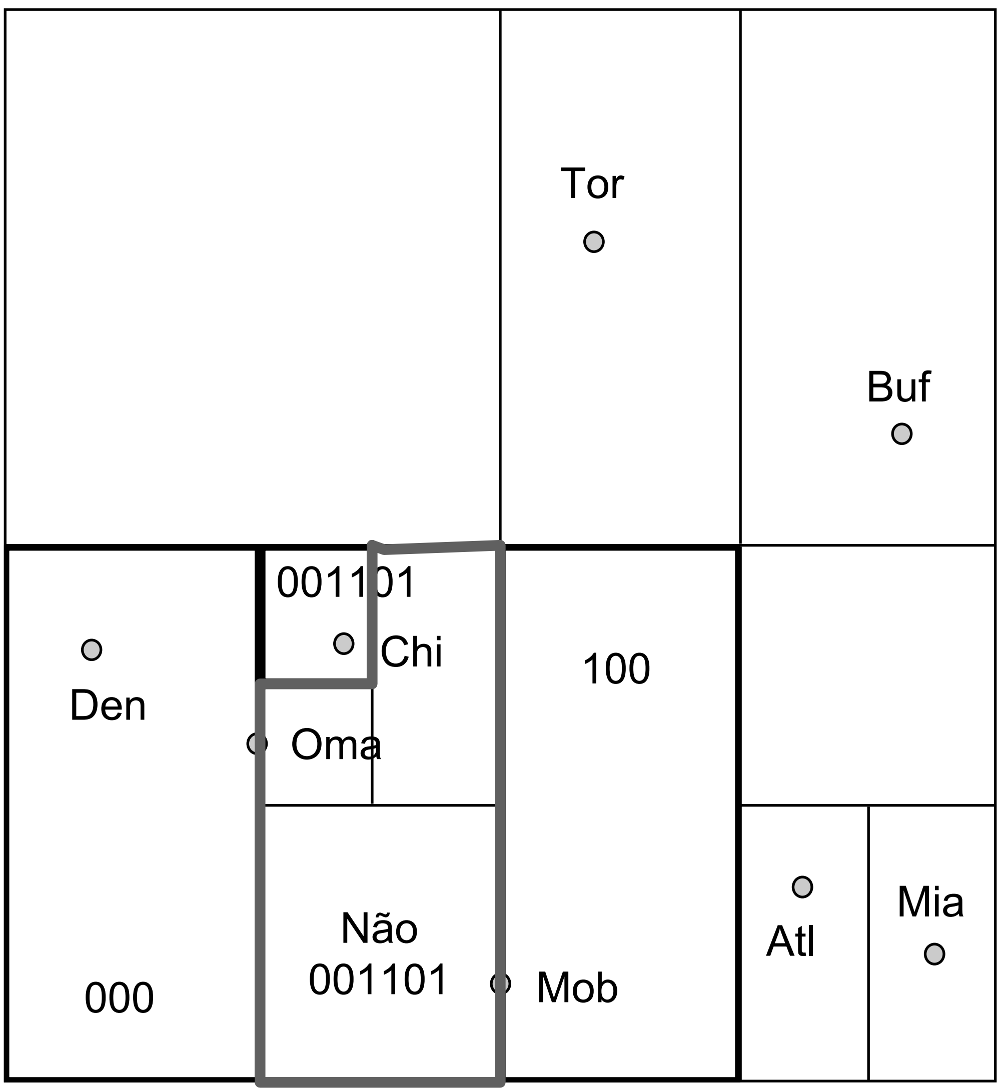
Quadtree de região e métodos afim (cont.)

- BD-bintree (BD-k-d-tree)



Quadtree de região e métodos afim (cont.)

- Usando compressão de caminho, as regiões não necessariamente correspondem a hiperretângulos



Métodos de repositórios (“buckets”)

- Estruturas com ponteiros são (geralmente) ineficientes para armazenamento em disco
  - ponteiros podem atravessar a fronteira entre um bloco de disco e outro
- Uma técnica geral é usar repositórios (buckets) para armazenar os pontos (ou outros tipos de dados espaciais)
  - Um repositório corresponde a um conjunto de pontos espacialmente próximos e que cabem num bloco de disco
  - Exemplo mais simples: grid regular
  - Problemas fundamentais:
    - Como organizar a informação a respeito da região do espaço correspondente a cada bucket
    - O que fazer quando um bucket transborda (overflow)
    - Como evitar buckets com poucos pontos (underflow)

Métodos hierárquicos com repositórios

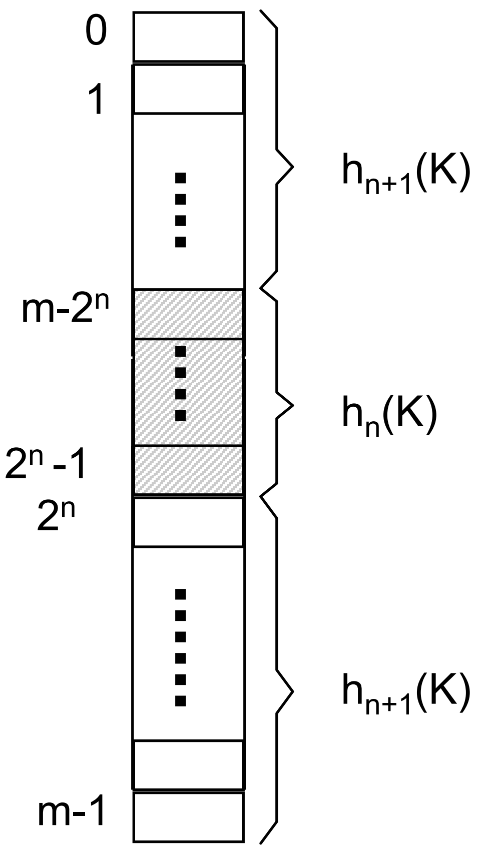
- Correspondem aos métodos hierárquicos vistos anteriormente, porém ocm o critério de subdivisão relaxado para que cada subregião contenha *c* pontos, no máximo (ao invés de apenas 1)
- Têm o efeito de diminuir a ligação entre a altura da árvore e a distância entre pontos individuais
  - Altura dependente da distância entre *conjuntos* de pontos
  - Efeito ainda mais pronunciado com BD-trees
  - Útil para aplicações onde deseja-se focalizar grupos depontos próximos (clusters)
    - bucket adaptive k-d tree(Dynamically Quantized Space -DQS)
    - Dynamically Quantized Pyramid
- Mais utilizados em GIS e BD espaciais
  - R-tree e suas variantes
  - PMR quadtree (linear)

Métodos não hierárquicos de repositórios

- Identidade do bucket pode ser determinada por
  - Uma função de hash
    - Bucket *B* corresponde aos pontos *P* tais que *H(P) = B*
    - À medida que o númerode pontos cresce, o espaço de endereçamento tem que ser aumentado
      - Reinserção de pontos
    - Ex.: linear hashing, spiral hashing
  - Região correspondente a uma célula de uma grade
    - grid file
    - EXCELL
- Problemas de overflow são geralmente tratados através de buckets auxiliares encadeados ao bucket principal (apontado pelo diretório ou função de hash)

Linear Hashing

- Esquema de hashing onde o espaço de endereçamento cresce um bucket (endereço) por vez
- Utiliza duas funções de hash para *m* buckets



$$h_n(K) = K \bmod 2^n$$

$$h_{n+1}(K) = K \bmod 2^{n+1}$$

$$2^n <= m < 2^{n+1}$$

Linear Hashing (cont.)

