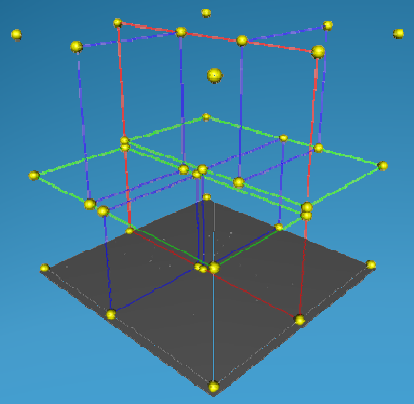


# DBSCAN

Usando kd-Trees

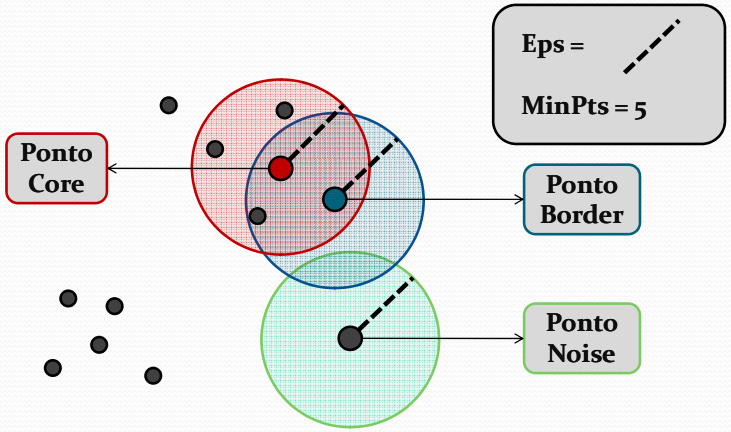
Lucas Vendramin
5961586

## Sumário

- DBSCAN
- kd-Trees
- DBSCAN com kd-Trees
- Outras Estruturas
  - R*-Trees
  - DBSCAN com R*-Trees



## DBSCAN






## DBSCAN

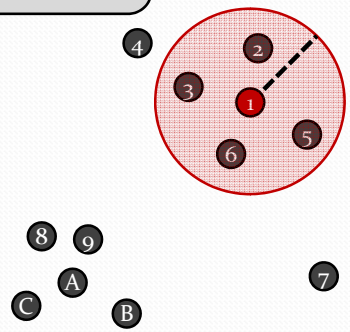
- **Parâmetros**: *Eps* e *MinPts*
- **Algoritmo**:
  1. Rotular os objetos como: **Core**, **Border** ou **Noise**.
  2. "Remover" objetos **Noise**.
  3. Inserir uma aresta entre cada par de objetos "**core**" vizinhos.
  4. Rotular cada componente conexo como um cluster.
  5. Atribuir cada objeto "**border**" ao cluster de um dos cores associados.

*M. Ester et.al, 1996*

# Exemplo

Eps = 

MinPts = 5



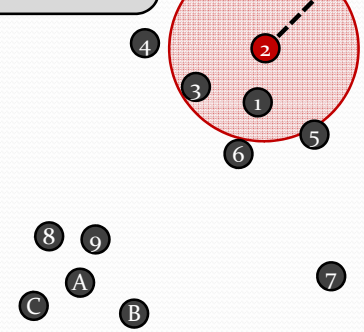

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | Border |
| 3 | Border |
| 4 | |
| 5 | Border |
| 6 | Border |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |



# Exemplo

Eps = 

MinPts = 5



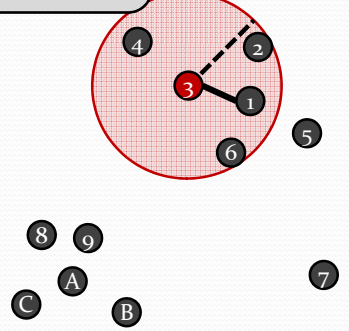

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | Border |
| 4 | |
| 5 | Border |
| 6 | Border |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |



# Exemplo

Eps = 

MinPts = 5



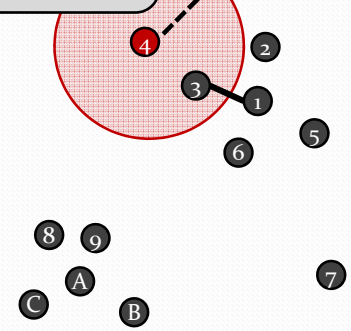

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | Border |
| 5 | Border |
| 6 | Border |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |



# Exemplo

Eps = 

MinPts = 5

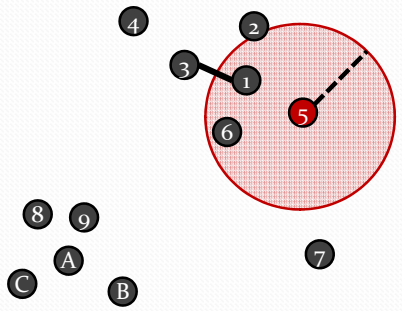



| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | Border |
| 6 | Border |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |

# Exemplo

Eps = 

MinPts = 5



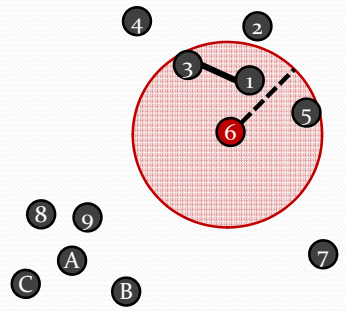

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | Border |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |



