Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Departamento de Computação
BCC203 – Estrutura de dados II

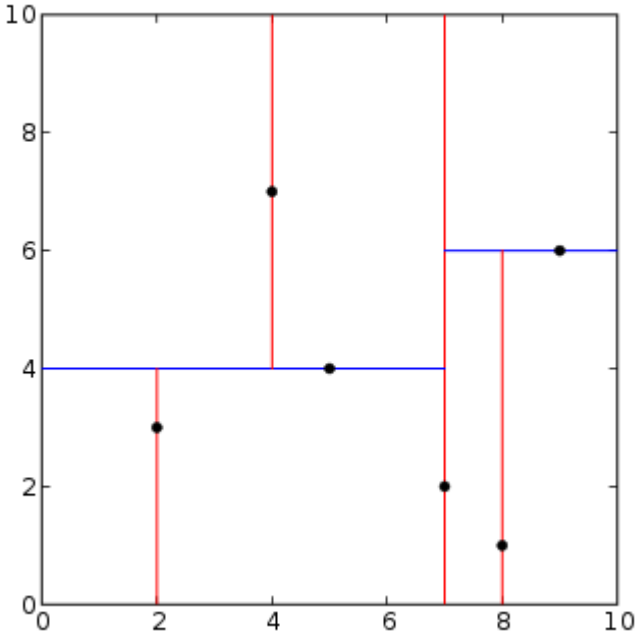
Arvore kd

Componentes:
André Riberio de Brito
Daniel Augusto da Costa Patrono
Guilherme Augusto Rodrigues Melo
Henrique Prates Caldeira
Leonardo Isaac Silva Flores
Nando Oliveira Coelho

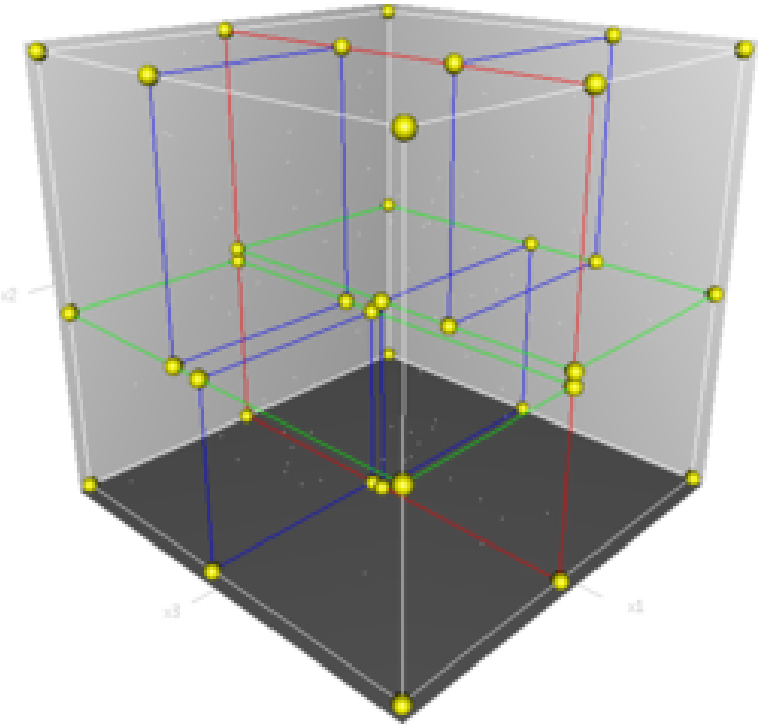
Professor: Guilherme Tavares de Assis

Introdução

- Criada por Jon Bentley, em 1975, enquanto cursava Ciência da Computação na Universidade de Carolina do Norte;
- A Árvore *k*-D é uma Árvore Binária que permite um eficiente processamento e armazenamento de um número finito de pontos, em um espaço multidimensional.
- árvore k-d (abreviação de k-dimensional), onde k é o numero de registros de um nó.
- A estrutura do nó da árvore kD armazena as coordenadas de um ponto e dois ponteiros para o mesmo tipo do nó, um para o filho a direita e outro para o filho à esquerda.



Árvore K-d bidimensional
(2,3), (5,4), (9,6), (4,7), (8,1), (7,2).



Árvore k-d 3-dimensional.