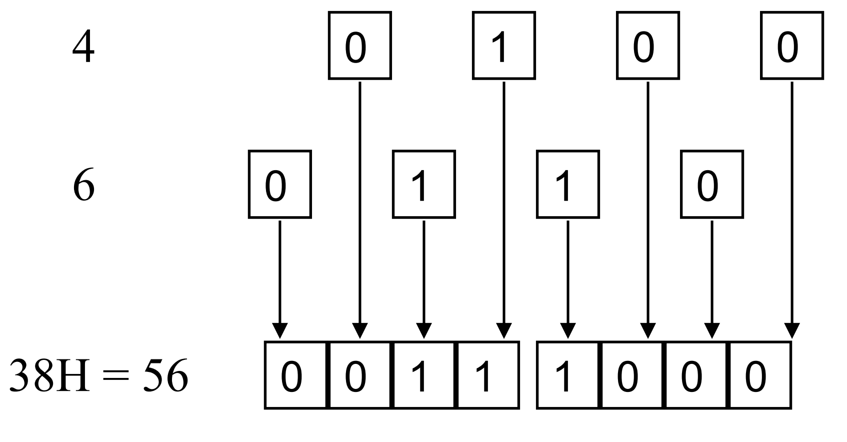
- Quando *m* é uma potência de 2, todos os buckets são endereçados por h_n
- Ao se usar linear hashing para armazenar pontos, é preciso utilizar um esquema de linearização do espaço (bit interleaving, por exemplo), isto é, $K = f(x,y)$
- A criação de um novo bucket ocorre quando a taxa de utilização ultrapassa um limite pré estabelecido
- Taxa de utilização = número de pontos / número de entradas vazias (tanto em buckets primários quanto em buckets de overflow)
- Ao se criar um novo endereço (m), os pontos residentes no primeiro endereço correspondente a h_n são reinseridos e buckets de overflow que se tornem desnecessários são desalocados
- Se buckets de overflow são desalocados, a taxa de utilização pode crescer novamente e provocar a criação de um novo endereço

Linear Hashing (cont.)

- Ao se criar um novo endereço, todos os endereços antigos, exceto um, continuam sendo acessados pelas mesmas funções de hash
- Nota: não há relação entre os buckets que contêm os pontos que serão reinseridos e aqueles que estão mais “cheios”
- Após o espaço de endereços ser dobrado, todos os pontos nos buckets anteriormente existentes terão sido reinseridos
- Em se tratando de pontos, as funções de hash originais são ineficientes em manter pontos próximos em buckets próximos
  - Solução, antes de aplicar a função de hash, inverter a ordem dos bits, de maneira que os mais significativos serão afetados pelo operador **mod**
- Qual função de hash deve-se aplicar primeiro?
  - Resposta: h_{n+1}
  - Se um dado K satisfaz as condições para as duas, h_{n+1} deve ser usada sob pena de K não mais ser encontrado quando houver entre 2^{n+1} e 2^{n+2} endereços

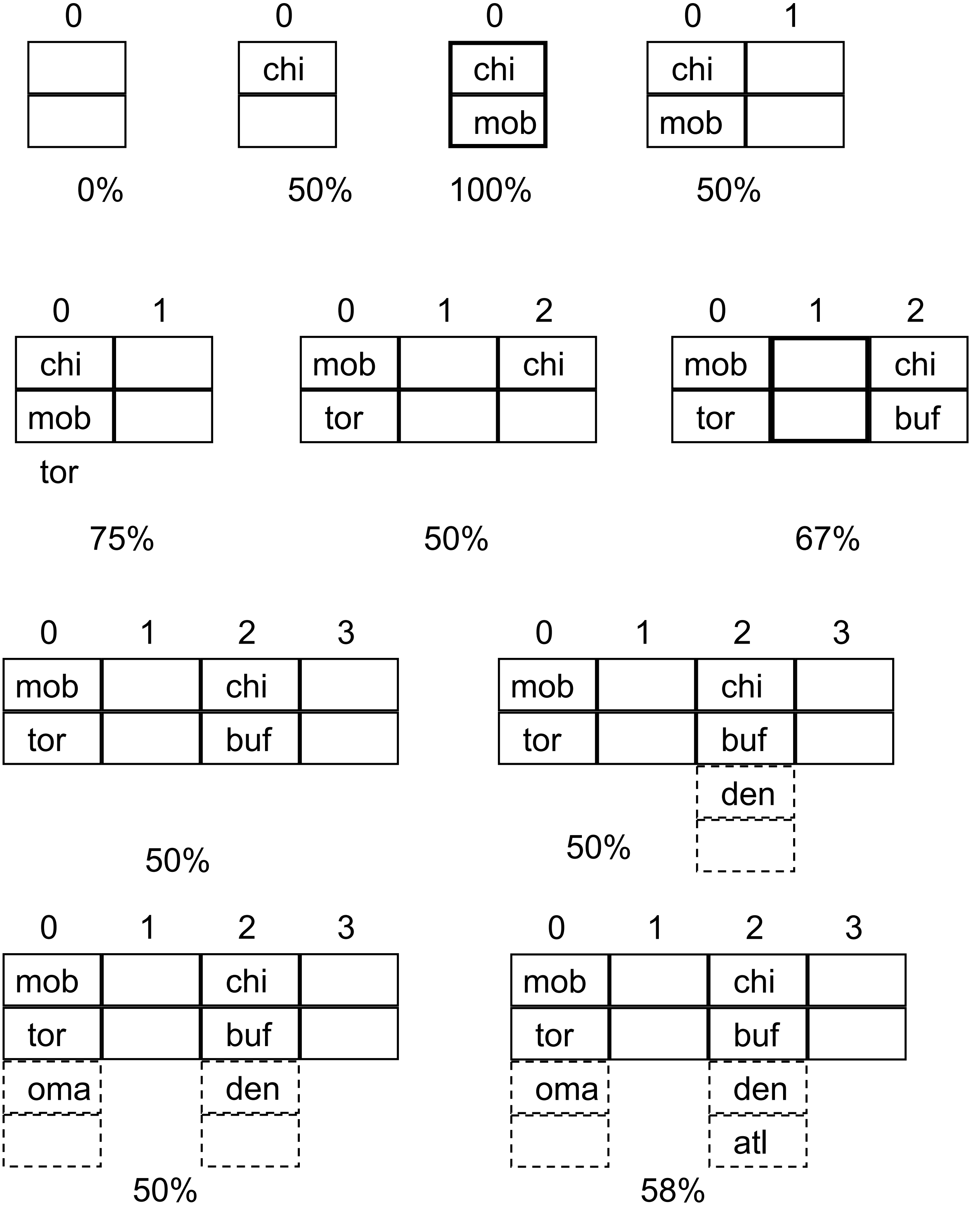
Linear Hashing (cont.)