# Exemplo

Eps = 

MinPts = 5



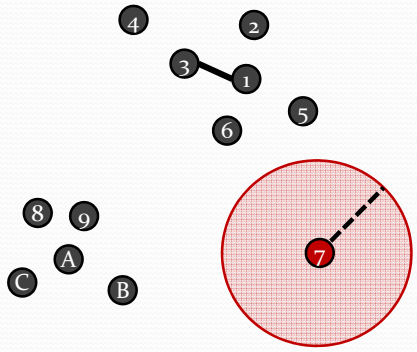

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |



# Exemplo

Eps = 

MinPts = 5



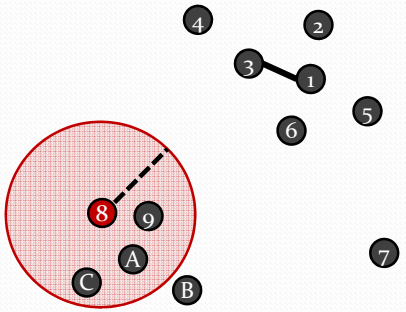

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |



# Exemplo

Eps = 

MinPts = 5

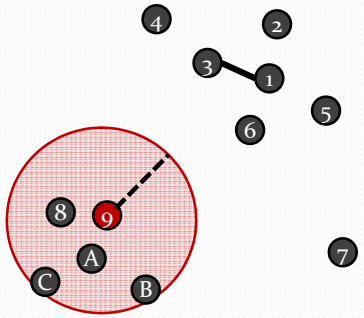



| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | |
| 8 | |
| 9 | |
| A | |
| B | |
| C | |

# Exemplo

Eps = 

MinPts = 5



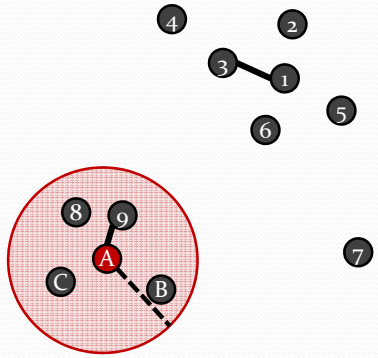

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | |
| 8 | BORDER |
| 9 | CORE |
| A | Border |
| B | Border |
| C | Border |



# Exemplo

Eps = 

MinPts = 5



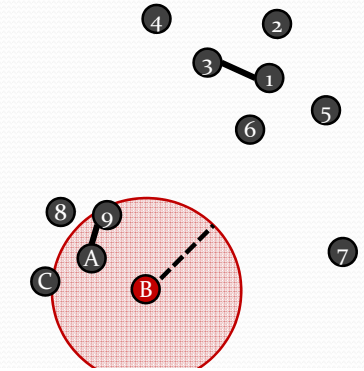

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | |
| 8 | BORDER |
| 9 | CORE |
| A | CORE |
| B | Border |
| C | Border |



# Exemplo

Eps = 

MinPts = 5



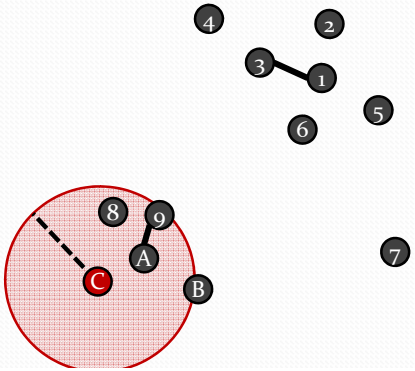

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | |
| 8 | BORDER |
| 9 | CORE |
| A | CORE |
| B | BORDER |
| C | Border |



# Exemplo

Eps = 

MinPts = 5

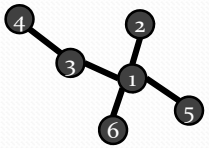


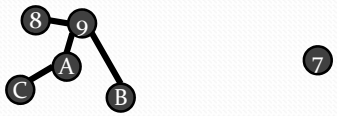

| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | |
| 8 | BORDER |
| 9 | CORE |
| A | CORE |
| B | BORDER |
| C | BORDER |

# Exemplo

**Eps =** 

**MinPts = 5**







| OBJETO | TIPO |
|---|---|
| 1 | CORE |
| 2 | BORDER |
| 3 | CORE |
| 4 | BORDER |
| 5 | BORDER |
| 6 | BORDER |
| 7 | NOISE |
| 8 | BORDER |
| 9 | CORE |
| A | CORE |
| B | BORDER |
| C | BORDER |



# Custo Computacional

- Complexidade?
  - Percorrer cada objeto para detectar se é **core**, **border** ou **noise**: O(*N*).
  - Para cada objeto, identificar os outros objetos que estão a uma distância máxima "*Eps*": O(*N*).
- Portanto: O($N^2$).



# Custo Computacional

- Mas se for possível identificar os objetos que estão num raio "*Eps*" de maneira mais eficiente que O(*N*)?