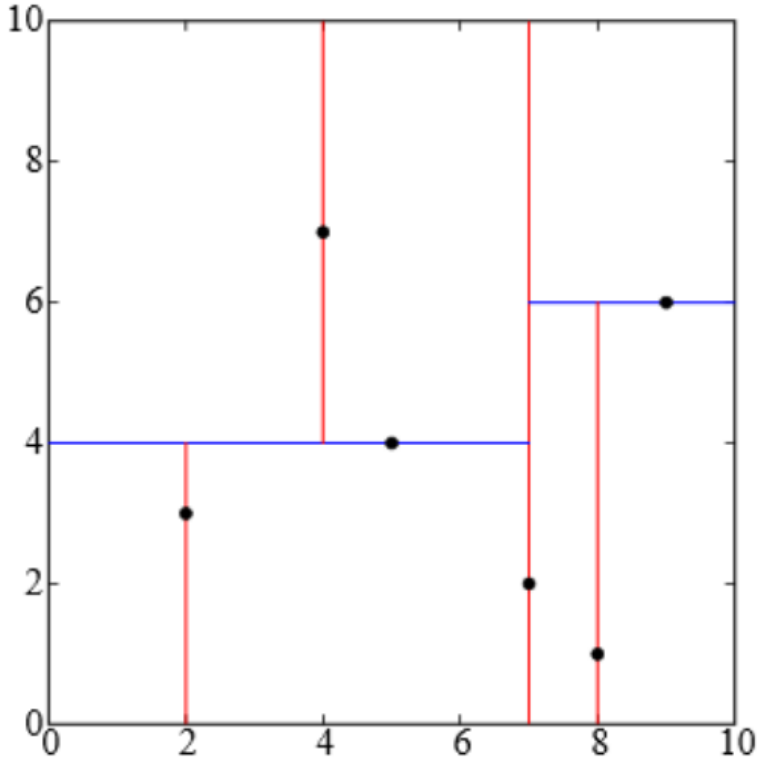
Introdução

- O árvore k-D é basicamente um tipo de árvore binária que divide os pontos de dados em parcelas menores mais viáveis (buckets). Minimizando, assim, o número dos pontos de dados que necessitam ser procurados. Esta estrutura de dados é muito usada durante a pesquisa em extensão ortogonal.

- árvore k-D é um dos métodos mais rápidos para resolver problemas de pesquisa de pontos vizinhos. Necessita somente uma média de O (Log n), assim como na inserção e remoção.

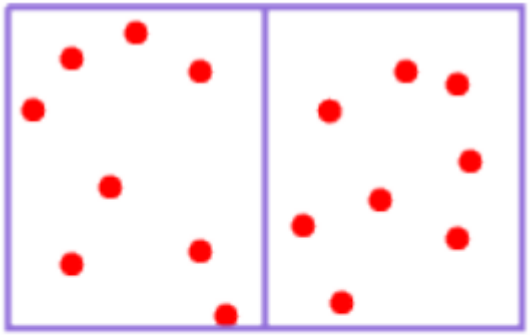
- Existem dois tipos principais da árvore kd :

  1)Baseada em regiões.

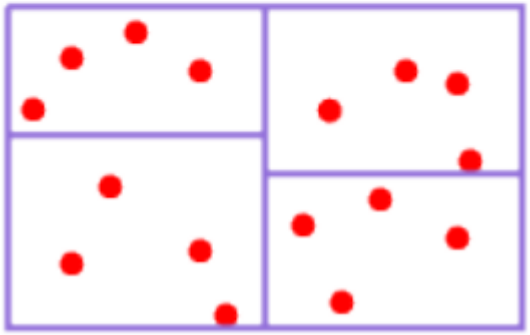


2) baseada em pontos

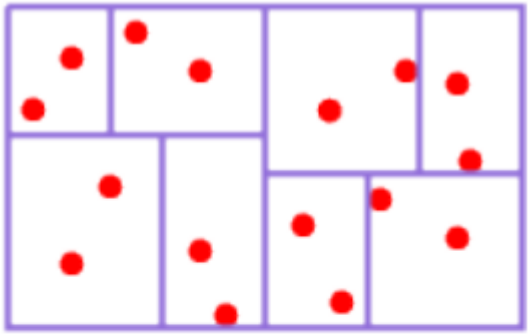
1.