- Exemplo usando a massa de dados “padrão” com linearização usando bit-interleaving
  - Denver (0,3) 10
  - Omaha (2,2) 12
  - Chicago (2,3) 14
  - Mobile (4,0) 16
  - Miami (7,0) 21
  - Atlanta (6,1) 22
  - Buffalo (6,5) 54
  - Toronto (4,6) 56
- Exemplo de cálculo:

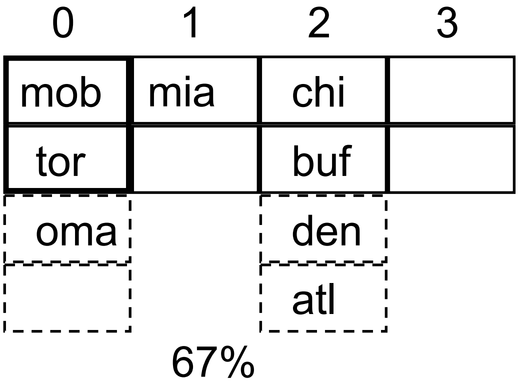


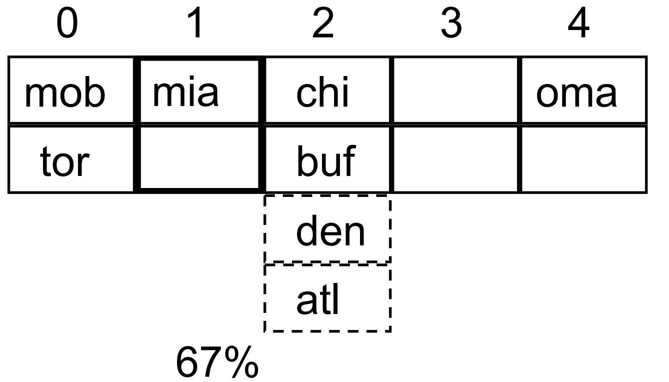
Linear Hashing (cont)

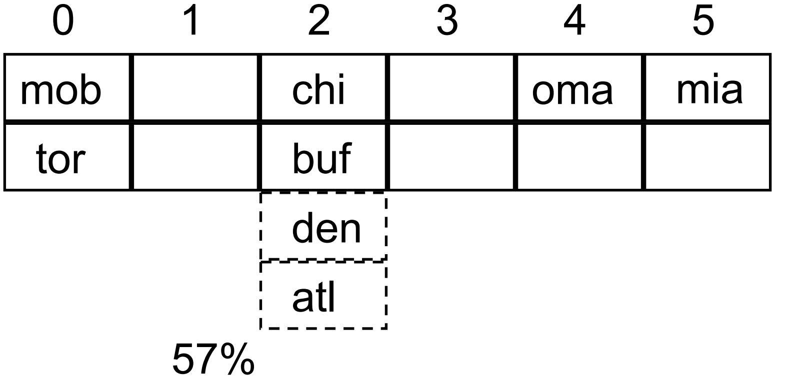
- Assumir 1 pontos por bucket, utilização máxima 66%



Linear Hashing (cont.)



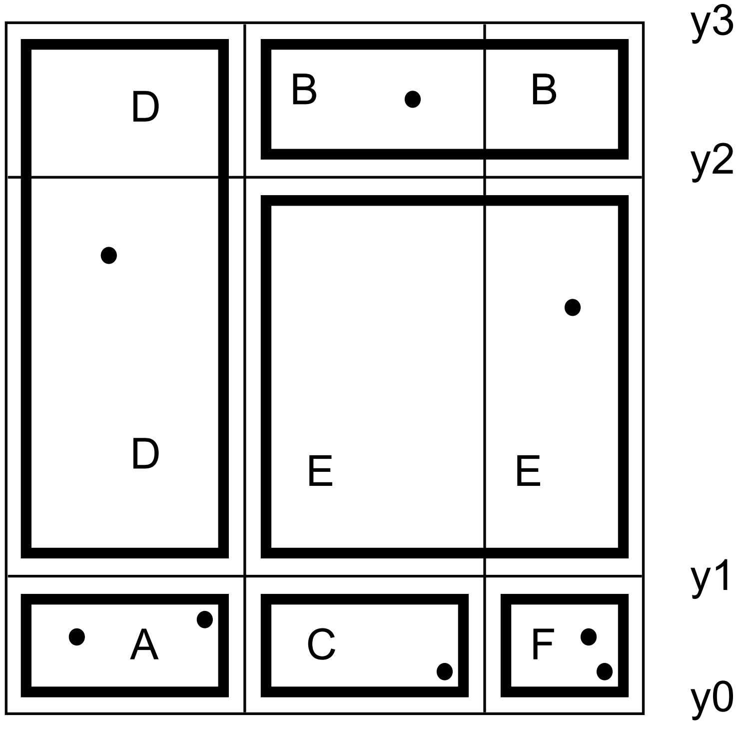




Grid File

- Método de repositório
- Espaço é dividido numa grade cujos hiperplanos divisores não precisam ser estar em posições regulares
- Cada célula da grade corresponde a um único bucket
- Cada bucket pode corresponder a mais de uma célula, porém a região definida por todas elas precisa ser convexa (hiper-retângulo)
- Estrutura de acesso composta de dois elementos
  - Um diretório (array) que mapeia células em buckets (armazenado em disco)
  - Escalas lineares (uma para cada dimensão), onde as cotas dos hiperplanos de partição são armazenados (mantido em memória)
- Buckets armazenam tipicamente de dezenas a milhares de pontos (um bloco)
- Principal virtude
  - Acesso a cada ponto usando apenas 2 acessos a disco

Grid file (cont.)