# Kd-Trees

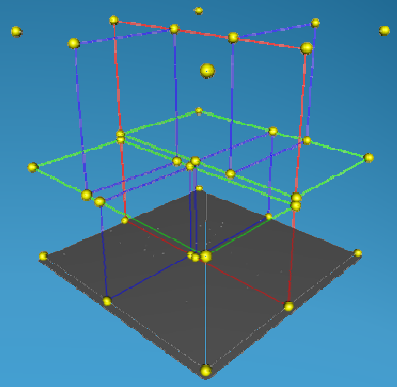
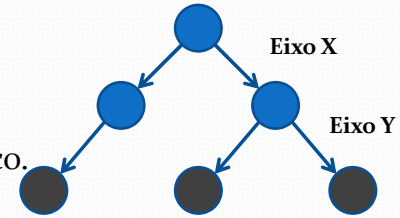


Figura: Wikipedia: http://en.wikipedia.org/wiki/File:3dtree.png

# kd-Trees




- Árvore binária.
- Cada **nó** é um ponto no espaço.
- Cada **nó não-folha divide o subespaço** em dois com um hiperplano em **um eixo** (pode-se usar a mediana)**.**
- **Sub-árvore esquerda** (direita) representa **os pontos** que estão **a esquerda** (direita) do hiperplano.
- Os **eixos** para geração dos hiperplanos são escolhidos de maneira **intercalada**: x, y, z, x, y, z, x, ... (Existem variações desta técnica).

*H. Samet, 2006*

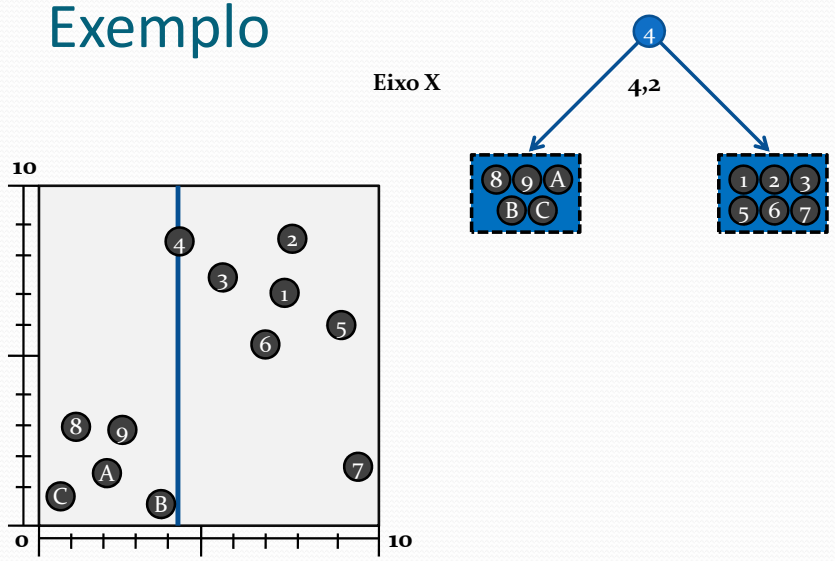


# Algoritmo de Construção

**Função: kdtree** (pontos, altura)
- **Se** pontos = vazio
  - **retorna** nulo
- **senão**
  - eixo = altura mod QT_DIM
  - mediana = selecionar_mediana(pontos, eixo)
  - criar Nó = mediana
  - Nó.esquerda = **kdtree**(pontos < mediana, altura+1)
  - Nó.direita = **kdtree**(pontos > mediana, altura+1)
  - **retorna** Nó