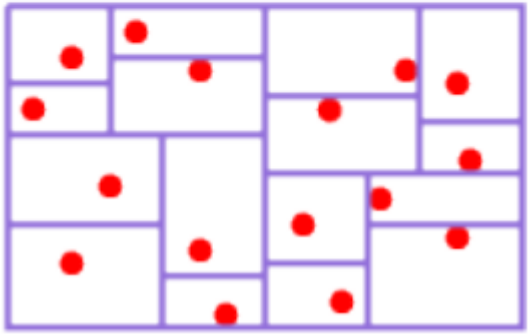
2.



3.



4.



Construção

- A arvore kd composta basicamente de:

1) Um nó raiz;
2) Ponteiros para nós filhos (sempre dois);
3) Extensão da célula (coordenadas máximas e mínimas em cada eixo);
4) Os dados (pontos) contidos na célula;
5) Apontador para vizinhos;

- Os nodos-folha são essencialmente caixas dos dados que estão em um determinado nível de busca.
- Os nodos internos têm dois filhos, esquerda e direita, que podem ser folhas ou internos.

Construção

TAD :

```
typedef struct kd_tree
{
    double *coordenadas;
    struct kd_tree *filho_esq;
    struct kd_tree *filho_dir;
} kd_tree;
```

Algoritmo ConstroiArvKD(P,prof)

1. SE P contém apenas um ponto **then**

2. **return** uma folha contendo este ponto

3. SENÃO SE prof é plano **ENTÃO**

4. divida P em dois subconjuntos com uma linha vertical pela coordenada x mediana dos pontos em P.

P1 é o conjunto de pontos da esquerda e P2 o conjunto de pontos da direita.

Pontos exatamente na linha pertencem a P1

5. **SENÃO**

6. divida P em dois subconjuntos com uma linha horizontal pela coordenada y mediana dos pontos em P.

P1 é o conjunto de pontos acima da linha e P2 o conjunto de pontos abaixo da linha.

Pontos exatamente na linha pertencem a P1.

7. Vright <- **ConstroiArvKD**(P1,prof+1).

8. Vleft <- **ConstroiArvKD**(P2,prof+1).

9. Crie um nodo V com Vright e Vleft juntamente com seus filhos direito e esquerdo, respectivamente.

10. retorne V.

Inserção

- A inserção em um novo nó na árvore kd é similar à inserção da Árvore Binária de Busca. O procedimento de pesquisa na árvore kd é realizado até que um ponteiro nulo seja encontrado, indicando um local propício para inserir um novo nodo.

Pontos para ser inseridos:

A(50,30)

B(40,30)

C(20,20)

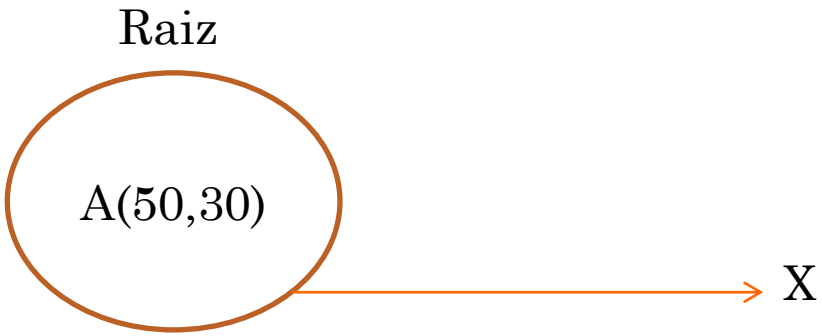
D(70,40)

E(45, 60)

F(80,70)

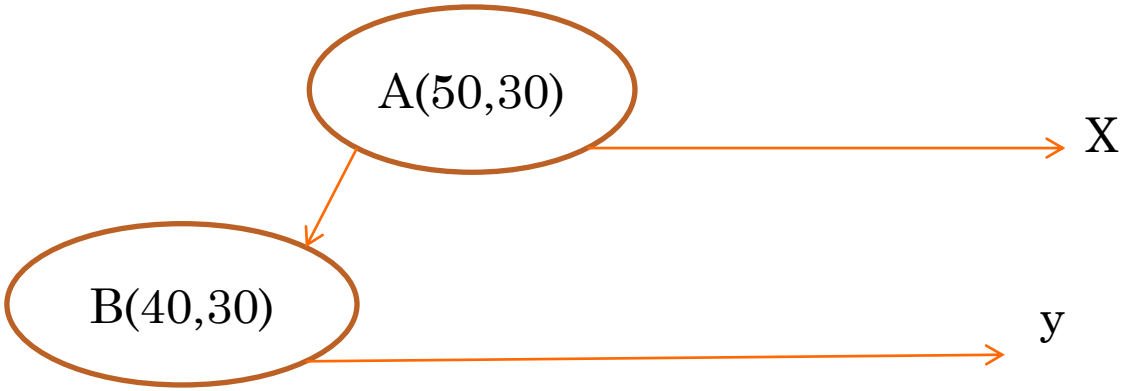
Inserção Arvore Kd

- A(50,30)



