Faculdade de Informática

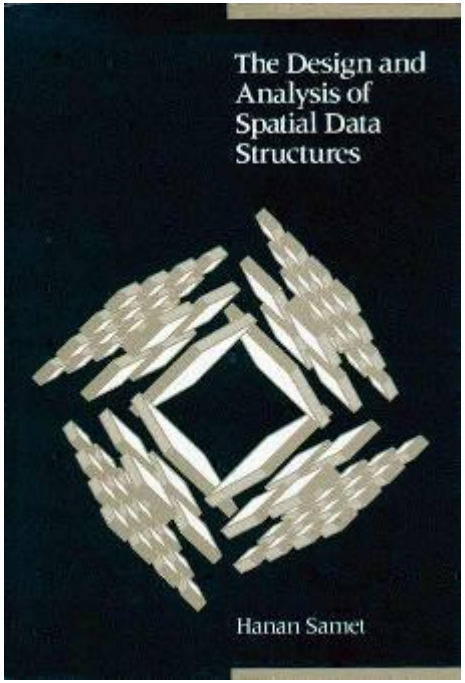
PUCRS

Algoritmos e Estruturas de Dados II
2023-2

Aula #13 – Estruturas de Dados Espaciais

Prof. Leonardo Heredia

Referências



Dados Escalares x Espaciais

Dados Escalares

Grandezas escalares
Valor numérico associado
Comprimento
Massa
Tempo
Nota
Preço

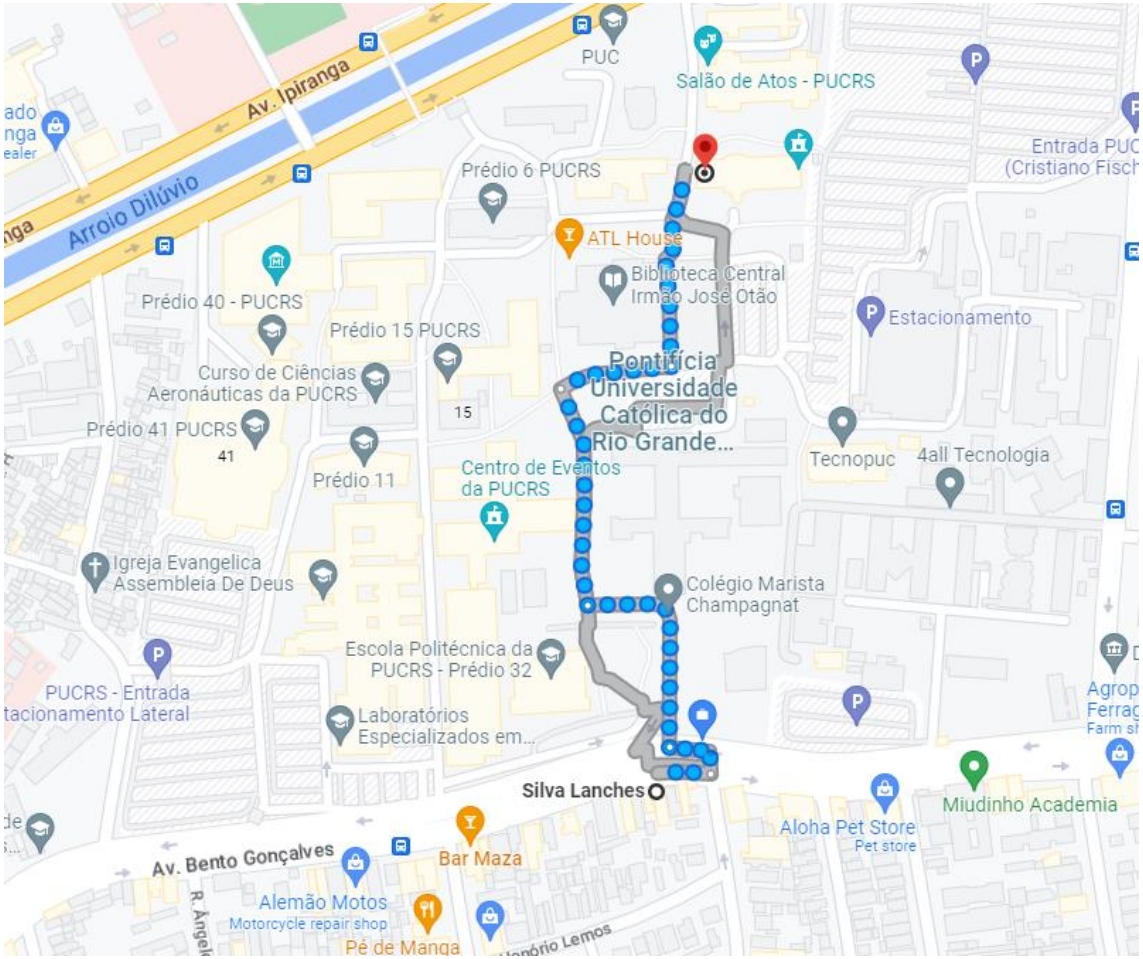
Tipos de operações:
Média, soma, valor mais alto, etc.

Dados Espaciais

Informações sobre elementos em um espaço.
Localização de um posto.
Localização de uma cidade.
Trecho de um rio.
Contorno de uma lagoa.

Tipos de operações:
Cidade mais próxima, perímetro do lago, área de um país, farmácias mais próximas.

Dados Espaciais



Caminho até o Silvas

Representação de Dados Espaciais

Representação Vetorial

Dados discretos.
Entidades do mundo real, como ruas, localização de casas, buracos, contorno de rios, entre outros são representados por entidades geométricas discretas, tais como pontos, linhas e polígonos.
As coordenadas são referenciadas através de um sistema de coordenadas geográficas ou cartesianas.

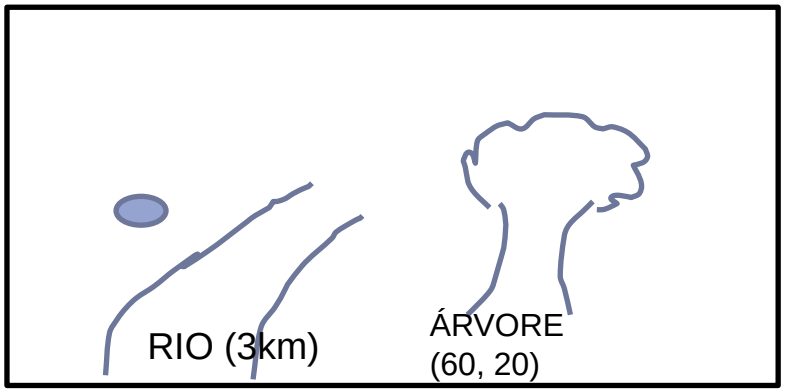
Representação Matricial *raster*

Espaço dividido em células.
São imagens.
Cada célula possui uma informação independente.
A quantidade de células para um determinado espaço define a resolução.

Representação de Dados Espaciais

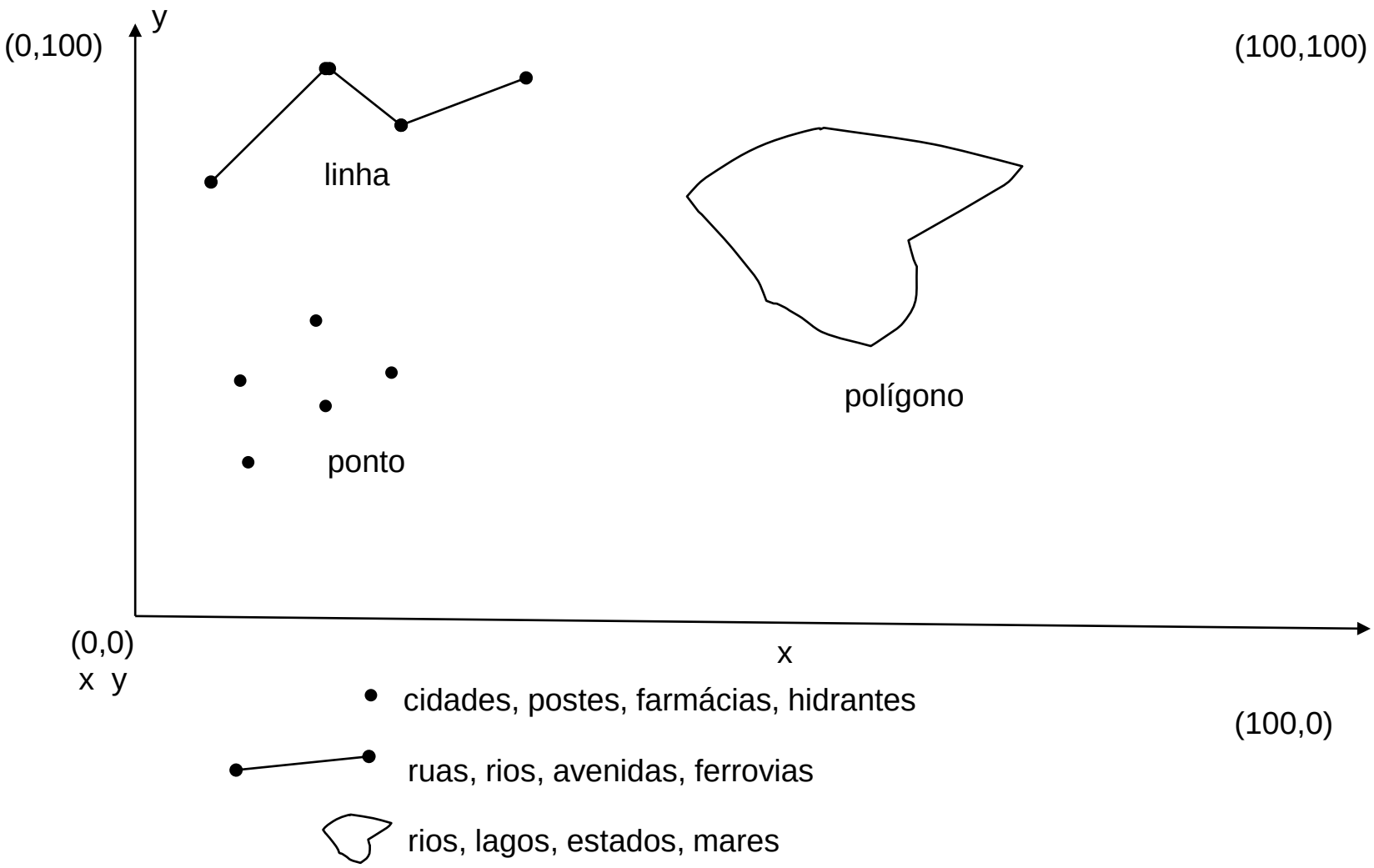
Representação Vetorial

Representação Matricial
raster



EXEMPLO MUUUUITO SIMPLES

Dados Vetoriais



Estruturas de Dados Espaciais

- Quadtree
  - Quadtree de pontos (vetorial)
  - Quadtree de região (matricial)
- K-d-tree
  - 2d tree
- R-tree