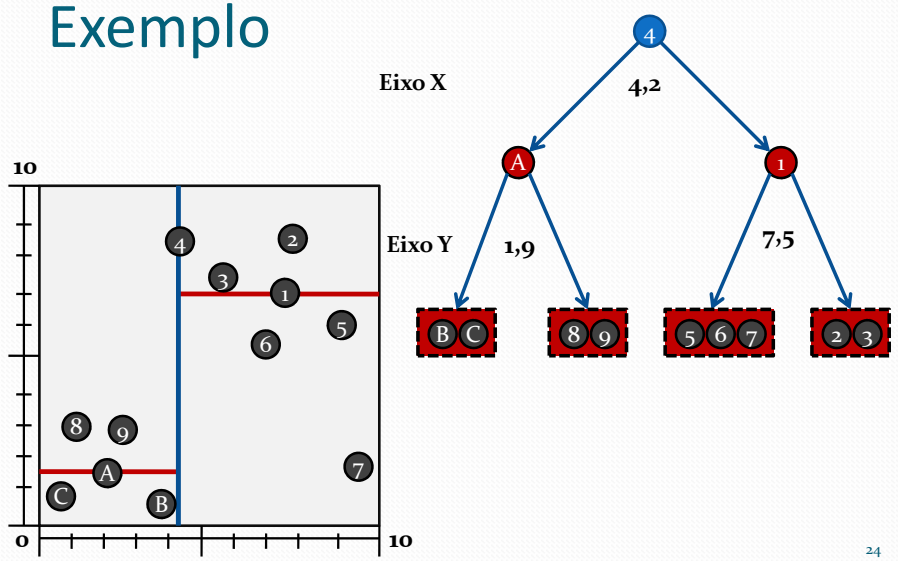


# Exemplo

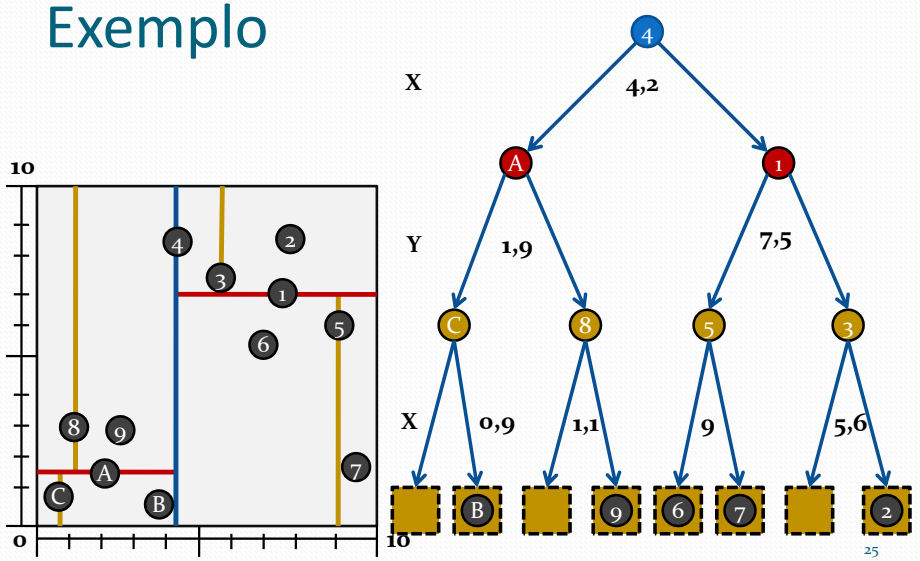





# Exemplo

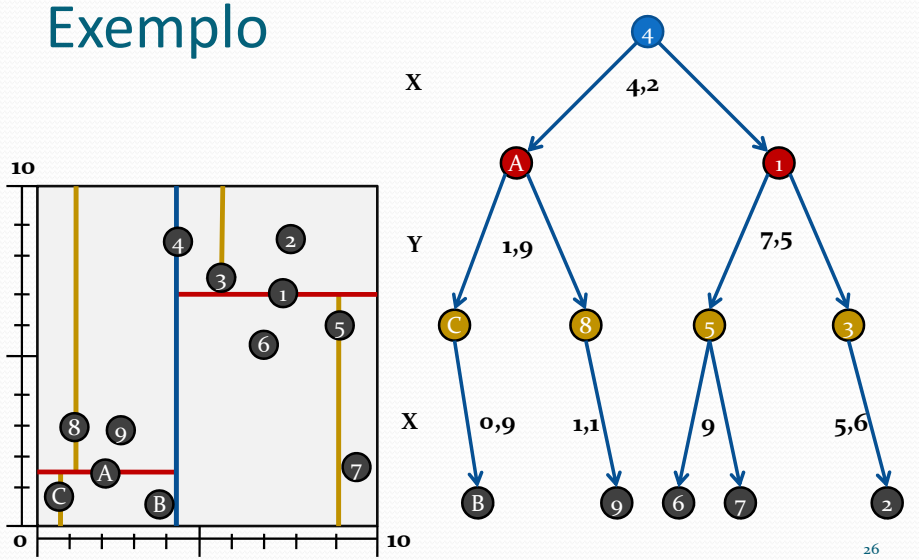


# Exemplo






# Exemplo






# Custo Computacional

- Custo de construção da kd-tree: $N \log_2 N$.
- Altura da árvore: $\log_2 N$.
  - Se for usado a mediana para construção do hiperplano.
- Logo, complexidade de encontrar um objeto é $\log_2 N$.



# Custo Computacional

- Mas é possível identificar os objetos que estão num raio "*Eps*" de maneira mais eficiente que O($N$)?

# Busca Intervalar

**Função: busca**(Nó, Intervalo)

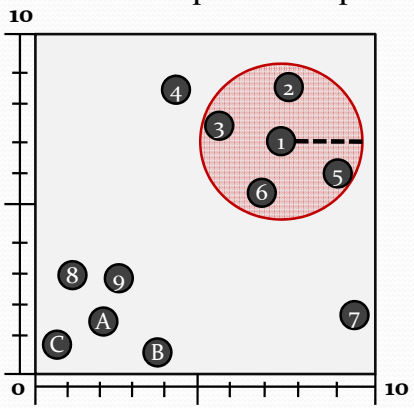
eixo = altura mod QT_DIM
visita(Nó)

**se** Nó.valor(eixo) > Intervalo.valor(eixo).menor
**busca**(Nó.esquerda, Intervalo)
**se** Nó.valor(eixo) < Intervalo.valor(eixo).maior
**busca**(Nó.direita, Intervalo)



# Busca Intervalar

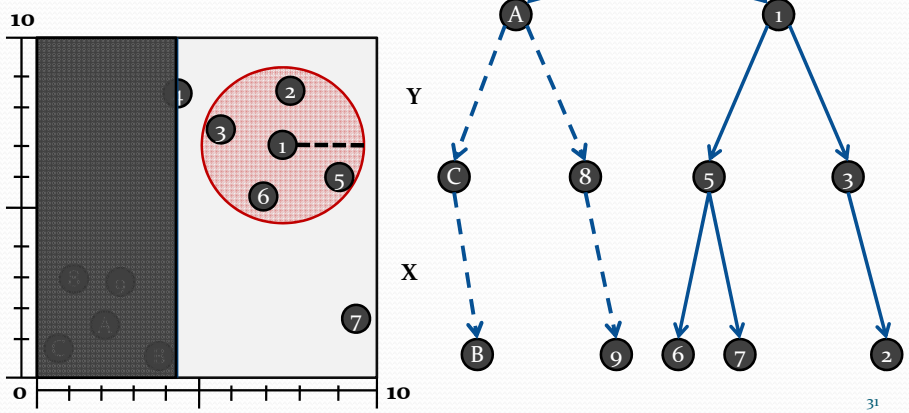
- É possível detectar os subespaços (hiperplanos) que "interceptam" a hiper-esfera com raio *Eps*.