D	B	B
D	E	E
A	C	F

Diretório

x0	x1	x2	x3
y0	y1	y2	y3

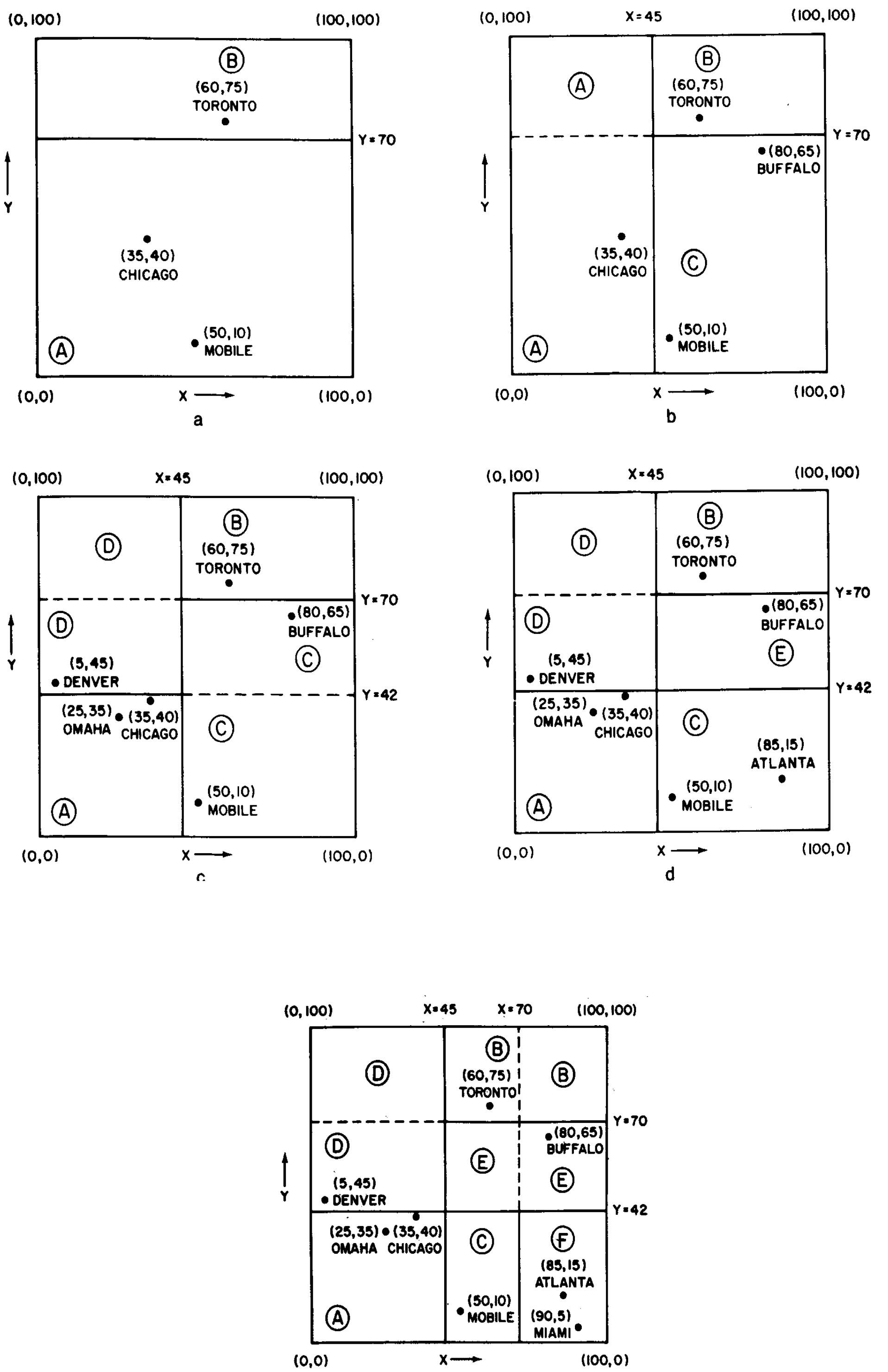
Escalas Lineares

Grid File (cont.)

- Algoritmo de inserção
  - Localizar, usando o diretório e as escalas, a célula onde o ponto deve ser inserido
  - Se o bucket apontado pela célula ainda tem espaço, inserir ponto e terminar.
  - Se o bucket com overflow atravessa alguma fronteira das escalas lineares, ele é compartilhado por mais de uma célula.
    - Criar um novo bucket
    - Repartir os pontos entre os dois buckets
    - Tentar a inserção novamente
  - Se o bucket com overflow é apontado por somente uma célula
    - Criar um novo hiperplano de subdivisão em um dos eixos coordenados
    - Atualizar a escala correspondente
    - Partir todas as células atravessadas pelo hiperplano e atualizar o diretório
    - Tentar a inserção novamente

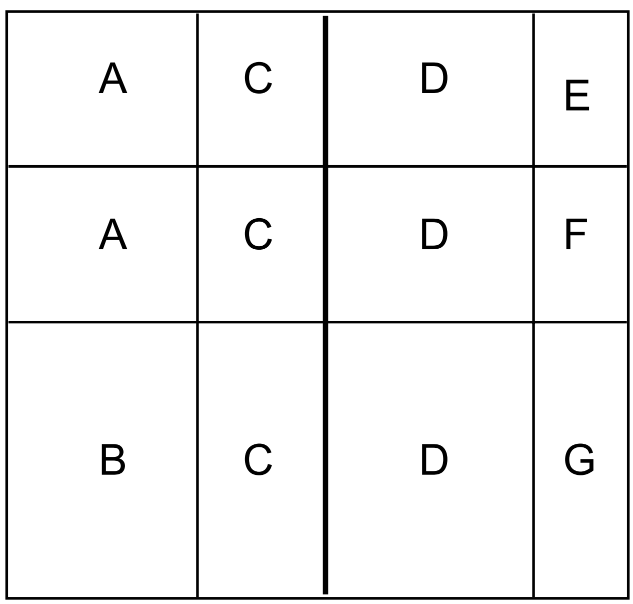
Grid file (cont.)

- Exemplo



Grid File (cont.)

- Implementação eficiente do diretório é fundamental
- Inicialmente, é comum operações de inserção requererem o refinamento da grade (hiperplanos)
- Mais tarde, o mais comum é haver buckets compartilhados com varias células
- Analogamente, operações de deleção podem requerer que buckets sejam desalocados sendo o conteúdo repartido com buckets vizinhos
- Deleção também pode requerer que hiperplanos sejam removidos