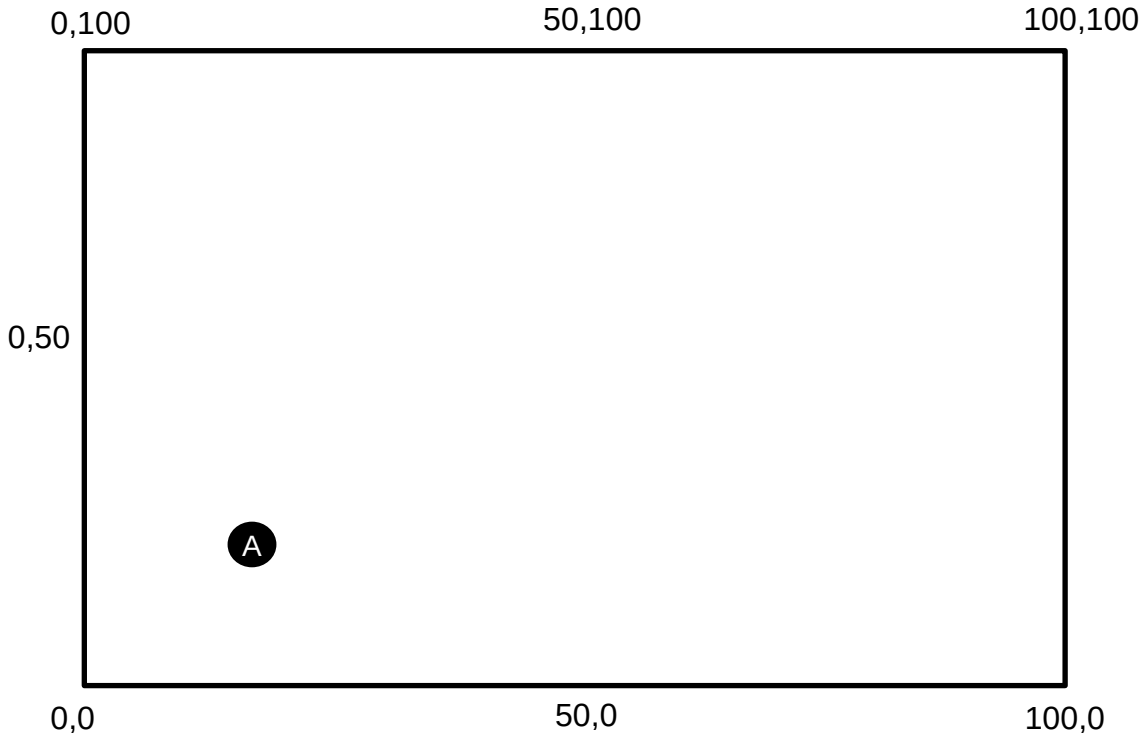
- Grid

Quadtree de Pontos

- Utilizada para armazenar em forma de árvore pontos em um plano bidimensional (x,y).
- A cada ponto inserido o plano é dividido em quatro quadrantes e o ponto é adicionado a um dos nodos. Esse processo ocorre recursivamente.
- A árvore final vai depender da ordem em que os pontos foram inseridos.

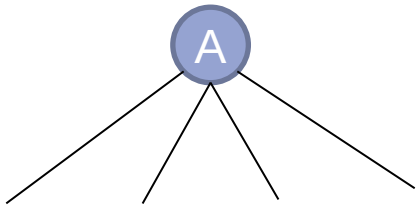
Quadtree de Pontos

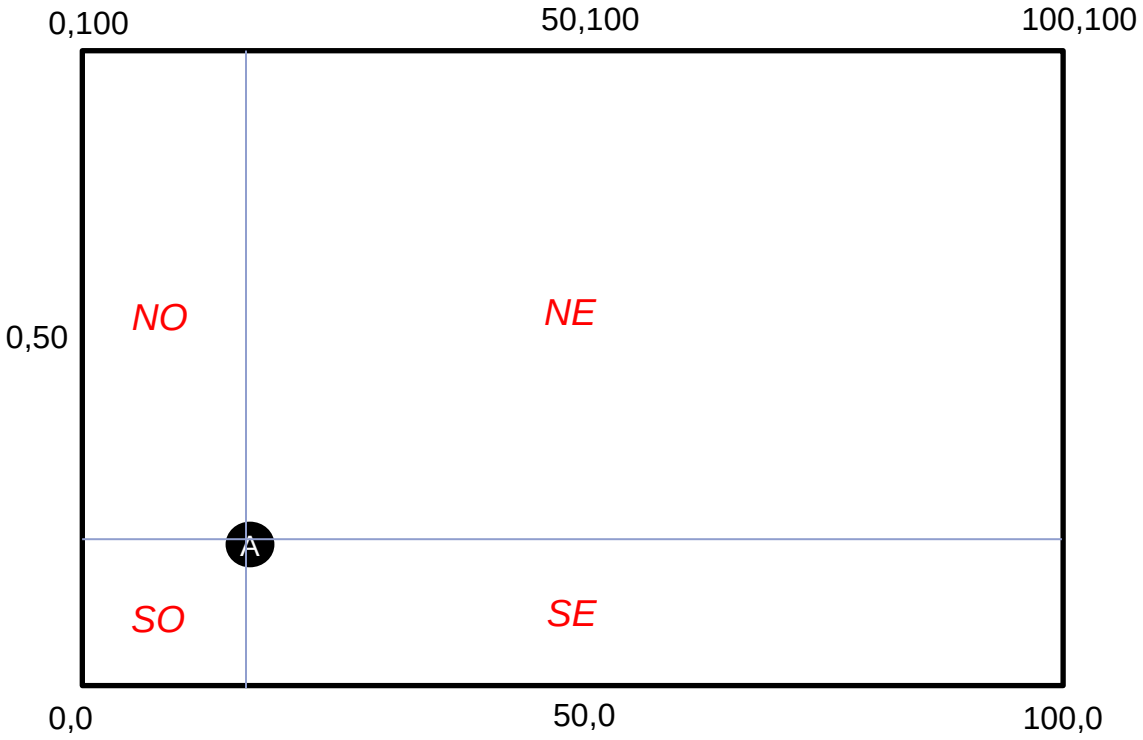
A(10,10)



Quadtree de Pontos

A(10,10)

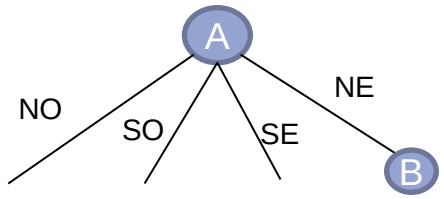


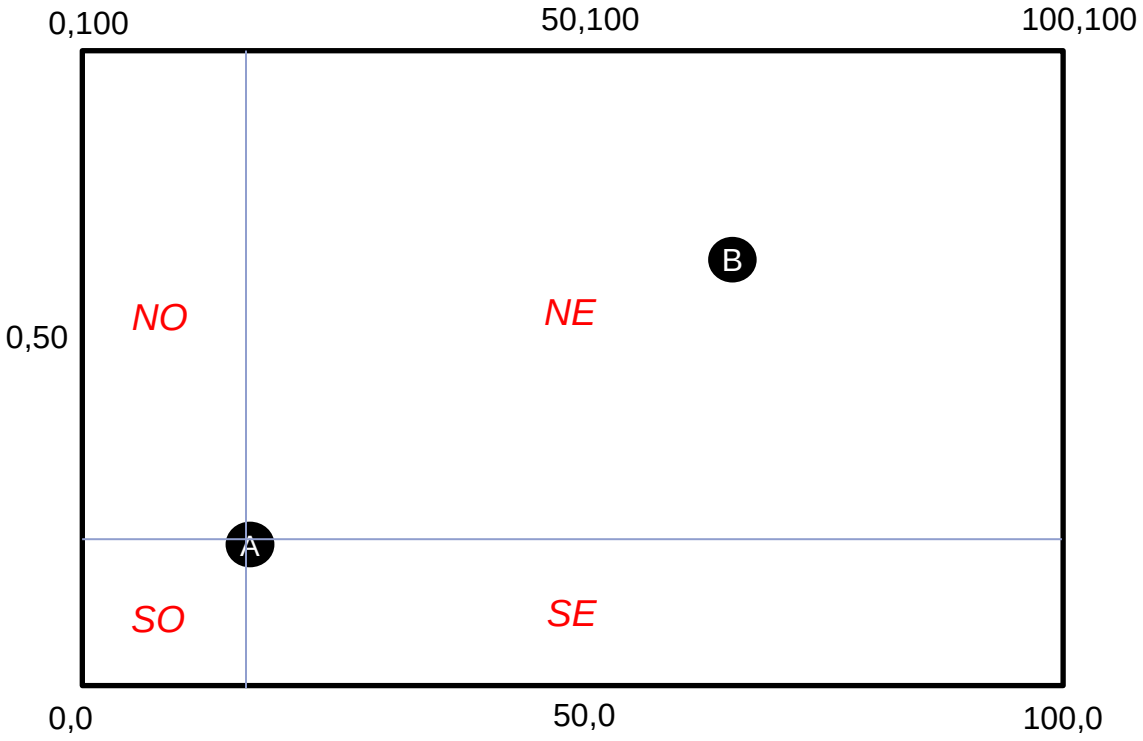


Quadtree de Pontos

A(10,10)

B(70, 60)

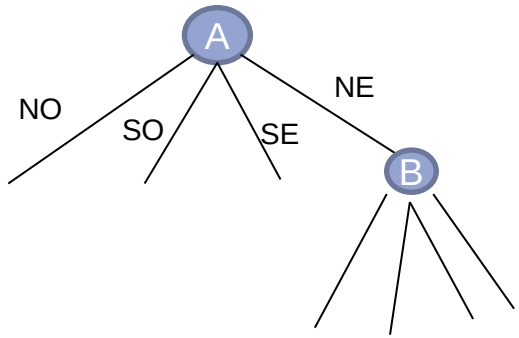


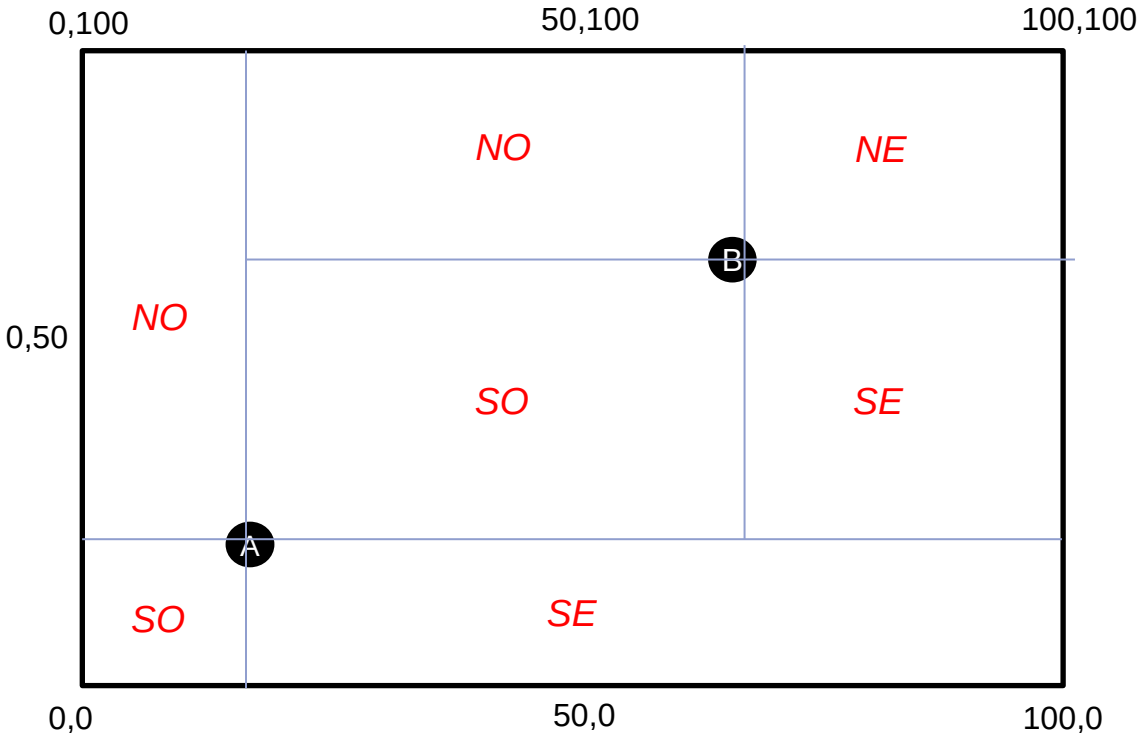


Quadtree de Pontos

A(10,10)