- Exemplo:
  - Hiper-esfera centrada no "Objeto 1" com raio 2,2.
  - Limites da esfera:
    - X: [5, 9,4]
    - Y: [4,8, 9,2]

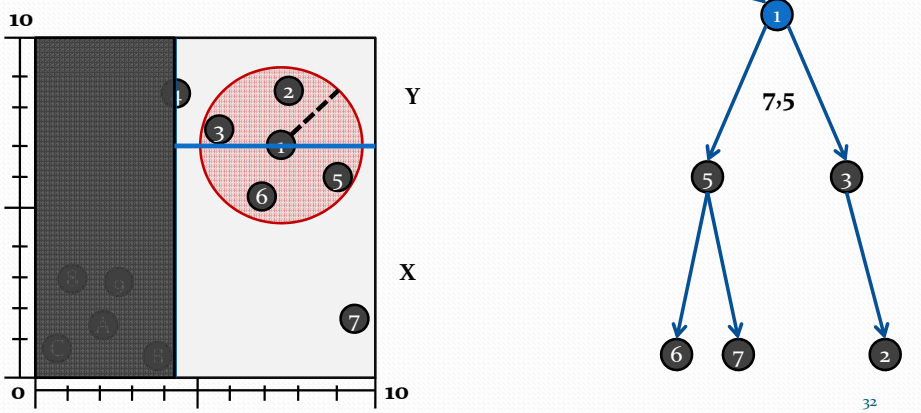


- Limites da esfera:
  - X: [5, 9,4]
  - Y: [4,8, 9,2]

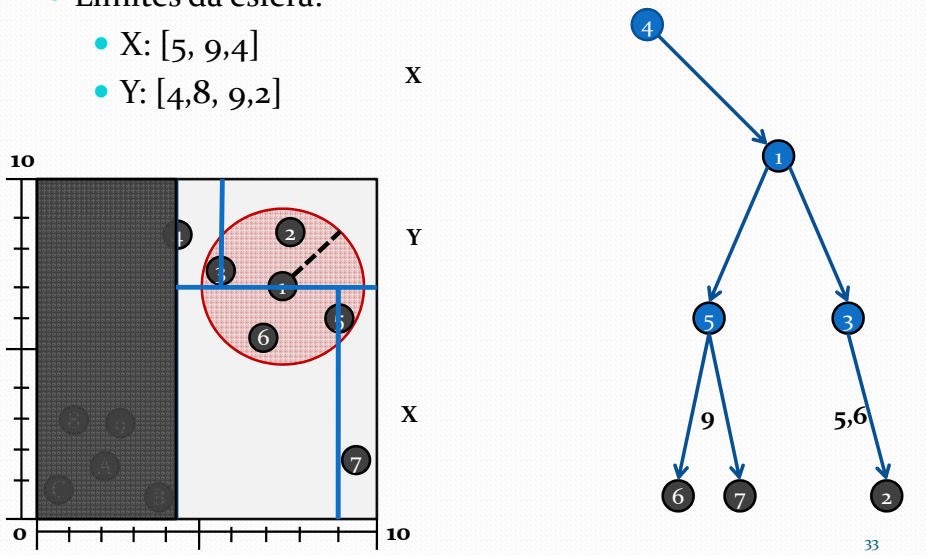





- Limites da esfera:
  - X: [5, 9,4]
  - Y: [4,8, 9,2]

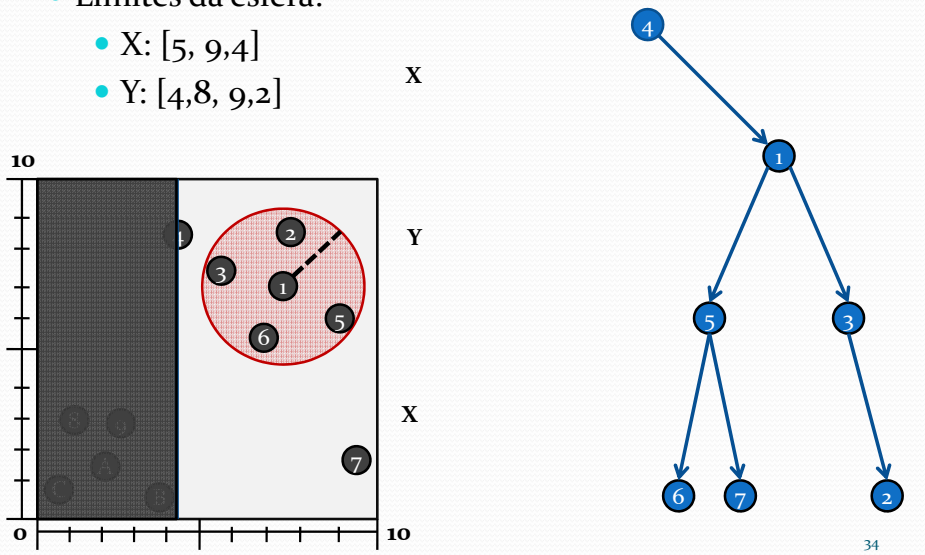


- Limites da esfera:
  - X: [5, 9,4]
  - Y: [4,8, 9,2]





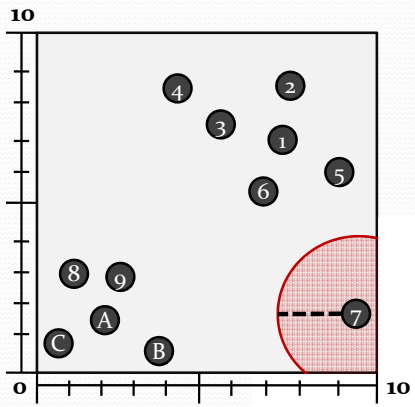

- Limites da esfera:
  - X: [5, 9,4]
  - Y: [4,8, 9,2]