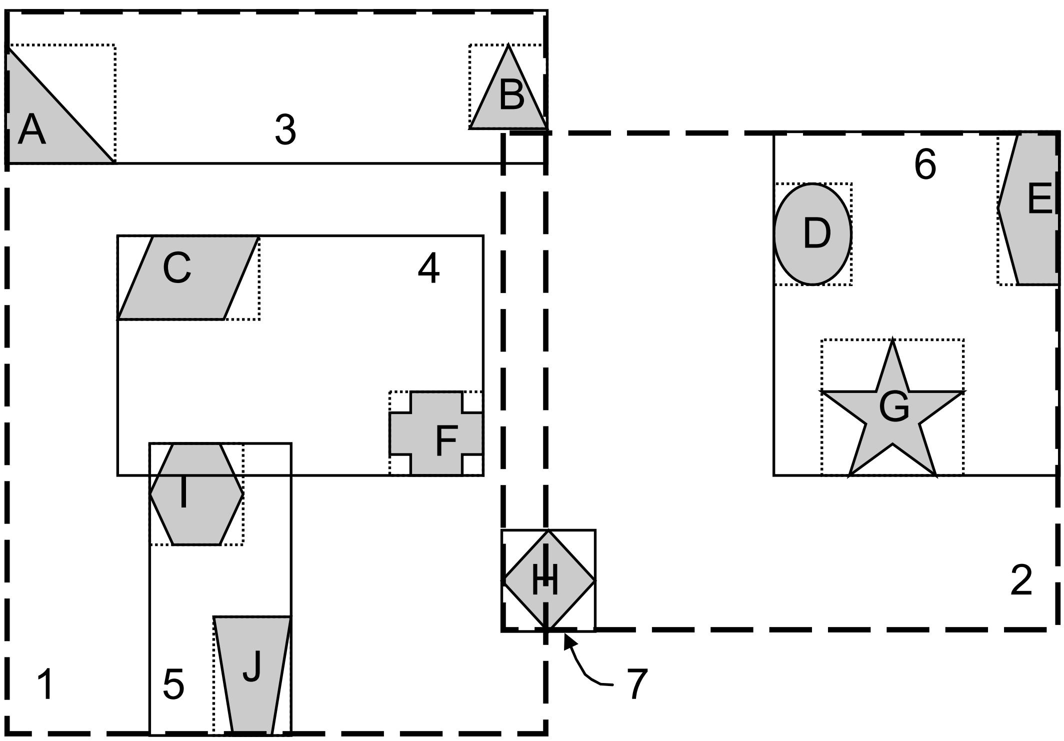
EXCELL

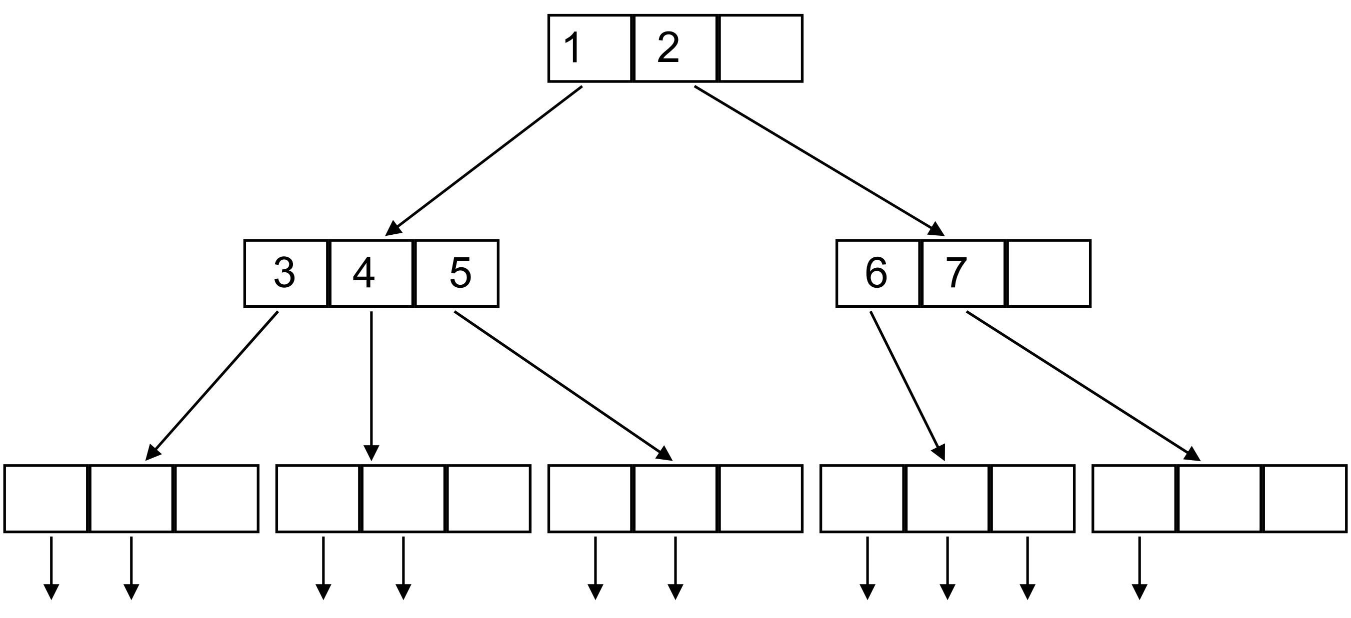
- Semelhante ao Grid File
- Partição do espaço entretanto, é regular, à la bintree
- Não precisa de escalas e acesso ao diretório é mais fácil
  - Pode ser usado cálculo de endereço
- Operação de splitting pode requerer que o diretório seja dobrado
- EXCELL também tem semelhança com linear hashing
  - Diretório ao invés de função de hash
  - Linear Hash requer mais buckets pois não há garantia que o bucket que dá overflow é o que é partido e redistribuido
  - EXCELL não tem buckets de overflow

R-tree

- Árvore balanceada nos moldes da B+-tree
- Cada nó (bloco de disco) contém uma coleção de pares retângulo/apontador
- Nós internos:
  - apontador para subárvore
  - retângulo é o MBR de todos os objetos na subárvore
- Nós-folha
  - apontador para dado espacial
  - retângulo é o MBR do objeto espacial
- Se *M* é a capacidade de um nó (número máximo de entradas), estabelece-se um limite mínimo $m <= M/2$ para a ocupação de todos os nós da árvore, exceto a raiz
- Crescimento da árvore se dá das folhas para a raiz, que tem ao menos 2 filhos, a menos que seja um nó folha
- Todos os nós-folha estão na mesma altura na árvore

R-trees (cont.)





R-trees (cont.)

- Busca (dado que intercepta um ponto *P*)
  - Descer todas as subárvores cujo retângulo contém *P*
- Inserção de um dado *E*
  - Escolher o nó folha *L* onde *E* será inserido
    - Seja *N* o nó raiz
    - [1] Se *N* é folha, retornar *N*
    - Escolher o filho *F* de *N* que resulte no menor crescimento do retângulo de *F*
      - Se há empate, escolher o filho com menor área
    - *N* ← *F*; goto [1]
  - Se *L* tem espaço suficiente, instale *E* em *L*; senão, divida o nó *L* em dois nós folha *L* e *LL* contendo todos as entradas de *L* e mais *E*
  - Subir na árvore a partir de *L* ajustando os retângulos dos nós-pai
  - Se um segundo nó *LL* foi criado, inserir uma entrada contendo o MBR de *LL* e um ponteiro para *LL* no nó pai de *L* usando a mesma técnica usada para inserir *E* no nó folha

R-trees (cont.)

- Deleção de uma entrada E
  - Procurar o nó folha L contendo E a partir da raiz
    - Nota: mais de uma subárvore pode ser visitada para cada nó
  - Remover E de L
  - Propagar as mudanças subindo na árvore
    - Defina Q, um conjunto de nós com underflow
    - Inicialize $N \leftarrow L$
    - [1] Se N é a raiz da árvore, termine este passo
    - Inicialize $P \leftarrow$ pai(N)
    - Remova a entrada de N de P
    - Se N tem menos de m entradas, inclua N em Q; senão, ajuste o MBR de N em P
    - $N \leftarrow P$; goto [1]
  - Reinserir todos as entradas de todos os nós em Q
    - Nota: Tomar cuidado em reinserir entradas em suas alturas originais na árvore
  - Se a raiz tiver agora apenas uma entrada, delete-a e mova o filho único para a raiz

R-trees (cont.)

- Deleção pode também ser feita à semelhança do processo análogo em B-trees
  - Nó com underflow redistribuido em nó irmãos
    - Usar o irmão que terá menor acréscimo em sua área
  - Reinserção é mais simples e tem a vantagem de “rearrumar” a árvore
  - Ambas as alternativas podem requerer que nós tenham que ser divididos
- Algoritmo para divisão de nós
  - Requerido para inserção e deleção
  - *M+1* entradas são repartidas em duas coleções (dois nós) tais que
    - ambos tenham entre *m* e *M* entradas
    - a divisão satisfaça um critério heurístico que minimize a complexidade de novas operações de busca
      - critério de Guttman: MBR’s das duas coleções devem ser os menores possíveis (área mínima)

R-trees (cont.)