B(70, 60)



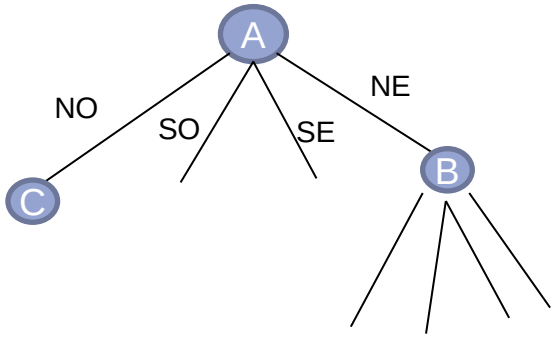


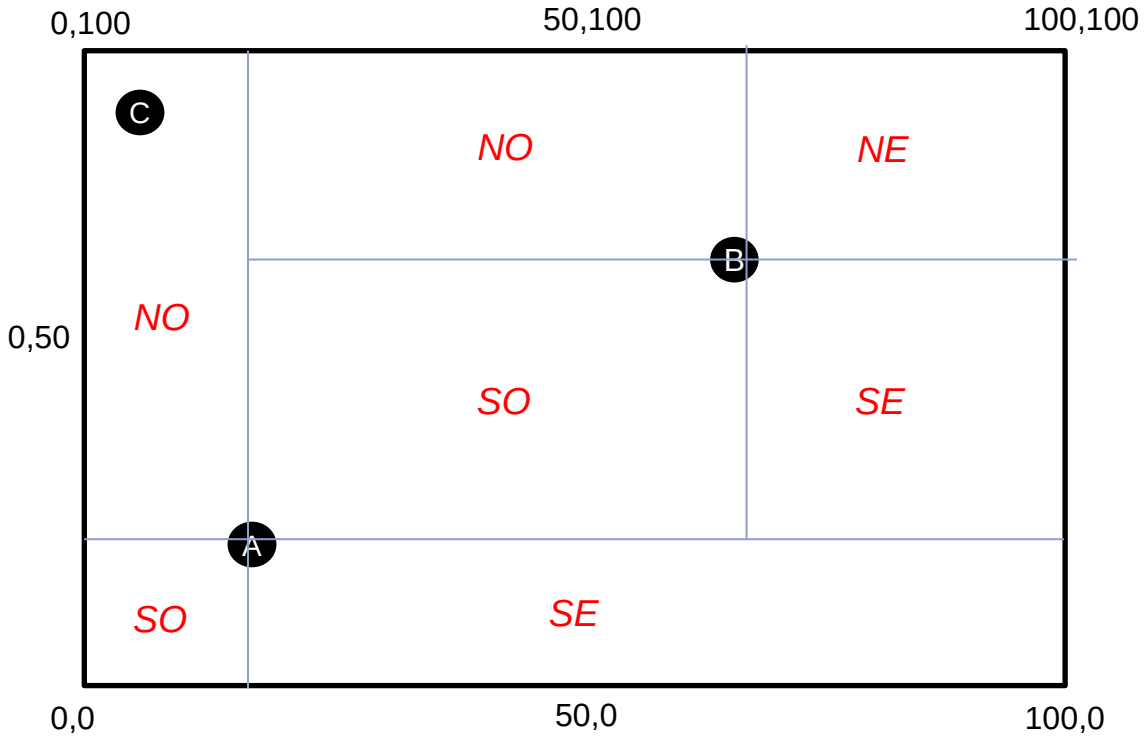
Quadtree de Pontos

A(10,10)

B(70, 60)

C(05, 90)



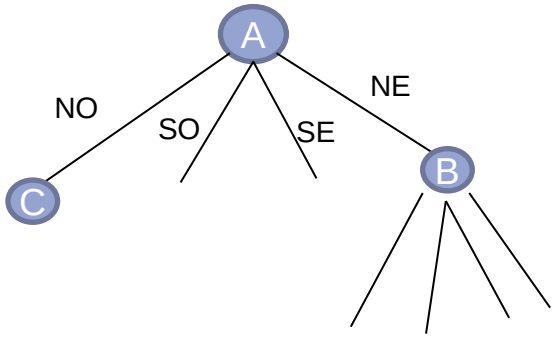


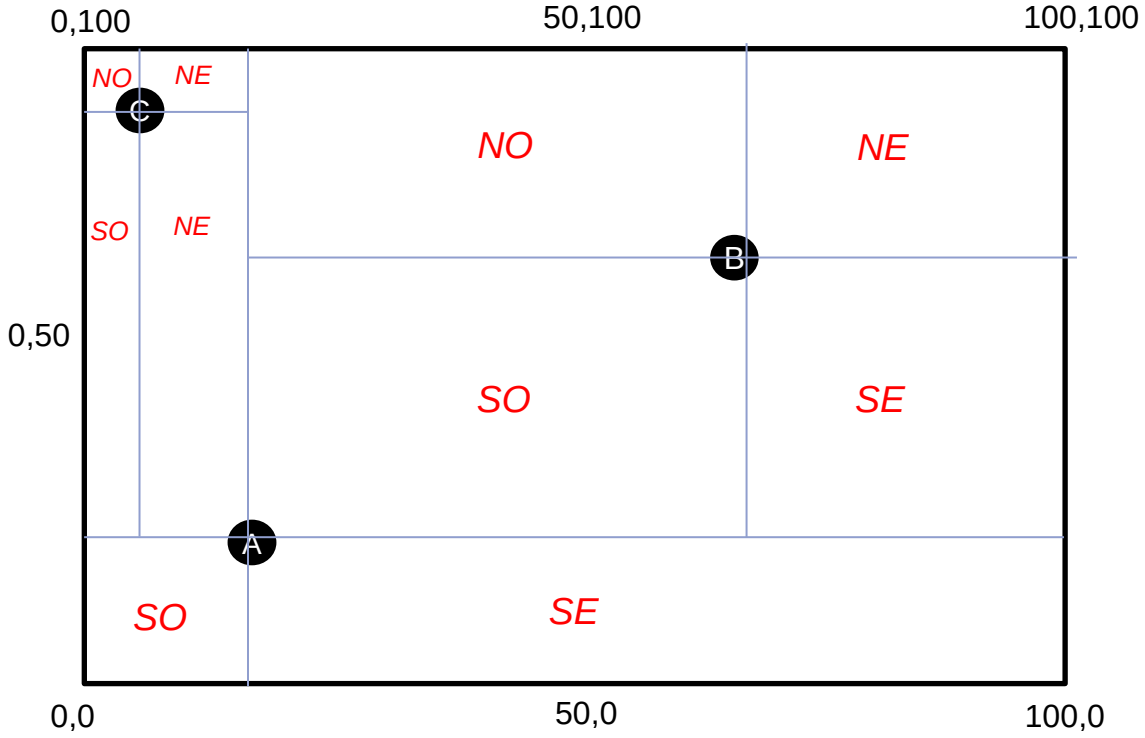
Quadtree de Pontos

A(10,10)

B(70, 60)

C(05, 90)





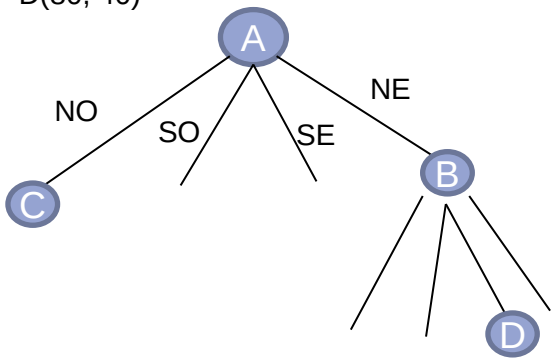
Quadtree de Pontos

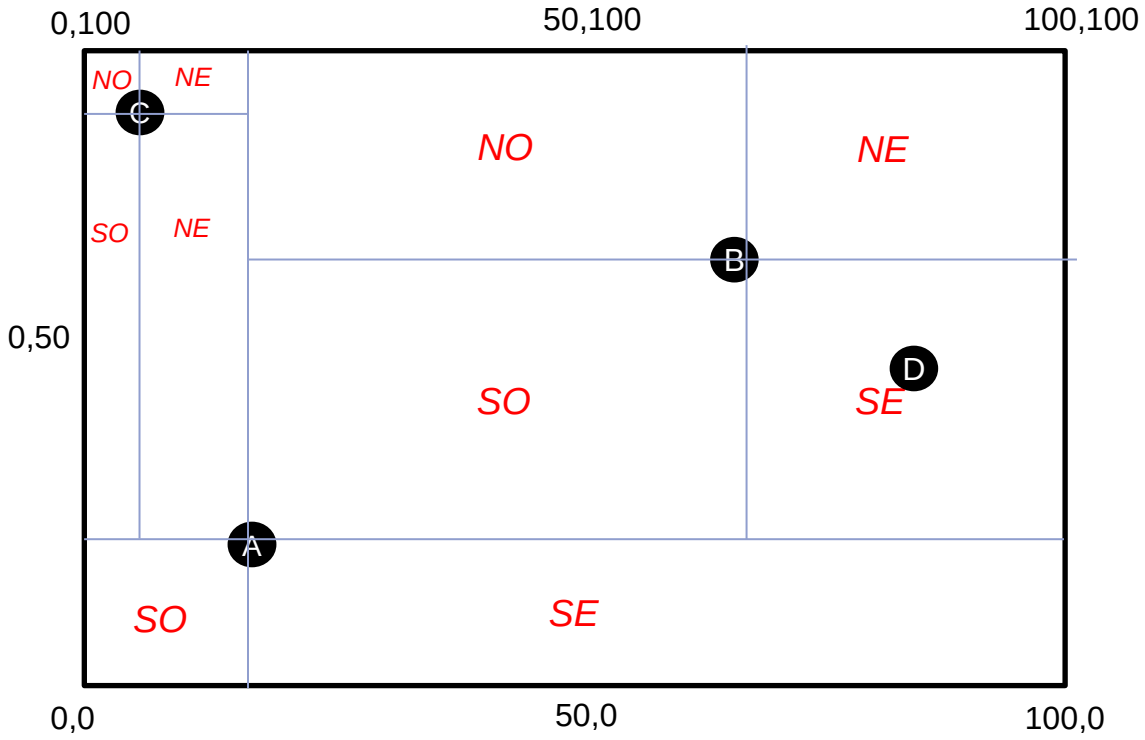
A(10,10)

B(70, 60)

C(05, 90)

D(80, 40)