Inserção Arvore Kd

- B(40,30)

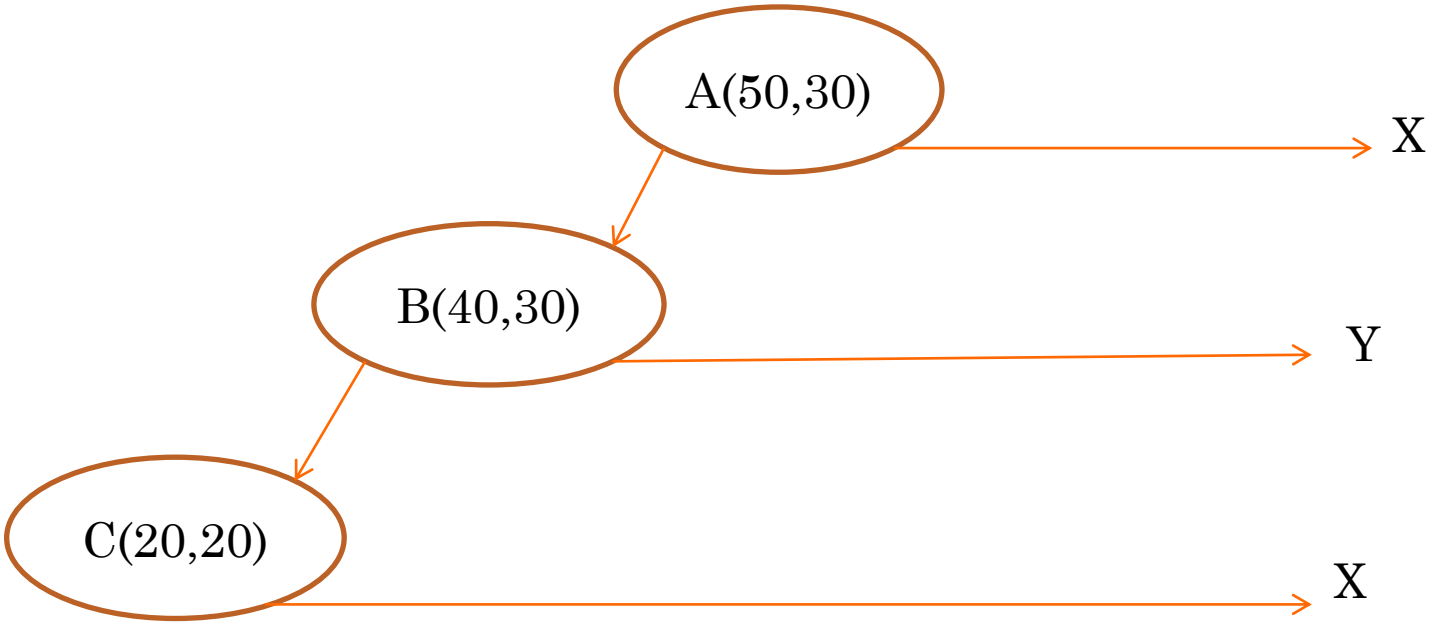


- Algoritmo:

Se Bx < Ax, B é inserido no nó da esquerda

Inserção Arvore Kd

- C(20,20)