- Divisão de Nós (cont.)
  - Algoritmo exaustivo
    - tempo exponencial em *M*
    - imprático
  - Algoritmo quadrático
    - Escolher como sementes o par de entradas (i,j) cujo MBR conjunto desperdiça mais área com relação aos MBR's individuais
      - desperdício = area (MBR (Ri,Rj))-area area (MBR (Ri))-area(MBR(Rj))
      - custo = $O(M^2)$
    - [1] Escolher entre as entradas restantes a que exibe maior preferência por um dos dois grupos
      - di = acrescimo de area requerido para inserir entrada no grupo i.
      - escolher a entrada para a qual |di-dj| é maximo
    - Repetir [1] ate que todas as entradas estejam distribuidas ou ate que um grupo esteja tão vazio que requeira todas as entradas restantes

R-trees (cont.)

- Divisão de Nós (cont.)
  - Algoritmo Linear
    - Semelhante ao algoritmo quadratico
    - Escolha das sementes: escolher o par a maior separacao normalizada em qualquer dos eixos
      - Ex.: separacao em x.
        - Assumir xmax(j) < xmin(i)
        - sep = (xmax(j)-xmin(i)) / (xmax(i)-xmin(j))
    - Escolha da proxima entrada a ser distribuida em um dos grupos
      - Escolher qualquer uma
      - Inserir no grupo que terá menor incremento de area

R*-trees

- Beckmann et al identificam 4 parâmetros de otimização para o problema de distribuição de retângulos nos nós
  - A área de cada retângulo deve ser mínima
    - reduz a área de desperdicio entre o MBR do nó e o conjunto de retângulos incluídos no nó
  - A área de interseção entre retângulos em cada nível da árvore deve ser mínima
    - reduz a probabilidade de ter que descer em diversas subárvores durante buscas
  - O perímetro (margem) de cada retângulo deve ser mínimo
    - faz com que os MBR's sejam aprox. quadrados
    - quadrados são mais facilmente agrupados que retângulos
  - A taxa de utilização deve ser máxima
    - nós cheios levam ao acesso de menos nós
    - critério importante em range queries com intervalos grandes

R*-trees (cont.)

- Problemas da R-tree de Guttman
  - O critério de área (1) é o único contemplado, o que leva à diminuição das áres de interseção (2), mas não a retângulos approx. quadrados (3) ou a boas taxas de utilização (4)
  - O algoritmo de escolha de sementes privilegia sementes “pequenas”
    - Leva a retângulos não “quadrados”
    - Uma vez que um dos grupos tem um MBR “grande”, é provavel que este seja escolhido novamente para entradas subsequentes, pois o incremento de área fica relativamente pequeno
  - Se durante a divisão, um grupo é escolhido raramente, a uma certa altura todas as entradas restantes são atribuidas a ele sem levar em conta qualquer criterio geométrico
  - problemas podem resultar em grupos com excessiva área de interseção ou distribuidos desigualmente (baixa taxa de utilização)

R*-trees (cont.)

- Algoritmo modificado para escolher a subárvore onde inserir uma dada entrada
  - Se o nó sendo considerado aponta para nós-folha
    - Escolher a folha que resultar em menor incremento da área total de interseção no nó pai
    - Se empatar escolher a folha que resultar em menor incremento de área
    - Se empatar escolher a folha com menor área
  - Nos outros casos, agir como na R-tree
  - Algoritmo é quadrático, porém custo pode ser reduzido se nem todos os retangulos forem considerados
    - ordenar os entradas por incremento de área
    - aplicar o algoritmo apenas às primeiras *p* entradas (tipicamente 32)
  - Experimentos indicam melhor performance para range queries com retângulos pequenosem massas de dados com pequenos retângulos ou pontos
  - Obs.: Cálculo da área de overlap

$$\text{overlap}(E_k) = \sum_{i=1..k,\, i\neq k} \text{area}(E_k \cap E_i)$$

R*-tree(cont.)

- Algoritmo modificado de divisão de nós
  - Para cada eixo
    - ordenar retângulos uma vez pelo valor mínimo e outra pelo valor máximo
    - para cada ordenacao, computar M-2m+2 distribuições. Distribuição k
      - Grupo 1: as primeiras (m-1)+k entradas
      - Grupo 2: as entradas restantes
    - para cada ordenação computar: perímetro(MBR(Gr.1))+perímetro(MBR(Gr.2))
    - somar os valores de perímetro obtidos para todas as distribuições
  - Escolher o eixo que resultar no menor valor total de perímetro e neste eixo computar para cada distribuição: area(MBR(Gr.1) $\cap$ MBR(Gr.2))
  - Escolher a distribuição que resultar em menor área de interseção. Se empatar, escolher a que resulte em area(MBR(Gr.1))+area(MBR(Gr.2))