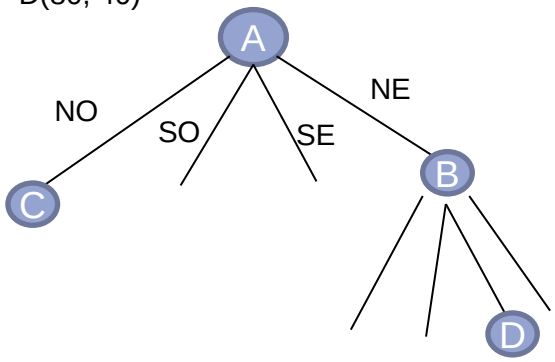
Quadtree de Pontos

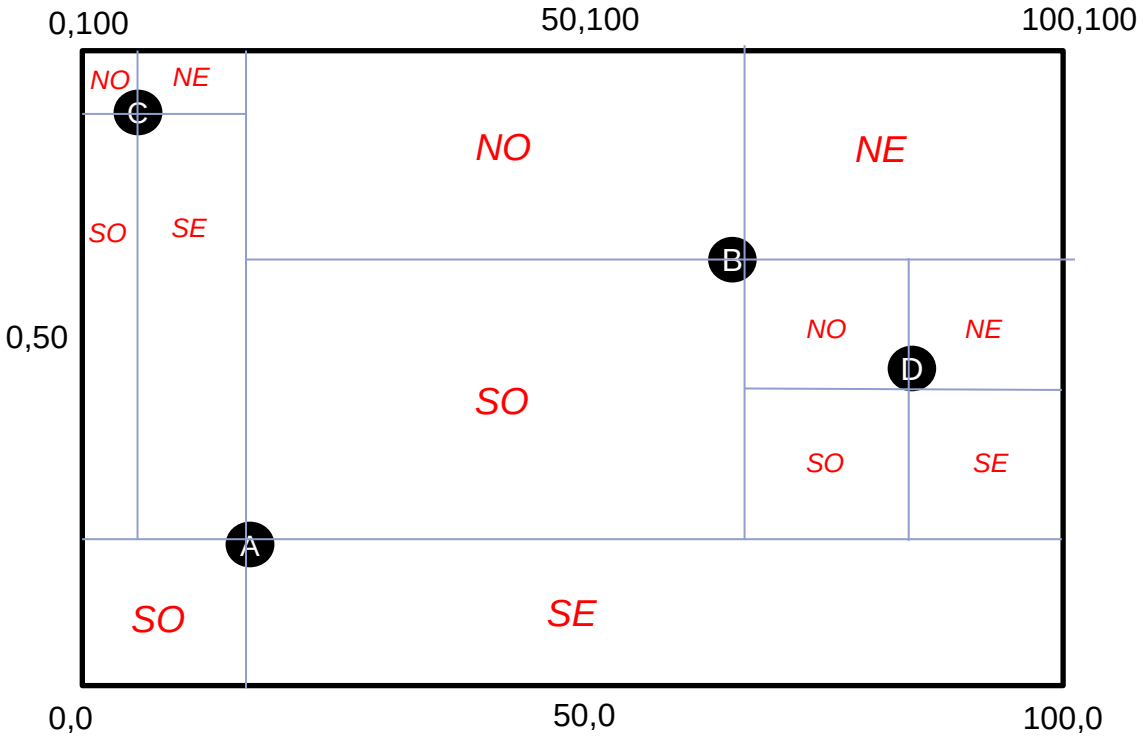
A(10,10)

B(70, 60)

C(05, 90)

D(80, 40)

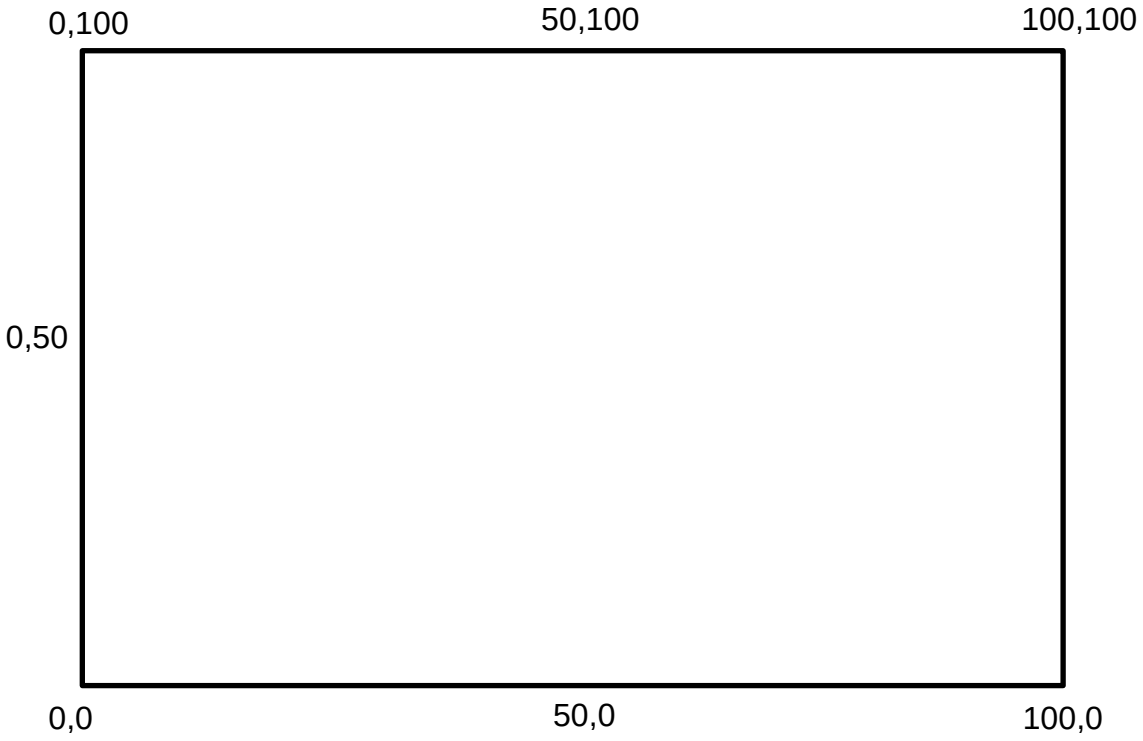




Exercício

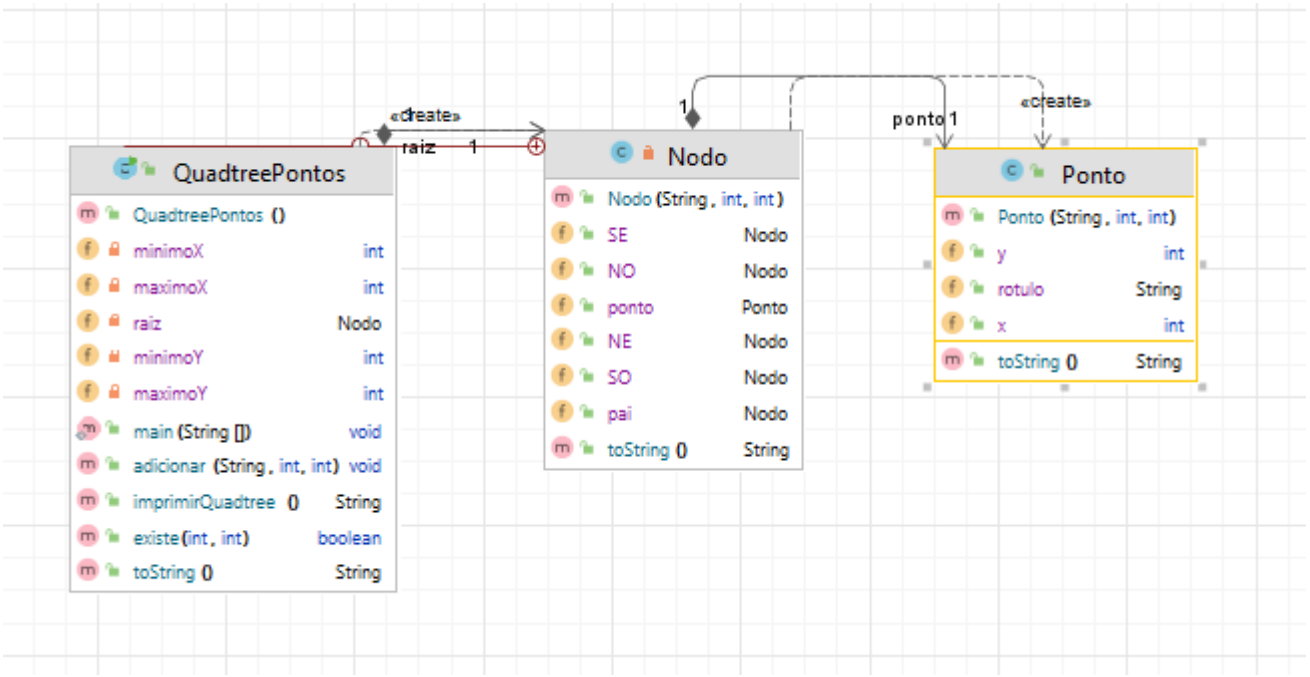
Construir a quadtree para os seguintes pontos:

A(30,20)
B(70,70)
C(90,10)
D(10,10)
E(60,40)
F(40,80)



Desenhar a árvore!

Implementação



Implementação

```
public class Ponto {
    public String rotulo;
    public int x;
    public int y;
    public Ponto(String rotulo, int x, int y) {
        this.rotulo = rotulo;
        this.x = x;
        this.y = y;
    }
    @Override
    public String toString() {
        return rotulo + "(" + x + "," + y + ")";
    }

public boolean existe(int x, int y) {
    //(30, 35)
    Nodo navegador = this.raiz;
    while(navegador!=null) {
        //testa se o ponto existe
        if(x == navegador.ponto.x && y == navegador.ponto.y) return true;
        //verifica NO
        if(x<navegador.ponto.x && y>=navegador.ponto.y) navegador = navegador.NO;
        //verifica SO
        else if (x<navegador.ponto.x && y<navegador.ponto.y) navegador =
navegador.SO;
        //verifica SE
        else if (x>=navegador.ponto.x && y<navegador.ponto.y) navegador
=navegador.SE;
        //verifica NE
        else if (x>=navegador.ponto.x && y>=navegador.ponto.y) navegador =
navegador.NE;
    }
    return false;
```

```
public class QuadtreePontos {

    private class Nodo {
        public Ponto ponto;
        public Nodo NE;
        public Nodo SE;
        public Nodo NO;
        public Nodo SO;
        public Nodo pai;
        public Nodo(String rotulo, int x, int y) {
            this.ponto = new Ponto(rotulo, x,y);
        }
        @Override
        public String toString() {
            String str = ponto.rotulo + " = ";
            str+="{";
            if(this.NO==null) str += " NO ";
            else str += " " + NO.ponto.rotulo + " ";
            str+=",";
            if(this.SO==null) str += " SO ";
            else str += " " + SO.ponto.rotulo + " ";
            str+=",";
            if(this.SE==null) str += " SE ";
            else str += " " + SE.ponto.rotulo + " ";
            str+=",";
            if(this.NE==null) str += " NE ";
            else str += " " + NE.ponto.rotulo + " ";
            str+="}";
            return str;
        }
    }
    private Nodo raiz;
    private int minimoX = 0;
    private int minimoY = 0;
    private int maximoX = 100;
    private int maximoY = 100;
    public QuadtreePontos() {

    }
```