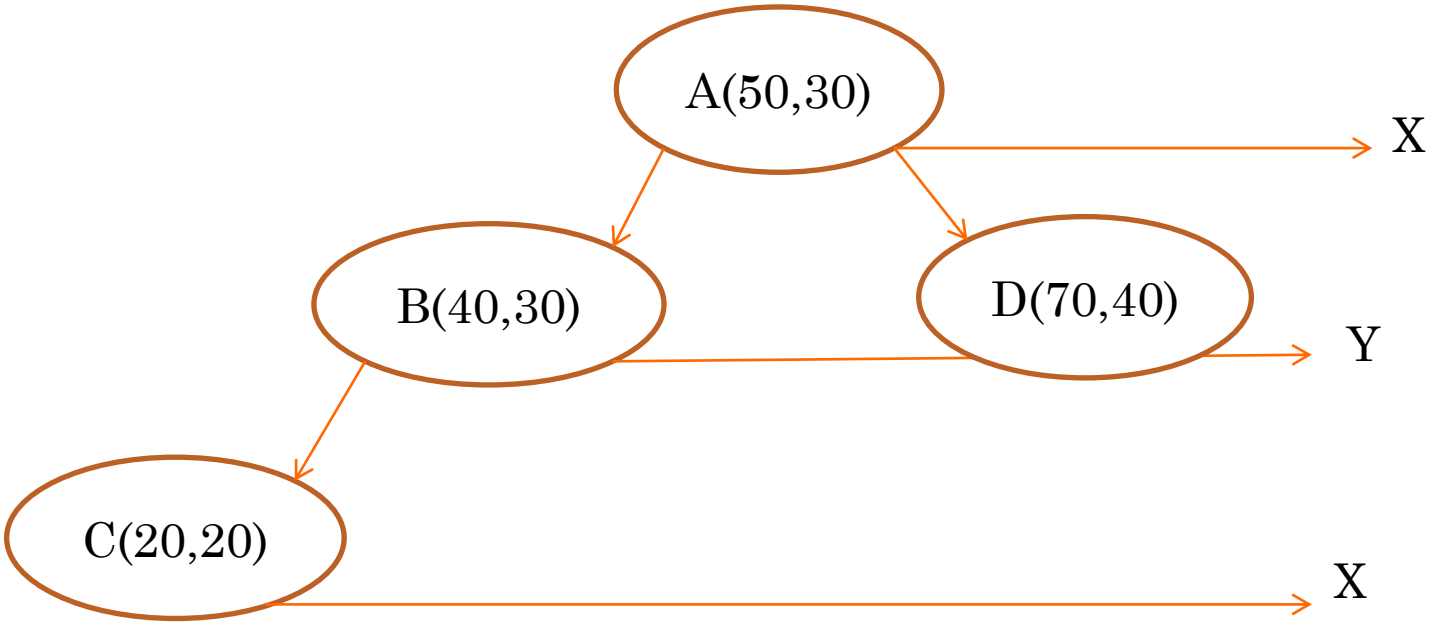
- Algoritmo:

Cx < Ax , C caminha pelo nó esquerdo de A.

Cy < By, C é inserido a esquerda de B.

Inserção Arvore Kd

- D(70,40)