# Exemplo 2

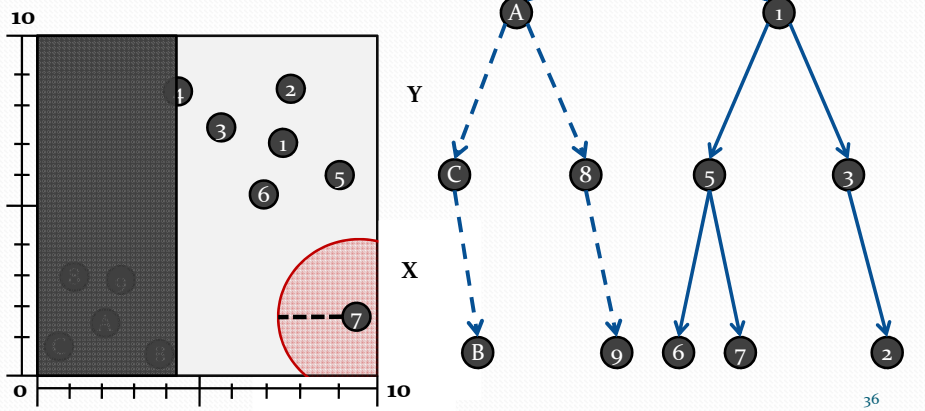
- Hiper-esfera centrada no "Objeto 7" com raio 2,2.



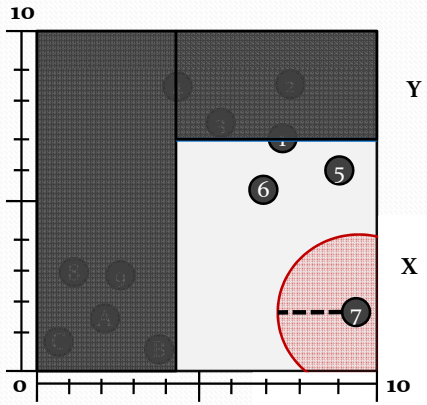

- Limites da esfera:
  - X: [7,3, 11,7]
  - Y: [-0,3, 3,9]

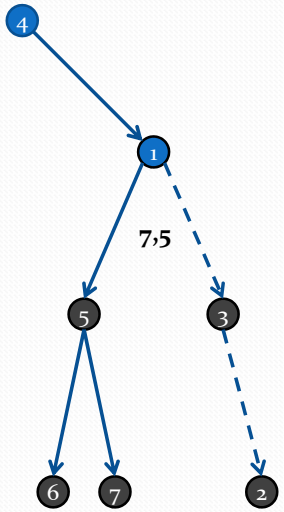


- Limites da esfera:
  - X: [7,3, 11,7]
  - Y: [-0,3, 3,9]



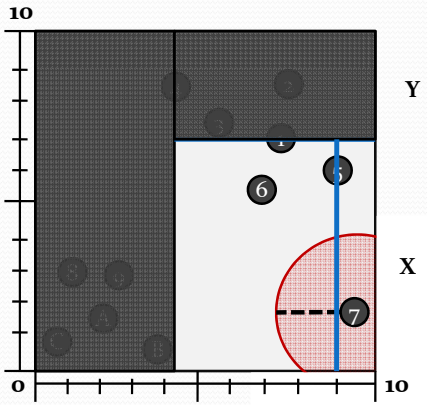
- Limites da esfera:
  - X: [7,3, 11,7]
  - Y: [-0,3, 3,9]

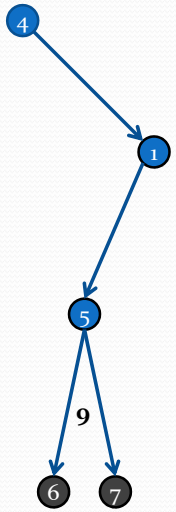




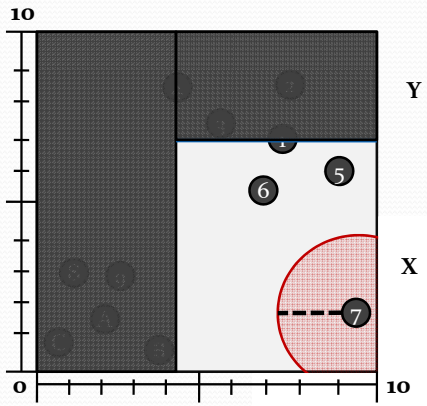

- Limites da esfera:
  - X: [7,3, 11,7]
  - Y: [-0,3, 3,9]

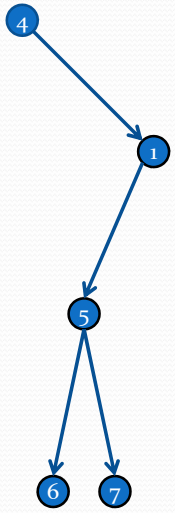





- Limites da esfera:
  - X: [7,3, 11,7]
  - Y: [-0,3, 3,9]






# Busca Intervalar

- A quantidade de objetos visitados depende do parâmetro *Eps*:
  - Quanto maior for o *Eps*, mais objetos serão visitados.
  - Se *Eps* = 0 (zero), então o custo computacional será $\log_2 N$.
  - Se *Eps* = Infinito, então o custo será *N*.
- Pode-se dizer que o custo é da ordem de $O(\log_2 N)$ para o caso médio.