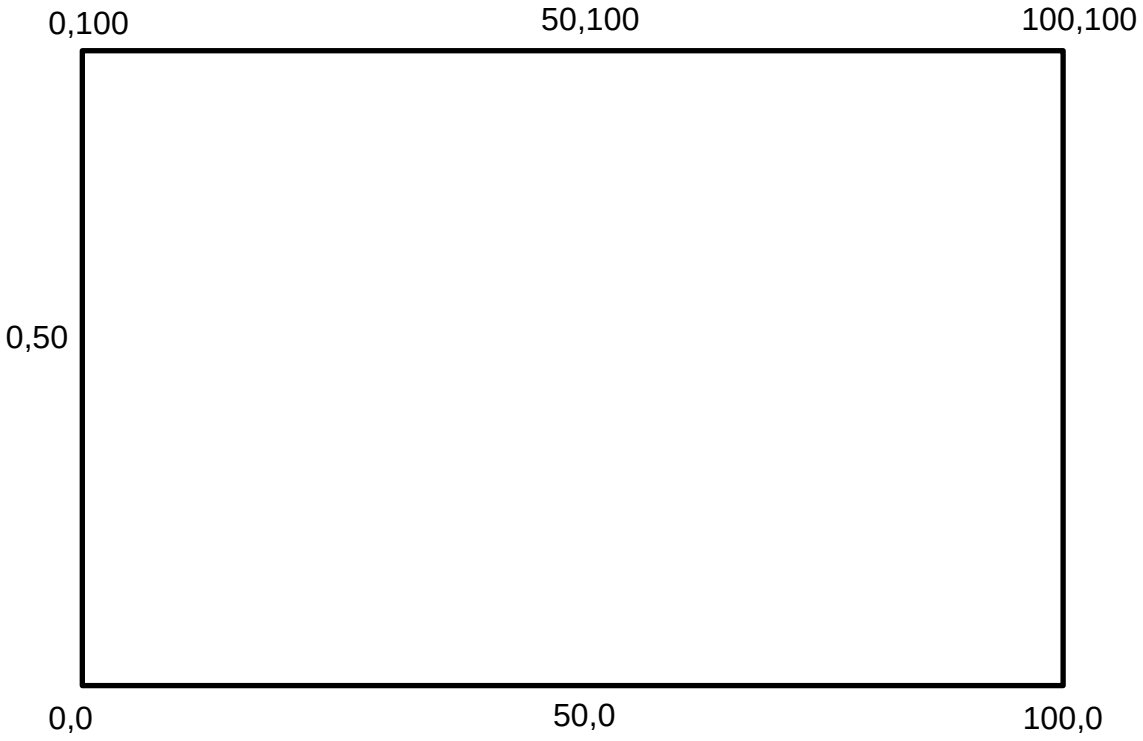
```
public void adicionar(String rotulo, int x, int y) {
   Nodo novoNodo = new Nodo(rotulo, x, y);
   if(this.raiz==null) {
      raiz = novoNodo;
   }
   else {
      Nodo explorador = raiz;
      while (explorador!=null) {
         if (novoNodo.ponto.x < explorador.ponto.x && novoNodo.ponto.y >= explorador.ponto.y){
            if(explorador.NO == null)  {
               explorador.NO = novoNodo;
               explorador = null;
            }
            else explorador = explorador.NO;
         }
         else if(novoNodo.ponto.x < explorador.ponto.x && novoNodo.ponto.y < explorador.ponto.y) {
            if(explorador.SO == null)  {
               explorador.SO = novoNodo;
               explorador = null;
            }
            else explorador = explorador.SO;
         }
         else if (novoNodo.ponto.x >= explorador.ponto.x && novoNodo.ponto.y >= explorador.ponto.y) {
            if(explorador.NE == null)  {
               explorador.NE = novoNodo;
               explorador = null;
            }
            else explorador = explorador.NE;
         }
         else if (novoNodo.ponto.x >= explorador.ponto.x && novoNodo.ponto.y < explorador.ponto.y)  {
            if(explorador.SE == null)  {
               explorador.SE = novoNodo;
               explorador = null;
            }
            else explorador = explorador.SE;
         }
      }
   }
}
```

K-d-Tree

- Árvore de busca binária eficiente para chaves multidimensionais.
- Utilizada em dados espaciais para armazenar e buscar pontos (X,Y).
- Nesse caso a árvore é uma 2D-Tree.
- Cada nível da árvore é ramificado de acordo com uma das dimensões. As dimensões vão sendo alternadas a cada nível.
- No caso de 2D-Tree a árvore vai sendo ramificada alternando entre a coordenada X e Y.
- Cada coordenada utilizada para dividir a árvore a cada nível é chamada de *discriminador*.
- Arbitrariamente o primeiro discriminador é o X, depois o Y, depois o X a assim sucessivamente.
- O forma final da árvore, assim como a quadtree, depende da ordem de inclusão dos pontos.

K-d-Tree

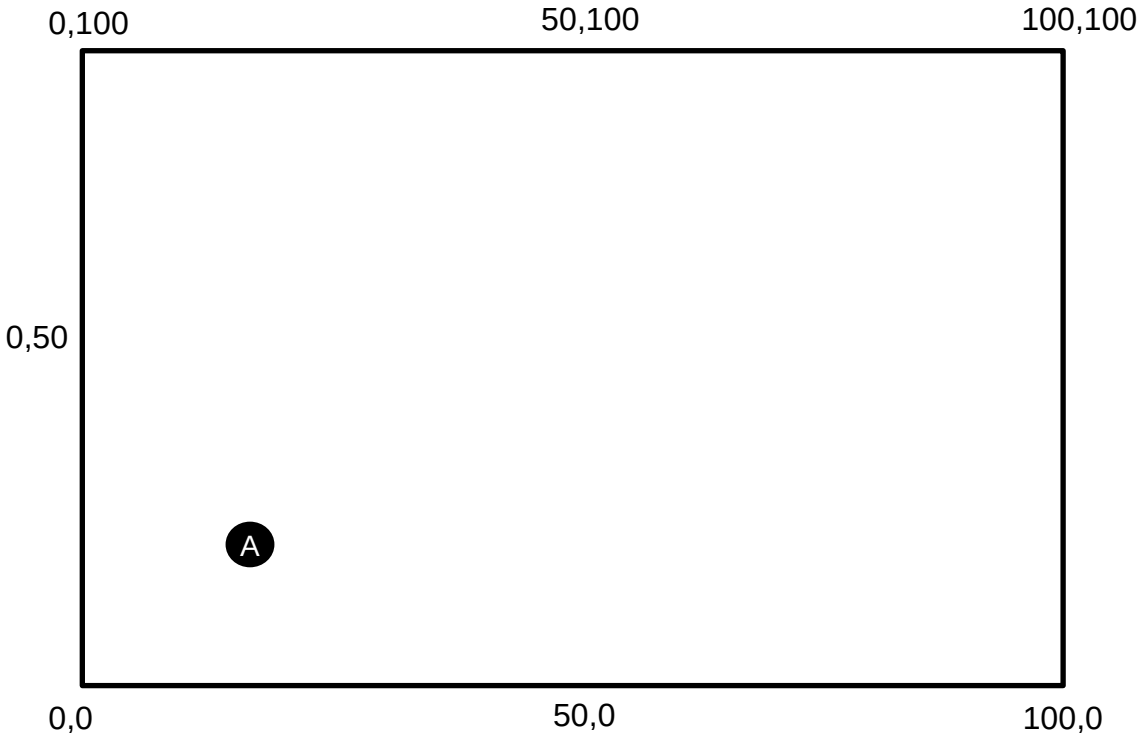
Pontos a serem inseridos



K-d-Tree

Pontos a serem inseridos

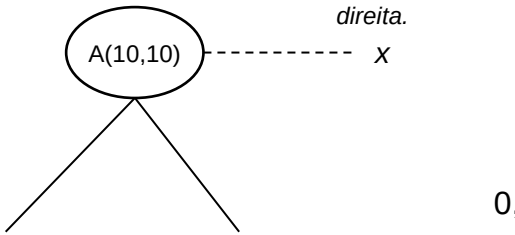
A(10,10)

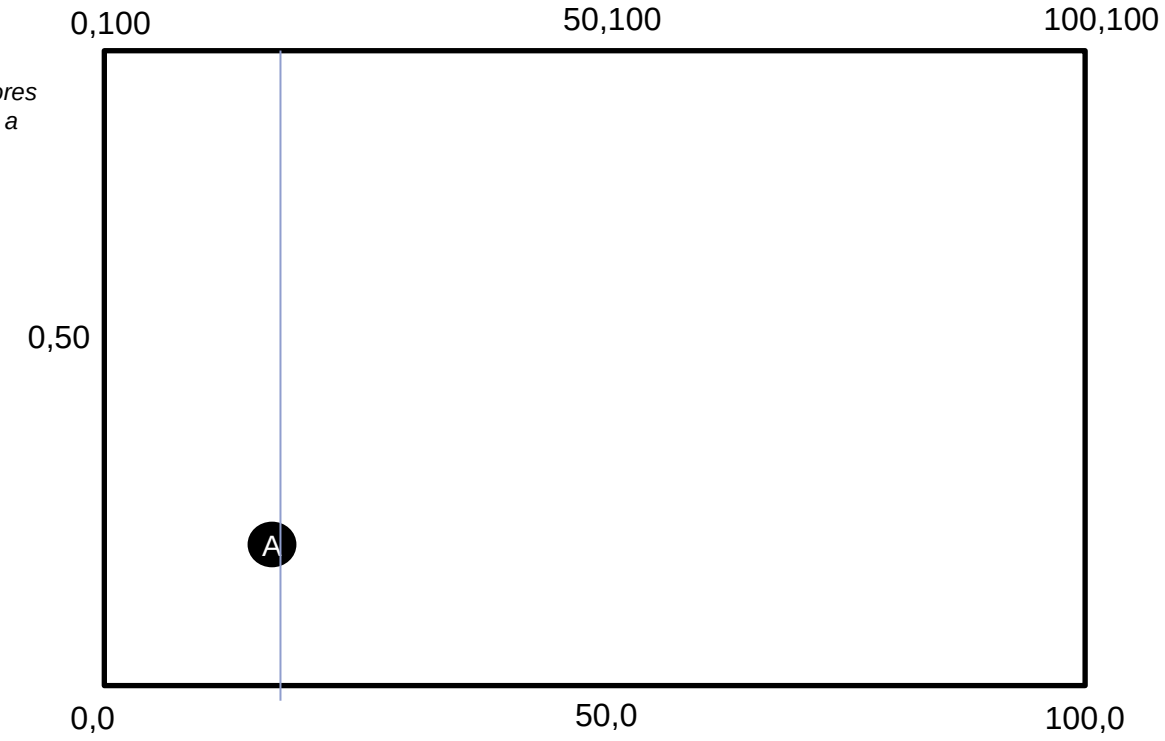


K-d-Tree

Pontos a serem inseridos

A(10,10)