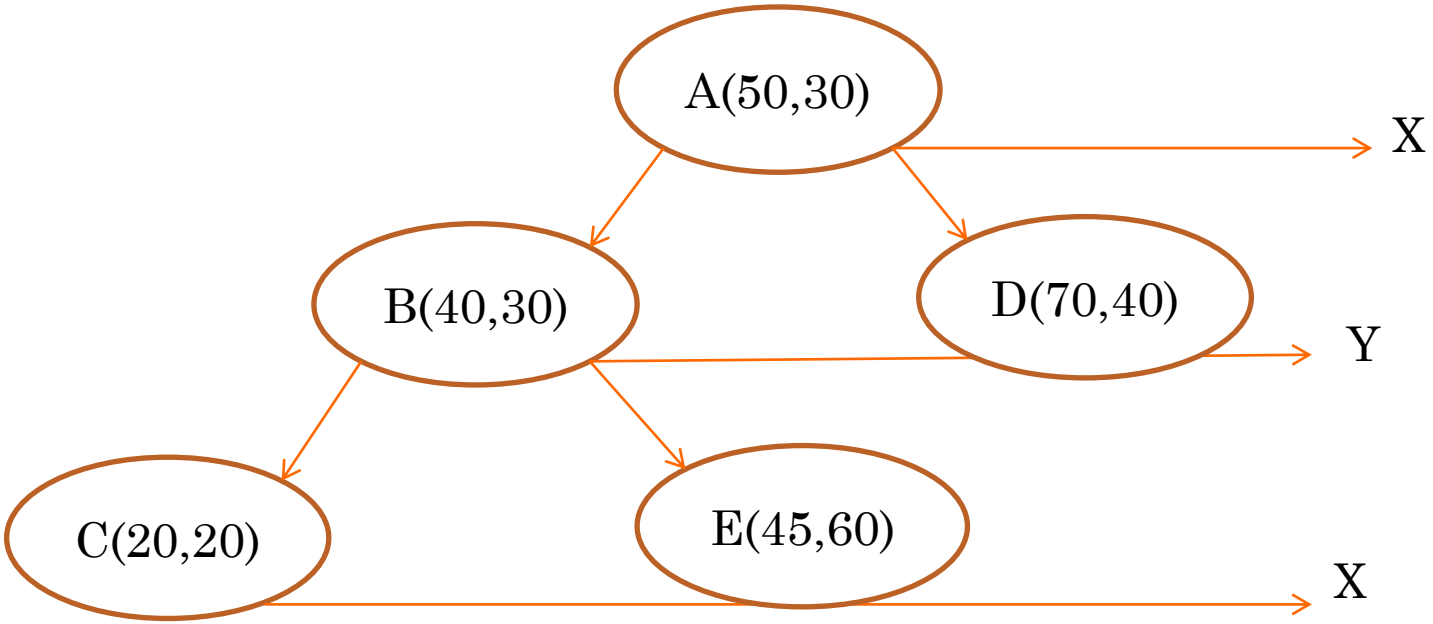


- Algoritmo:

Dx > Ax , D é inserido a direita de A.

Inserção Arvore Kd

- E(45,60)



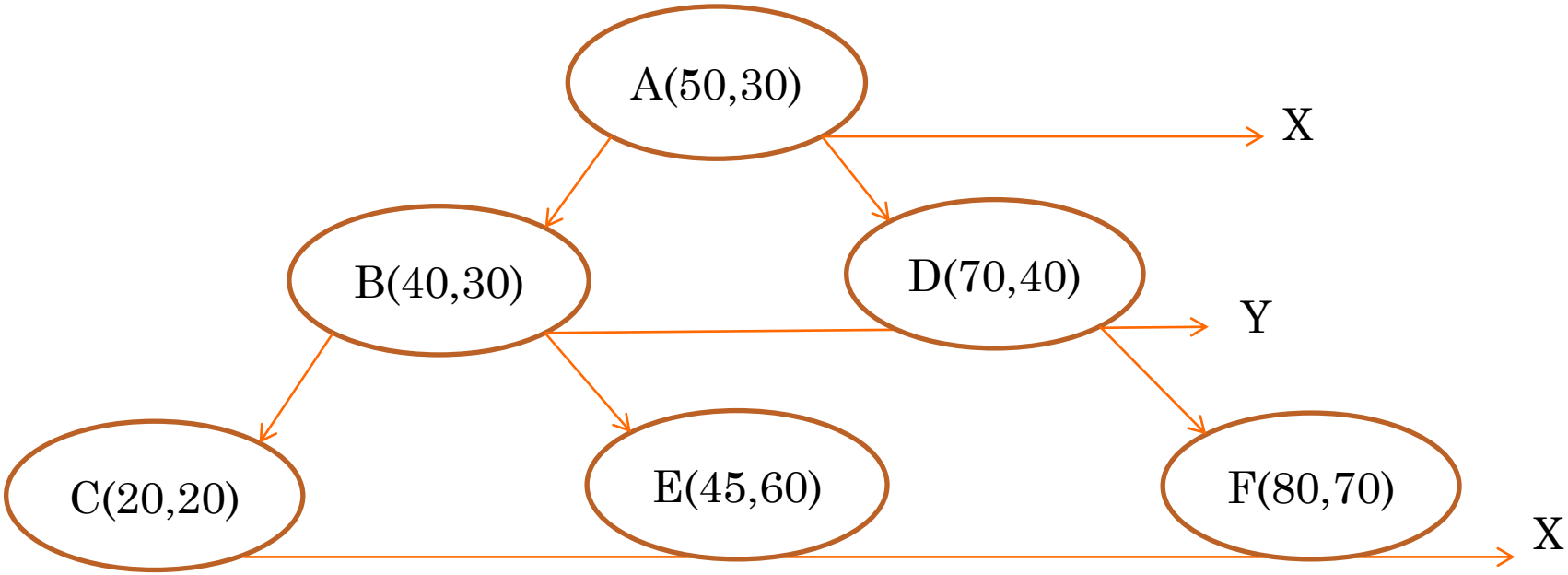
- Algoritmo:

Ex < Ax , E caminha pelo nó esquerdo de A.

Ey > By, E é inserido a direita de B.