# DBSCAN: Custo Computacional

- Qual a complexidade do DBSCAN usando kd-trees?
  - Percorrer cada objeto para detectar se é **core**, **border** ou **noise**: O(*N*).
  - Para cada objeto, identificar os outros objetos que estão a uma distância máxima "*Eps*": $O(\log_2 N)$ (caso médio).
- Portanto, na média: $O(N \log_2 N)$.
- OBS: Foi assumido distância Euclideana, mas pode ser adotadas outras distâncias.



# Outras Estruturas

R*-Trees

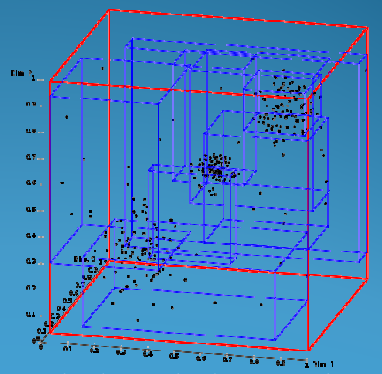
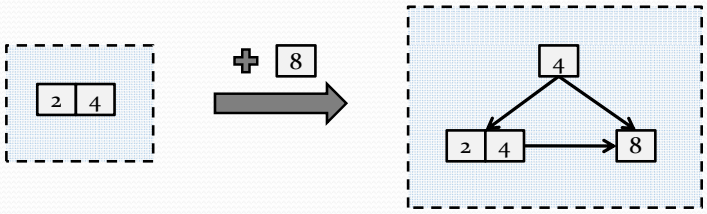


Figura: Wikipedia: http://en.wikipedia.org/wiki/File:RTree-Visualization-3D.svg



# R*-Trees

- Baseadas em árvores B+.
  - Ex: $M = 2$, $m = 1$

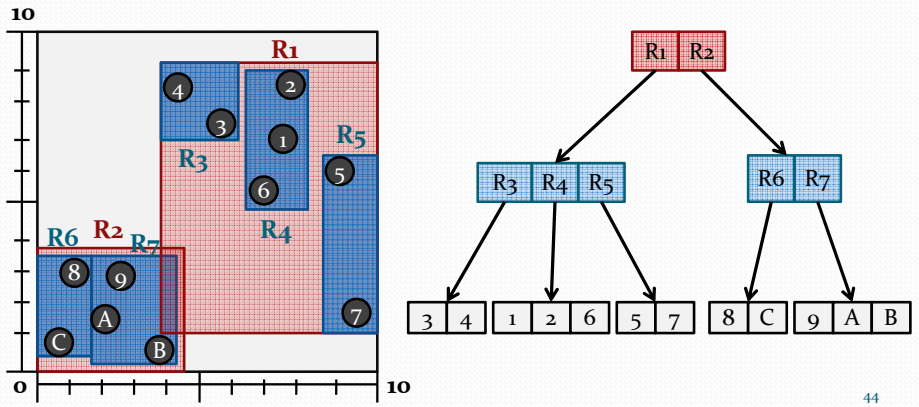



- Nós não-folha contém as chaves.
- Registros (pontos/objetos) somente nos nós folhas.

*N. Beckmann, 1990*

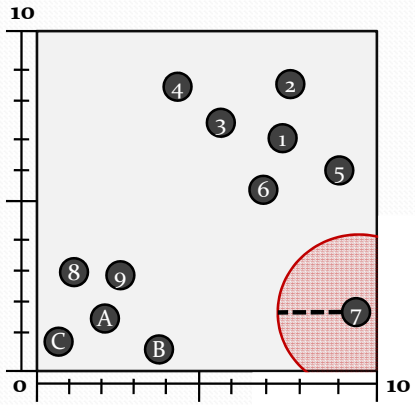


- Cada nó não-folha contém:
  - Endereço de um nó filho.
  - Área do retângulo que cobre todos os nós filhos.



# Busca Intervalar

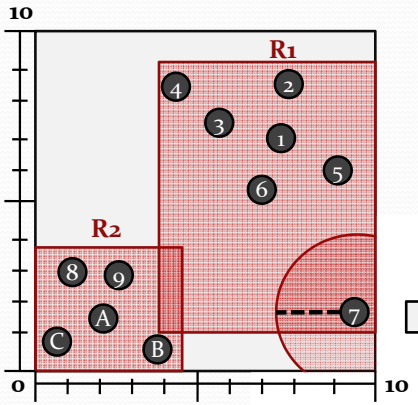
- Hiper-esfera centrada no "Objeto 7" com raio 2,2.

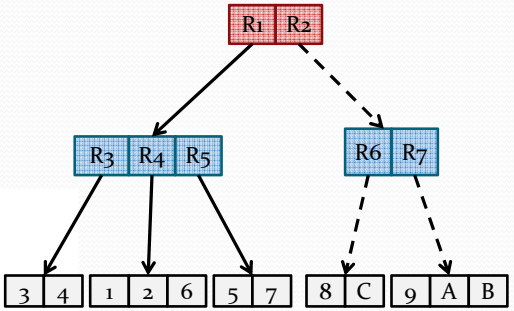



- Limites da esfera:
  - X: [7,3, 11,7]
  - Y: [-0,3, 3,9]