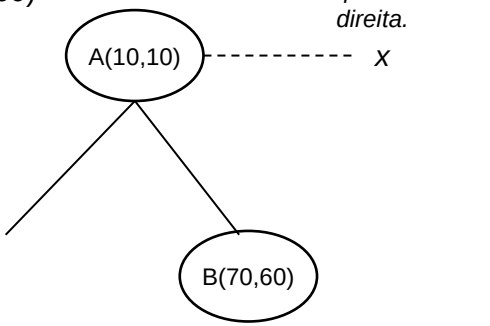


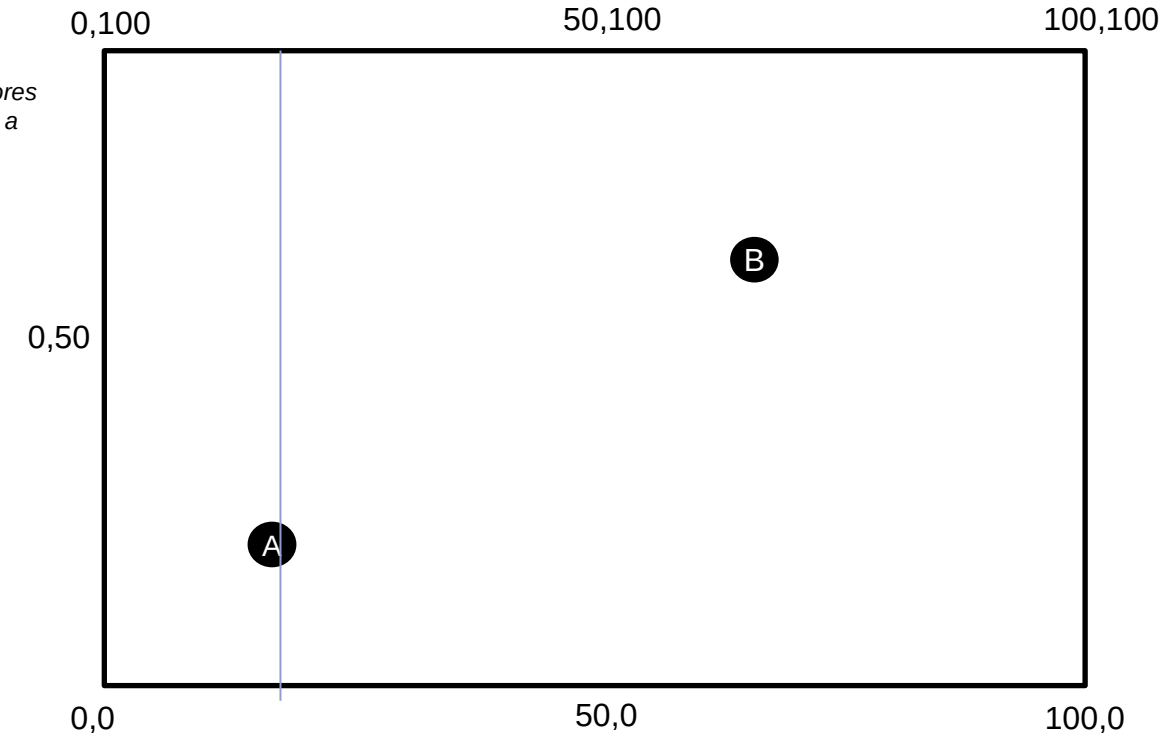


K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)

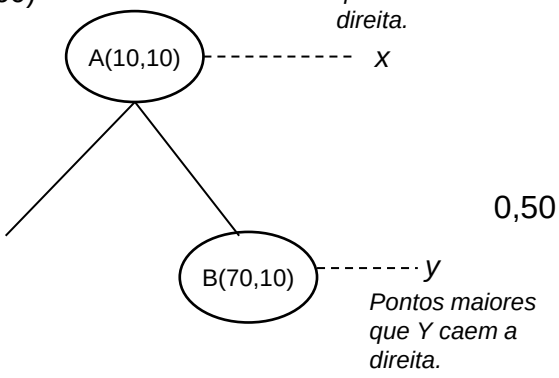


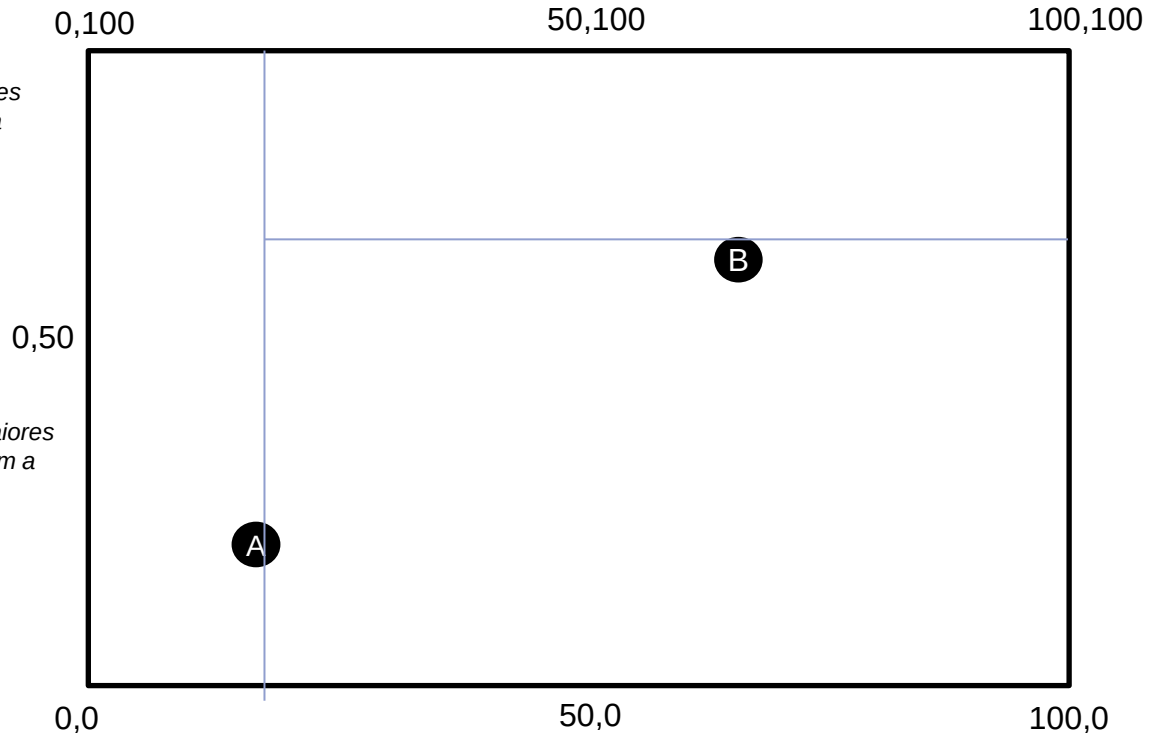


K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)

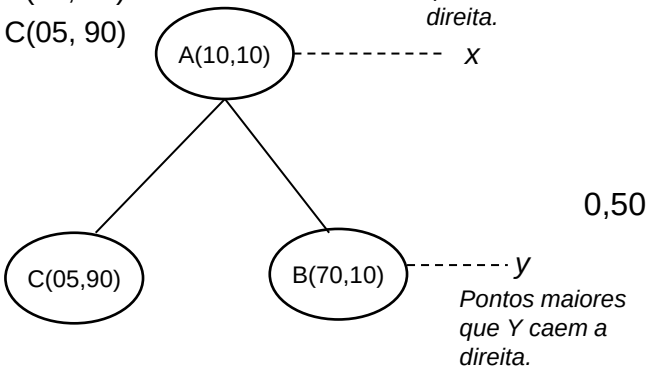


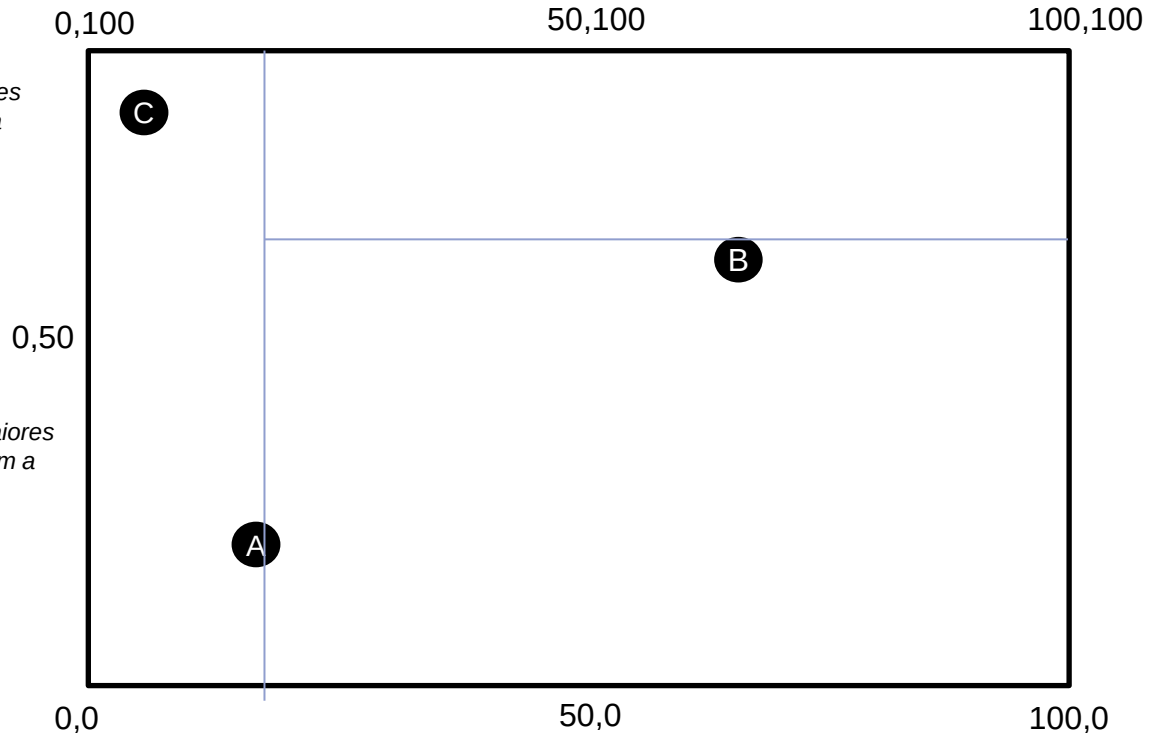


K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)
C(05, 90)