Coleções de Retângulos

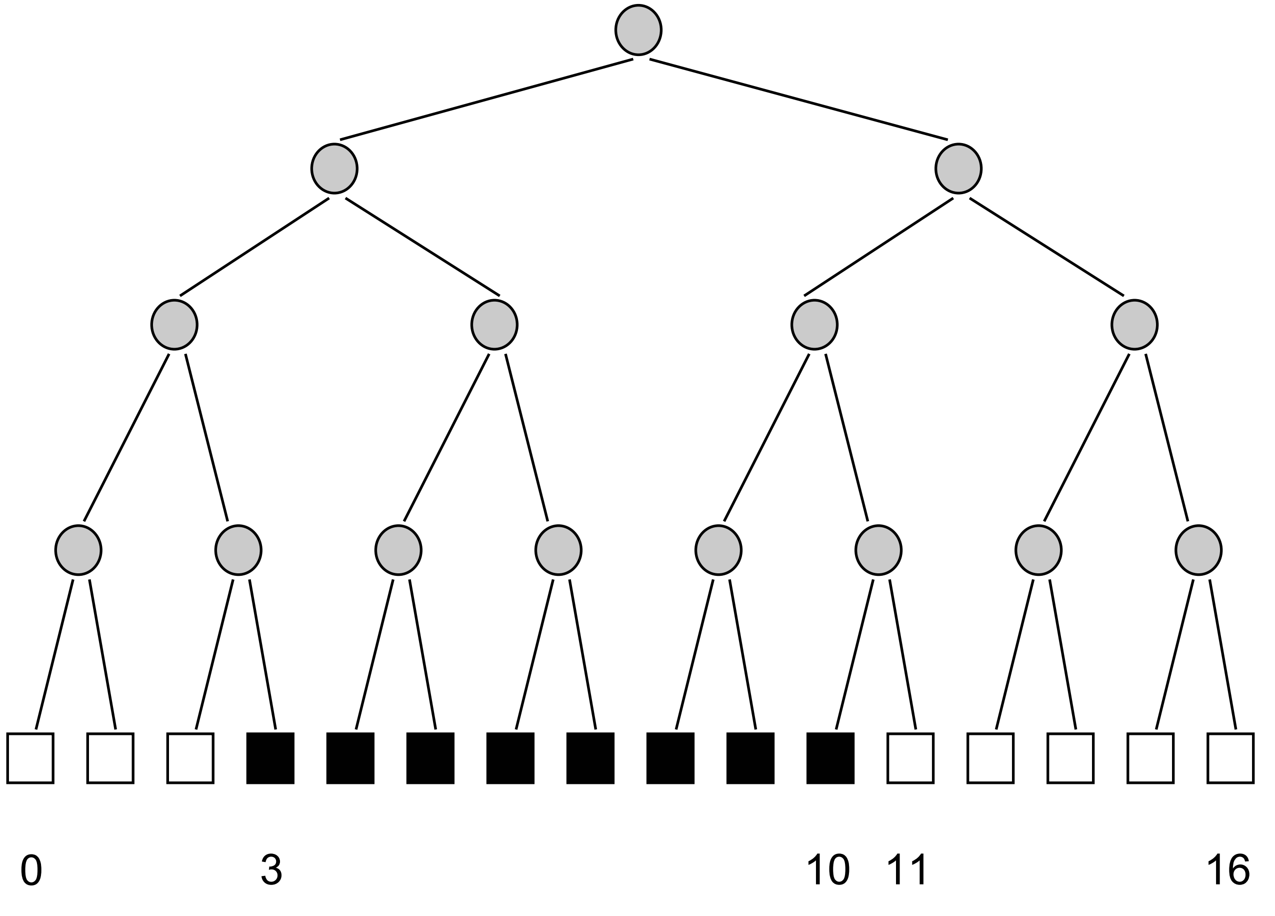
- Coleções de retângulos são massas de dados tipicamente usadas no desenho de máscaras VLSI
  - Processamento desses dados requerem operações tais como verificação de proximidade entre dois caminhos ou de conectividade elétrica entre dois retângulos
  - Retângulos se caracterizam por ser “finos” e “compridos”
- Também usadas para aproximar a distribuição espacial de outros dados, isto é, cada retângulo é o MBR de um dado espacial. Operações típicas
  - detectar pares de candidatos que podem satisfazer um determinado predicado num join espacial
  - detectar adjacência entre dados. Por exemplo, se um mapa poligonal é especificado por uma coleção de segmentos de reta, deseja-se determinar a conectividade entre esses segmentos
  - range search: busca de todos os dados que intersectam um retângulo

Métodos de Varredura

- Em inglês: plane sweep
- A idéia geral é dividir um problema n-dimensional em diversos problemas (n-1)-dimensionais
- Tal divisão é conseguida deslocando-se um hiperplano (de dimensão n-1) no espaço n-dimensional e detectando-se quais retângulos interceptam o plano (retângulos ativos)
- A varredura normalmente é precedida por um “sort” dos valores mínimos e máximos de cada retângulo na direção do eixo de varredura. Cada um desses valores corresponderá a um evento de inserção ou deleção do retângulo no hiperplano de varredura
- O plano de varredura é representado por uma estrutura de dados que seja adequada para a inserção e deleção de retângulos. Se os retângulos originais residem em espaço bidimensional, a estrutura deve ser própria para representar intervalos em uma dimensão. Exemplos:
  - Árvores de segmentos (segment trees)
  - Árvores de intervalos (interval trees)

Árvores de Segmentos Unitários

- A árvore de segmentos corresponde a uma sofisticação da árvore de segmentos unitários que, por sua vez nada mais é do que uma variação da quadtree de região em uma dimensão:
  - Nós pretos (folhas) são trechos do segmento
  - Nós pretos no último nível da árvore (mais baixo) correspondem a pixels em uma dimensão
  - Ex: Segmento 3-11



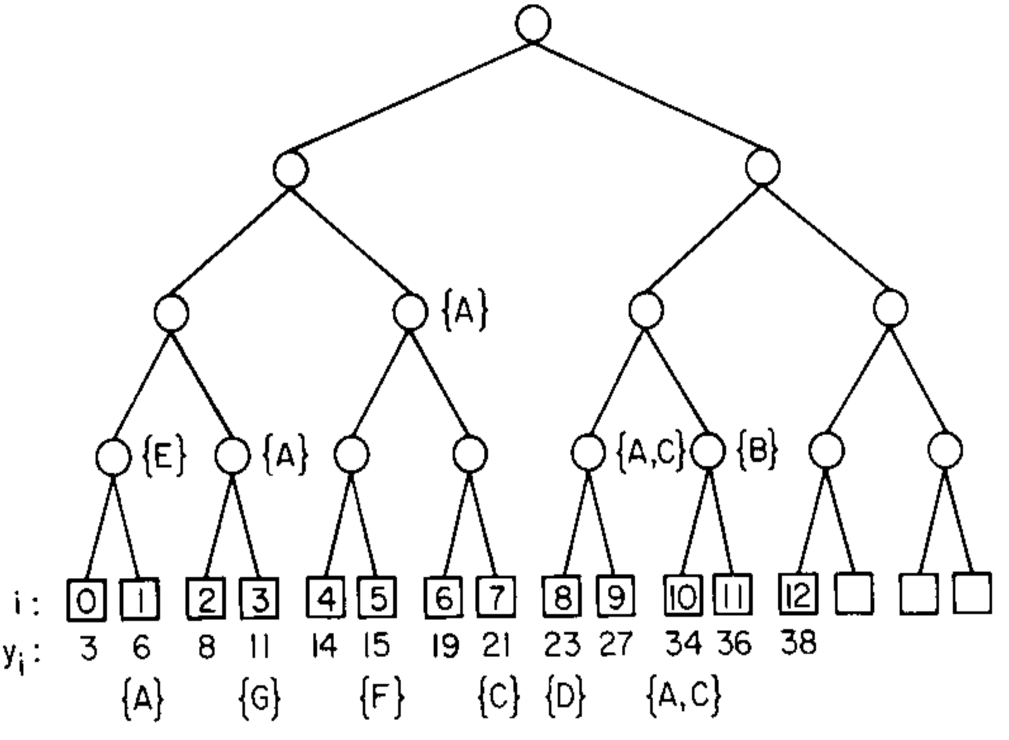
Árvores de Segmentos

- A árvore de segmentos unitários só funciona para um segmento por vez e requer uma discretização do espaço. A árvore de segmentos elimina essas restrições
- Dados N segmentos, ordena-se as coordenadas de suas extremidades (até 2.N valores): $y_o, y_1 \ldots y_m$
- Cada nó terminal *i* da árvore corresponde ao intervalo $[y_i, y_{i+1})$
- Se um nó está contido num intervalo *A*, mas seu pai não, então ele é rotulado com *A*
- Todos os rótulos de um dado nó são armazenados numa lista duplamente encadeada
- Inserção é feita de forma análoga à da árvore de segmentos unitários, exceto que o rótulo tem que ser inserido na lista encadeada
- Deleção requer uma estrutura auxiliar ligando cada segmento *S* a cada nó da árvore rotulado com *S*
  - Um array de listas de ponteiros

Árvores de Segmentos

- Espaço:$O\ (N \log N)$
- Tempo para inserir um segmento: $O\ (\log N)$
- Tempo para deletar um segmento: $O\ (\log N)$

A: [6:36)
B: [34:38)
C: [21:36)
D: [23:27)
E: [3:8)
F: [15:19)
G: [11:14)



Árvores de Segmentos (cont.)