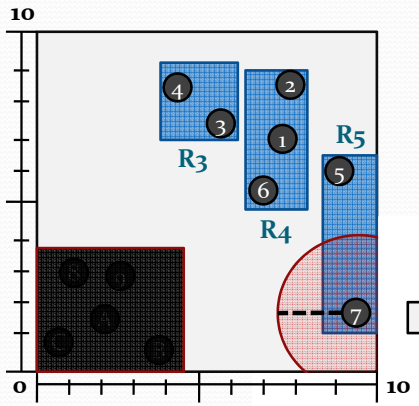
# Busca Intervalar

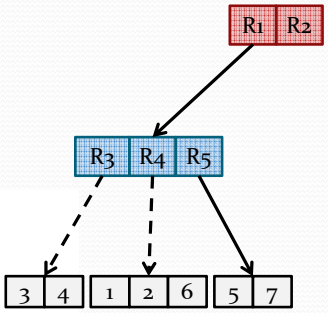




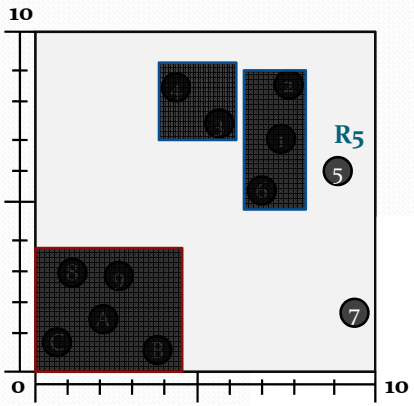

# Busca Intervalar

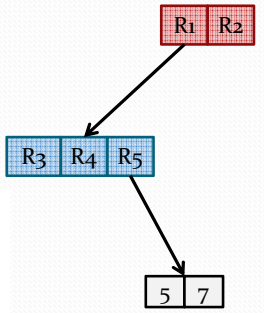





# Busca Intervalar



# Complexidade

- Custo de construção da R*-tree: $N \log_m N$.
- Custo da busca intervalar:
  - A quantidade de objetos visitados depende dos parâmetros *Eps*, *M* e *m*.
  - Novamente, pode-se dizer que o custo é da ordem de $O(\log_m N)$ para o caso médio.
- Portanto, a complexidade do DBSCAN usando R*-tree é na média: $O(N \log_m N)$.



# Bibliografia


- M. Ester, H-P Kriegel, J. Sander, X. Xu: "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", Proc. 2nd KDD, 1996, pp. 226-231.
- N. Beckmann, H-P Kriegel, R Schneider, B. Seeger: "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", Proc. ACM SIGMOD, 1990, pp. 322-331.
- H. Samet: "Foundations of multidimensional and metric data structures", Morgan Kaufmaan, 2006.
- D. T. Lee, C. K. Wong: "Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees, Acta Informatica, 1977, pp. 23-29.




# Dúvidas


Lucas Vendramin
vendra@grad.icmc.usp.br

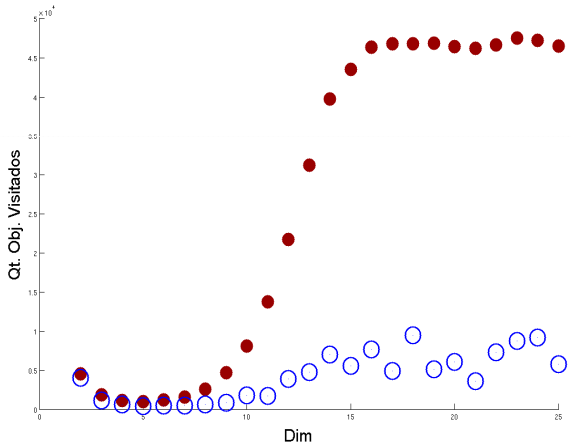



# kd-Tree com DBSCAN

- Na verdade, a "busca intervalar" pode ser modificada para "contar" os objetos que estão dentro da hiper-esfera:
  - Neste caso, se a quantidade de objetos for menor que MinPts, então a busca pode parar.

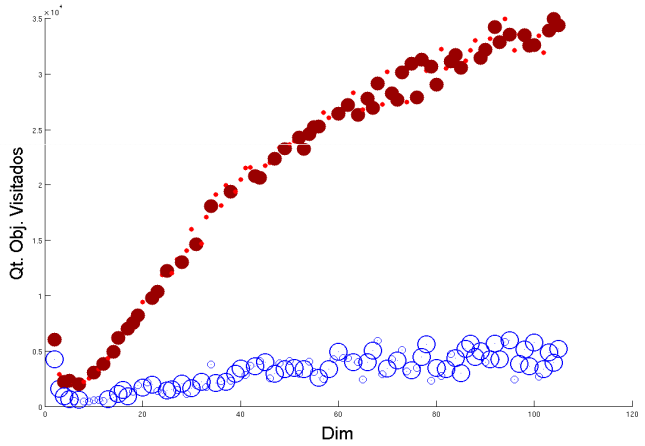
# Dimensões

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






# Dimensões






# R*-Tree - Inserção

Onde alocar o objeto E? Qual sub-retângulo?

- **ChooseSubtree:**
  - **IF** childpointers point to leaves **THEN**
    - determine the minimum overlap/area cost.
  - **IF** do not point to leaves **THEN**
    - determine the minimum area cost.
- **IF** N has less than *M* entries **THEN**
  - OK. Accommodate E in N.

Folha "estourou"

- **ELSE**
  - **IF** level is not root and is the first time **THEN**
    - ***REINSERT***

Elimina da árvore e reinsere

  - **ELSE** (*Split*):
    - ChooseSplitAxix (minimum margin).
    - ChooseSplitIndex (minimum overlap/area).

Divide o nó folha em 2