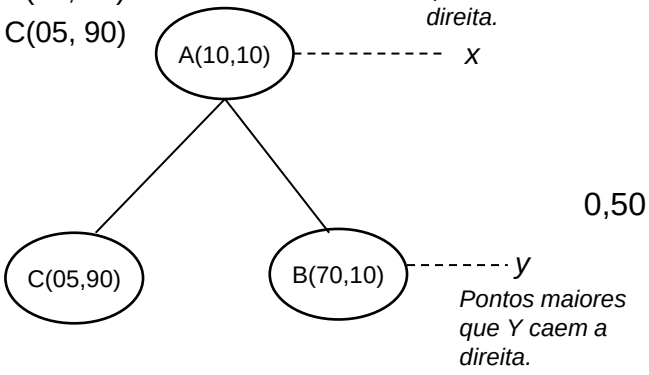


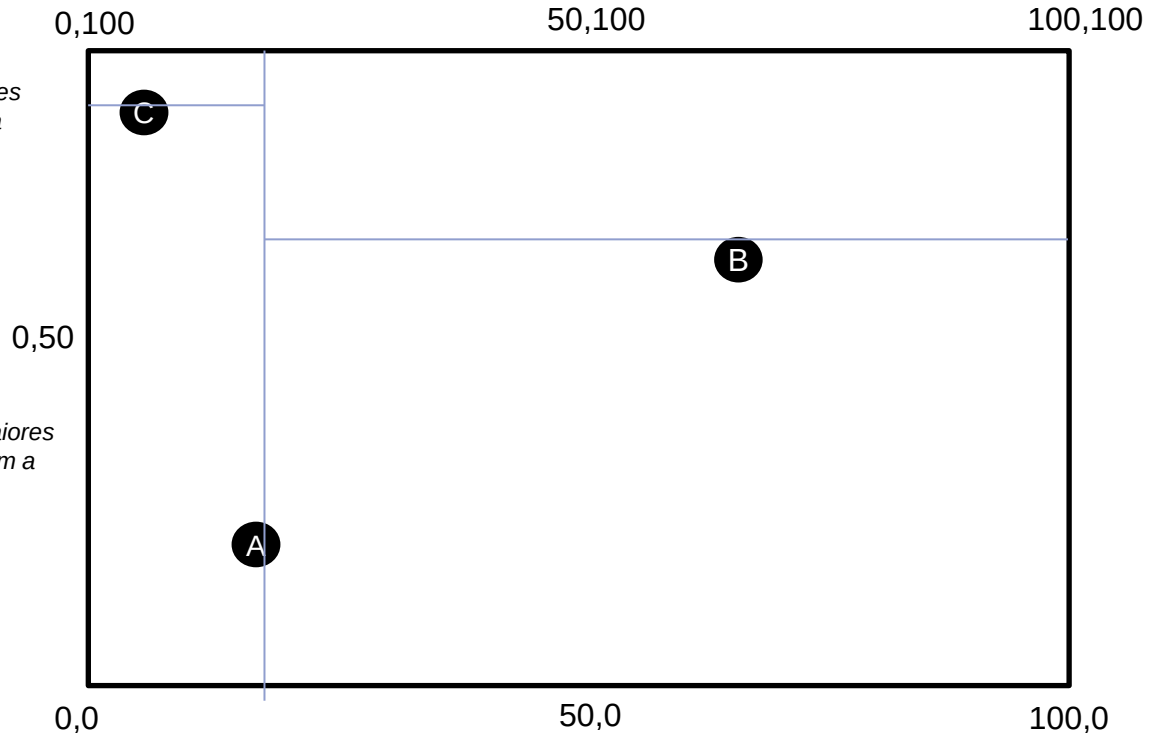


K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)
C(05, 90)

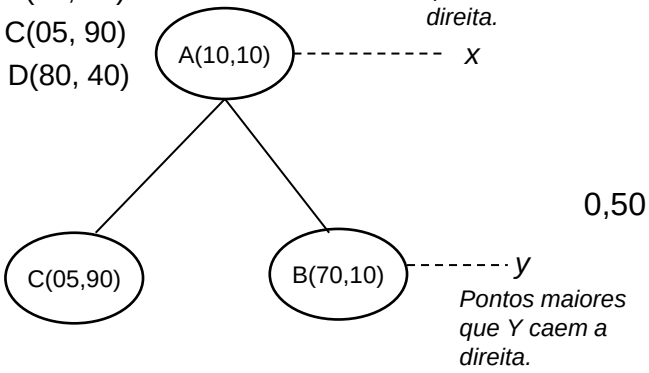


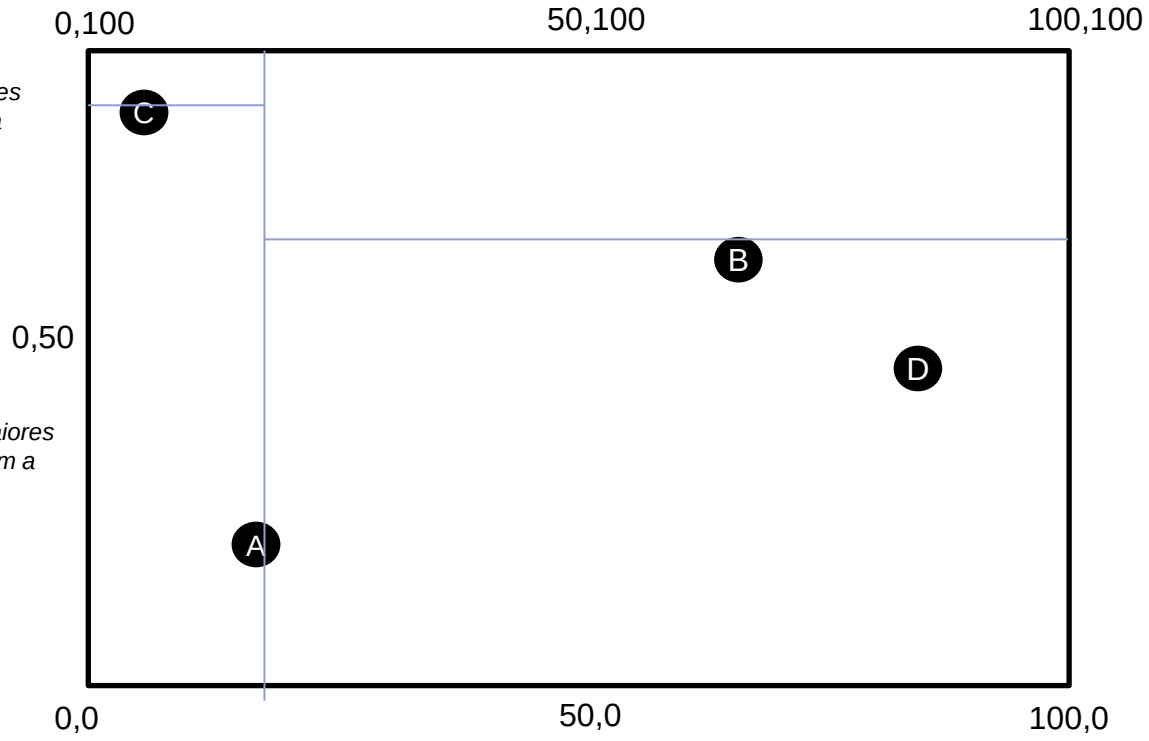


K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)
C(05, 90)
D(80, 40)

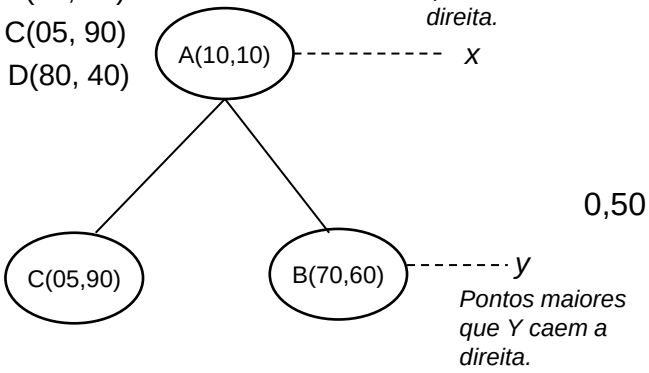


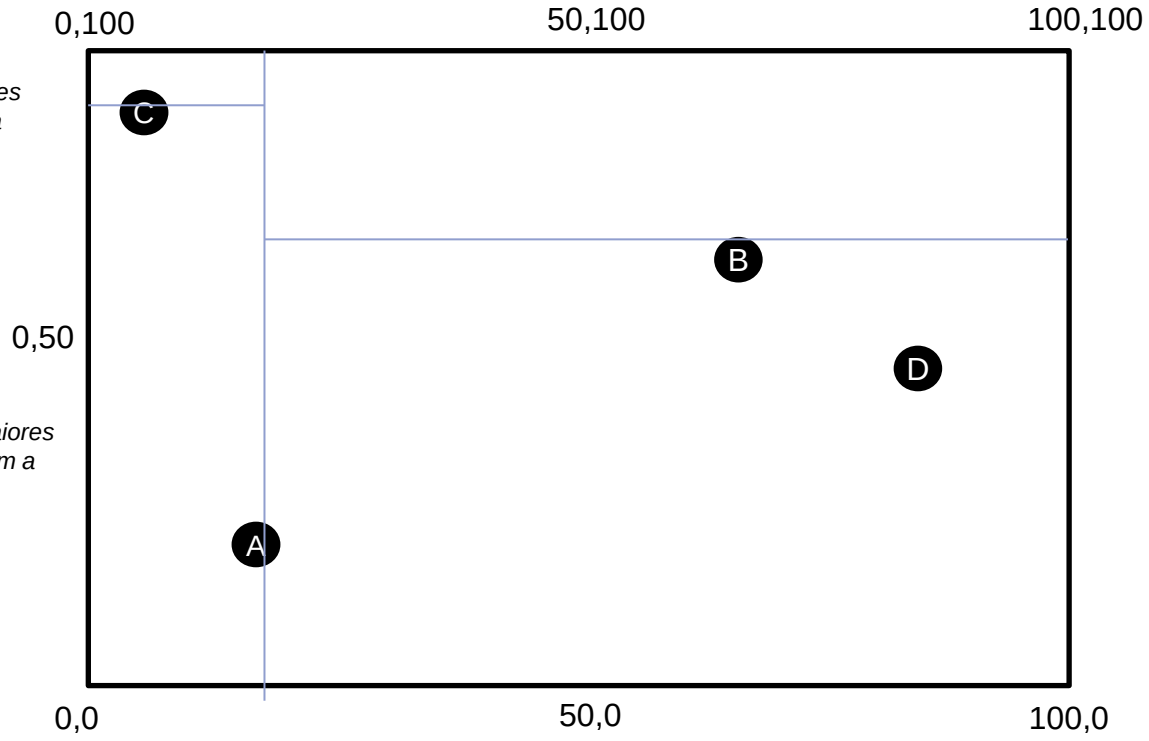


K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)
C(05, 90)
D(80, 40)