Inserção Arvore kd

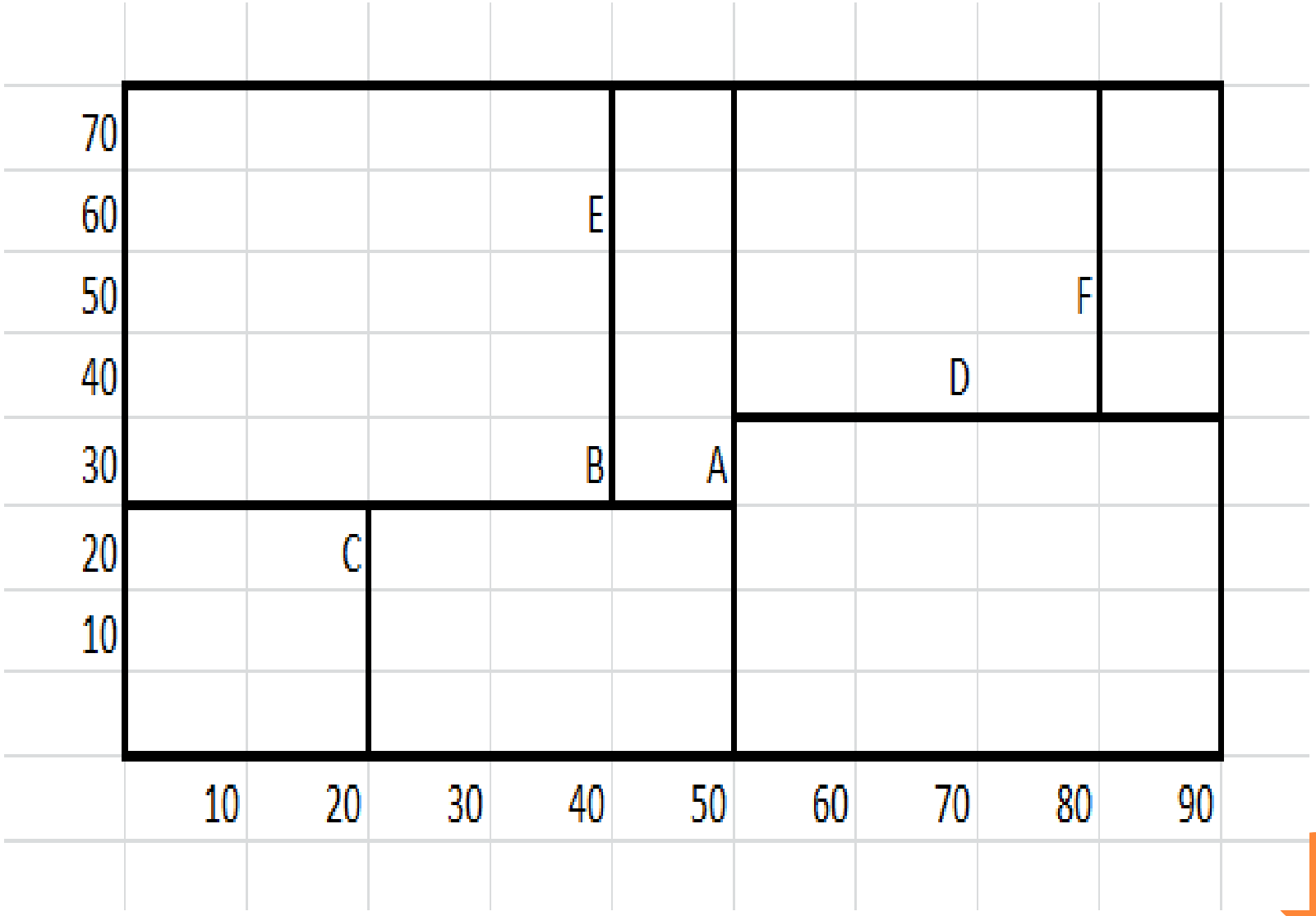
- F(80,70)