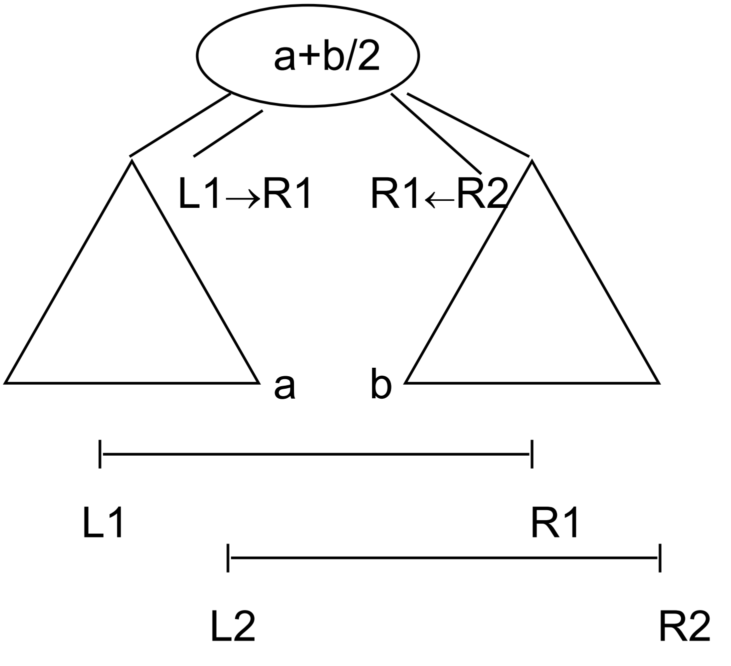
- Procedimento para achar todos as interseções entre os retângulos
  - Ao adicionar a borda mínima $S = [l,r)$ de um retângulo R na estrutura verificar
    - [1] Todos os segmentos que começam antes de l e terminam depois de l
    - [2] Todos os segmentos que começam entre l e r
  - Problema [1] pode ser solucionado com árvore de segmentos em $O(\log N+F)$
    - Fazer uma busca do menor segmento possível que começa em l, anotando os rótulos dos segmentos encontrados no caminho da raiz até a folha
  - Problema [2] é solucionado com árvore de segmentos em $O(N)$
    - É necessario visitar todos os nós folha entre l e r

Árvore de Segmentos (cont.)

  - Problema [2] pode ser resolvido em O (log N) usando uma range tree (espaço $O(N)$)
    - Procura-se todos os pontos de início de segmento entre l e r
- O problema de detecção de todos as interseções entre todos os retângulos pode ser resolvido em tempo $O(N.\log N+F)$ e espaço O (N) usando-se uma árvore de intervalos ou uma priority search tree

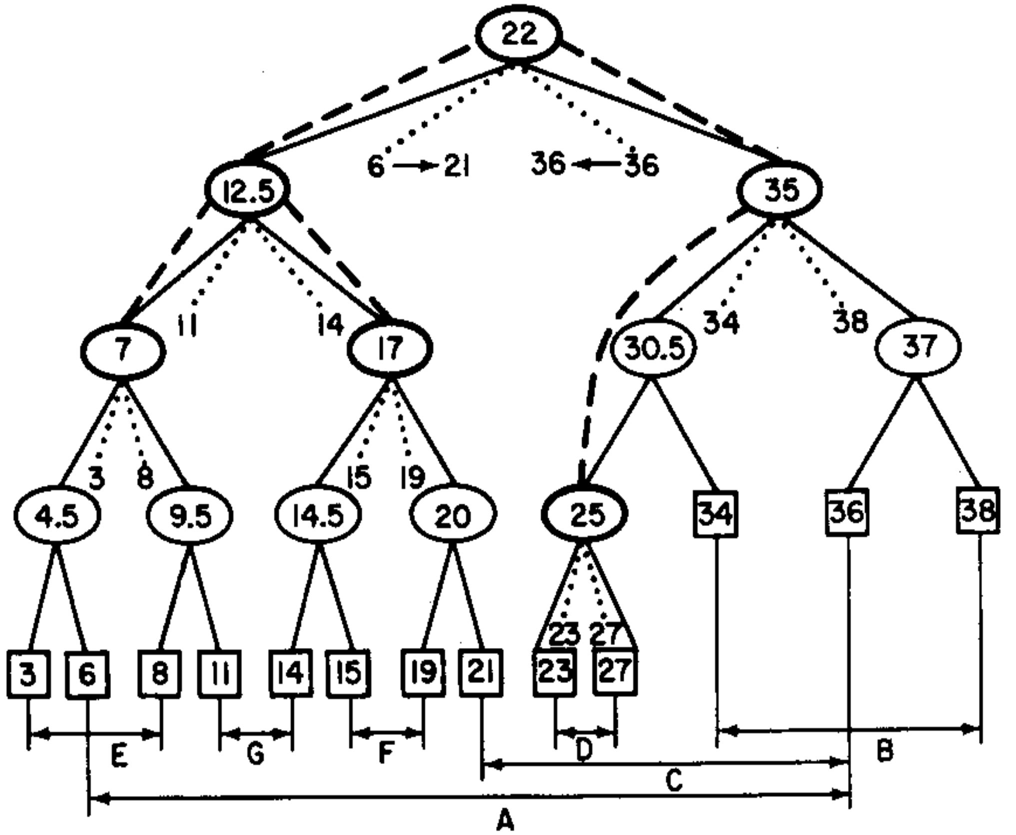
Árvore de intervalos

- Diferentemente da árvore de segmentos, cada intervalo é marcado apenas uma vez na árvore
  - Segmento $S=[l,r)$ é marcado apenas no nó interno que é seu ancestral comum mais próximo
- Nó interno contém discriminante que é um valor entre a maior folha descendente à esquerda e a menor folha descendente à direita



Árvore de Intervalos (cont.)

- Estrutura secundária:
  - Cada nó interno *V* aponta para os pontos mínimos (*Li*) / máximos (*Ri*) de cada segmento *I* para o qual *V* é o ancestral mais próximo encadeados em ordem crescente/decrescente
- Estrutura terciária:
  - Cada nó interno aponta para o nó ativo mais próximo da subárvore à esquerda/direita
  - Um nó ativo é aquele que possui estrutura secundária ou seus dois filhos tenham descendentes ativos



Árvore de Intervalos (cont.)

- Algoritmo de inserção de segmento [*L,R*):
  - Buscar *V,* o nó ancestral comum mais próximo das duas extremidades do segmento e inseri-las na sua estrutura secundária
- Algoritmo para reportar interseções

  1) Começar na raiz e buscar *V*

  2) Começar em *V* e localizar *L* na subárvore à esq.

  3) Começar em *V* e localizar *R* na subárvore à dir.