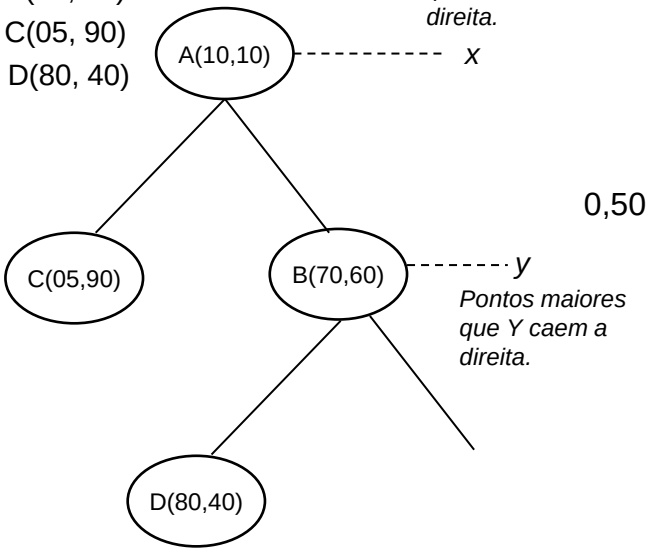


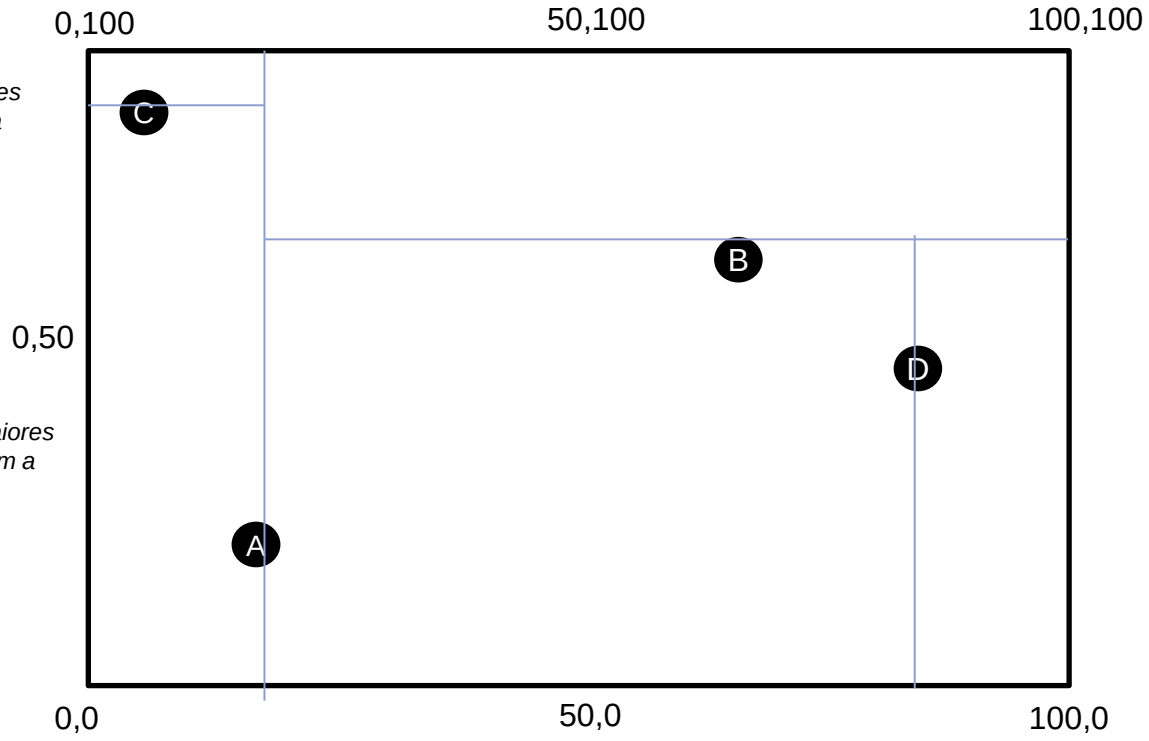
D(80,40) no primeiro nível o X 80 é maior que o X 10, logo cai a direita.
Então muda o discriminador, agora compara Y 40 do D com o Y 60 do B, logo cai a esquerda do B

K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)
C(05, 90)
D(80, 40)

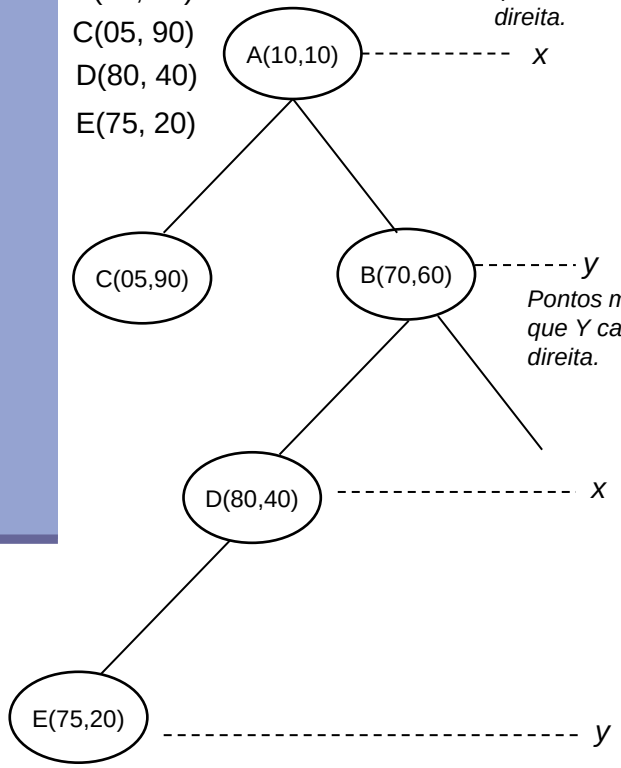


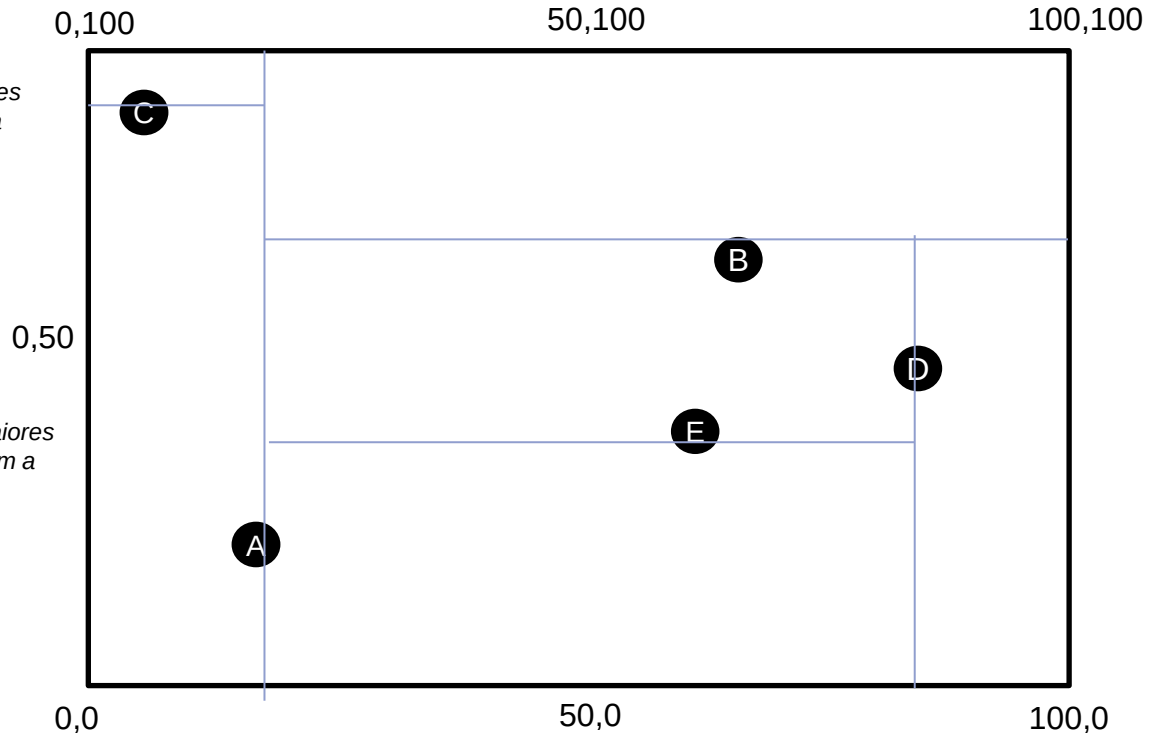


K-d-Tree

Pontos a serem inseridos

A(10,10)
B(70, 60)
C(05, 90)
D(80, 40)
E(75, 20)

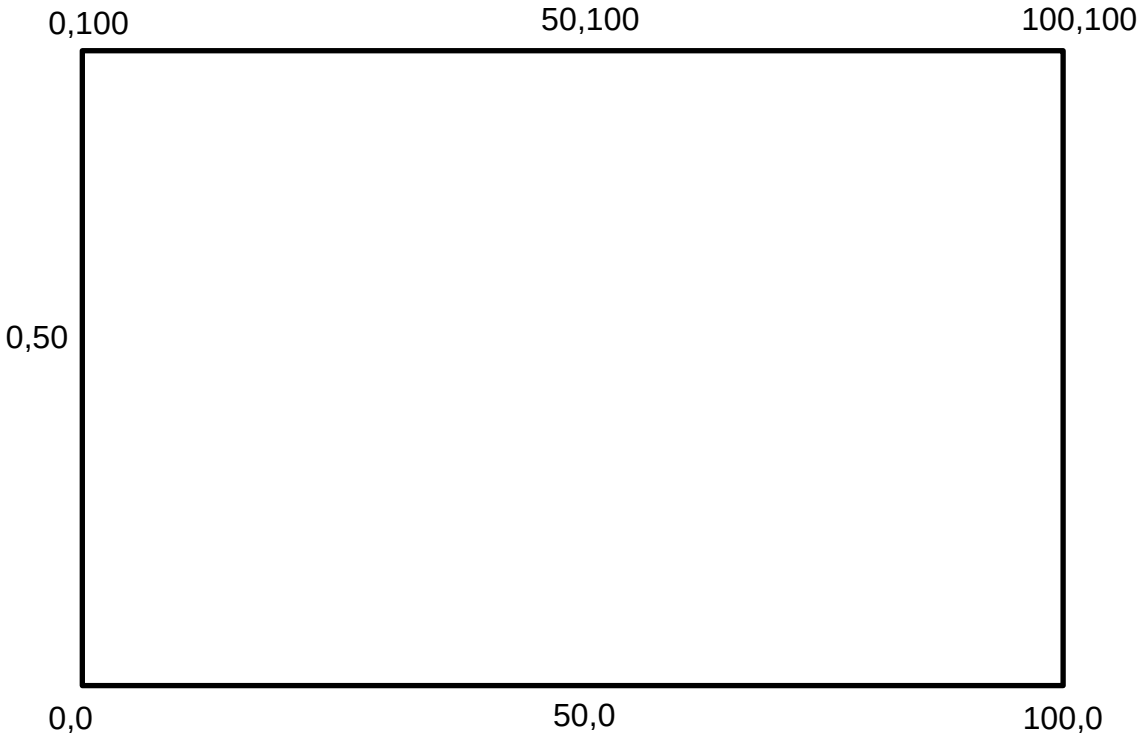




Exercício

Construir a 2dTree para os seguintes pontos:

A(40,45)
B(15,10)
C(70,10)
D(60,50)
E(65,85)
F(85,95)
G(30,10)
H(35,80)



Desenhar a árvore!

Uso em SGBDs

DBMS	R-Tree	Quadtree	Fixed-Grid	k-d-tree	B-tree
Oracle Spatial	Sim	Sim	Não	Não	Sim
PostgreSQL	Sim	Sim	Não	Sim	Sim
MySQL	Sim	Não	Não	Não	Sim
Microsoft SQLServer	Não	Não	Não	Não	Sim
Microsoft Access	Não	Não	Não	Não	Sim
IBM DB2 Spatial Explorer	Não	Não	Sim	Não	Sim
Firebird	Não	Não	Não	Não	Sim

Fonte: internet...a conferir....

```java
public class KdTree {

    private class Nodo {
        Ponto ponto;
        Nodo esquerda;
        Nodo direita;
        Ponto pai;
        public Nodo(String rotulo, int x, int y) {
            this.ponto = new Ponto(rotulo, x, y);
        }
        public Nodo(Ponto p) {
            this.ponto = p;
        }
    }

    private Nodo raiz;

    public void adicionar(String rotulo, int x, int y) {
        Nodo novoNodo = new Nodo(rotulo, x, y);
        if(raiz==null) {
            raiz = novoNodo;
        }
        else {
            Nodo explorador = raiz;
            int nivel = 0;
            while (explorador!=null) {
                if(nivel % 2 == 0) { //discriminador eh o x
                    if(novoNodo.ponto.x < explorador.ponto.x) {
                        if(explorador.esquerda==null) {
                            explorador.esquerda = novoNodo;
                            explorador = null;
                        }
                        else explorador = explorador.esquerda;
                    }
                    else {
                        if(explorador.direita==null) {
                            explorador.direita = novoNodo;
                            explorador = null;
                        }
                        else explorador = explorador.direita;
                    }
                }
                else {
                    if(novoNodo.ponto.y < explorador.ponto.y) {
                        if(explorador.esquerda==null) {
                            explorador.esquerda = novoNodo;
                            explorador = null;
                        }
                        else explorador = explorador.esquerda;
                    }
                    else {
                        if(explorador.direita==null) {
                            explorador.direita = novoNodo;
                            explorador = null;
                        }
                        else explorador = explorador.direita;
                    }
                }
                nivel++;
            }
        }
    }
```