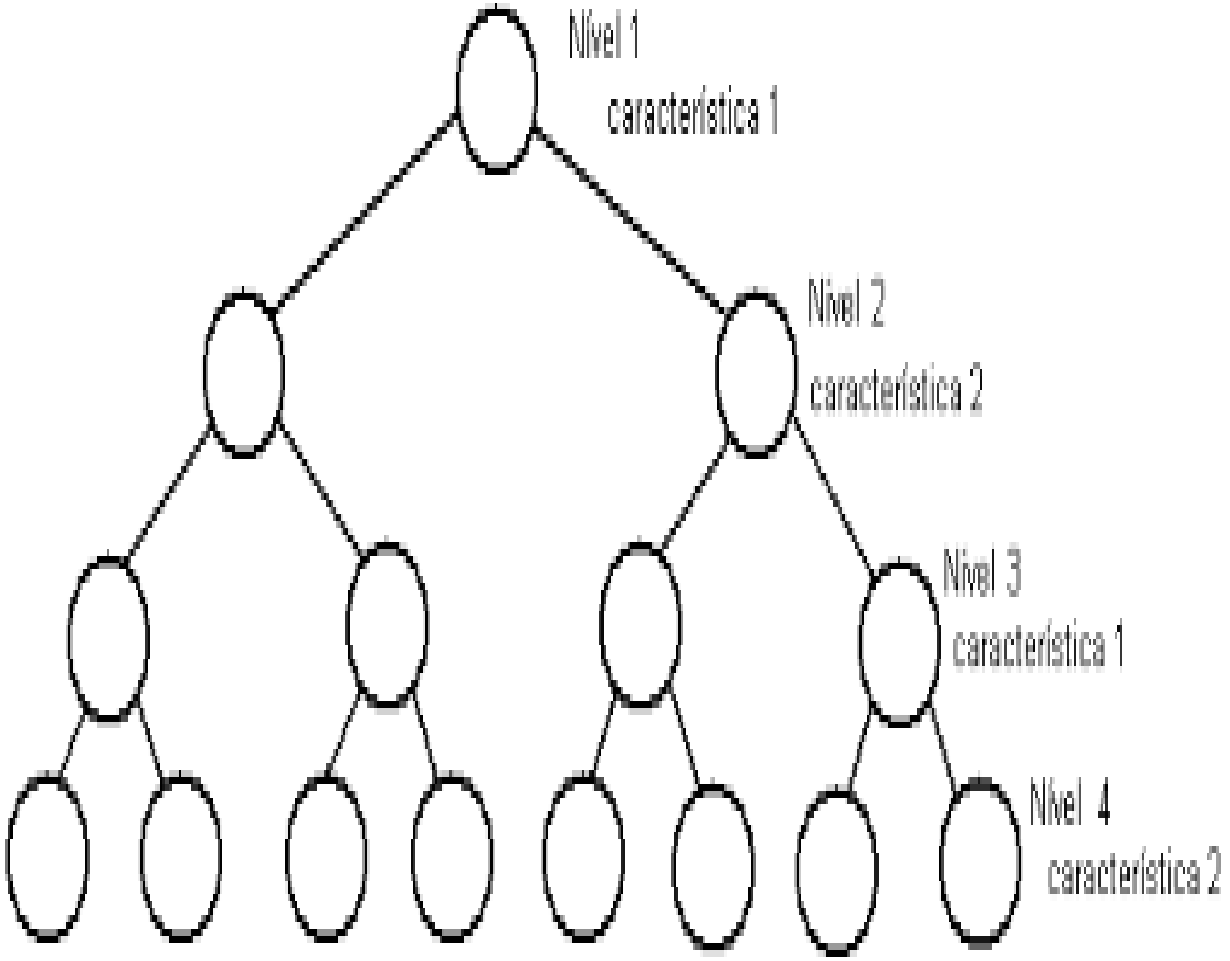


- Algoritmo:

Fx>Ax, F caminhada pelo nó a direita de A;

Fy>Dy, F é inserido a direita de D.


70
60
50
40
30
20
10
E
F
D
B
A
C
10
20
30
40
50
60
70
80
90


Nivel 1
característica 1
Nivel 2
característica 2
Nivel 3
característica 1
Nivel 4
característica 2

Remoção

- A remoção de um item da arvore KD é similar a uma remoção em uma arvore binaria, mas ligeiramente mais difícil. Primeiramente deve se percorrer a arvore ate que este item seja encontrado.
- Caso seja um no folha, basta remover.
- Caso contrario, verifica-se o nível do nó P;
- Se o nível for impar, o Y e escolhido como discriminador, caso seja par o X que e utilizado.
- Assim deve-se escolher o Ymim ou Xmax dependendo do discriminador escolhido.
- Ymim: registro da subarvore a direita com o menor valor de y;
- Xmax: registro da subarvore a esquerda com o maior valor de X;

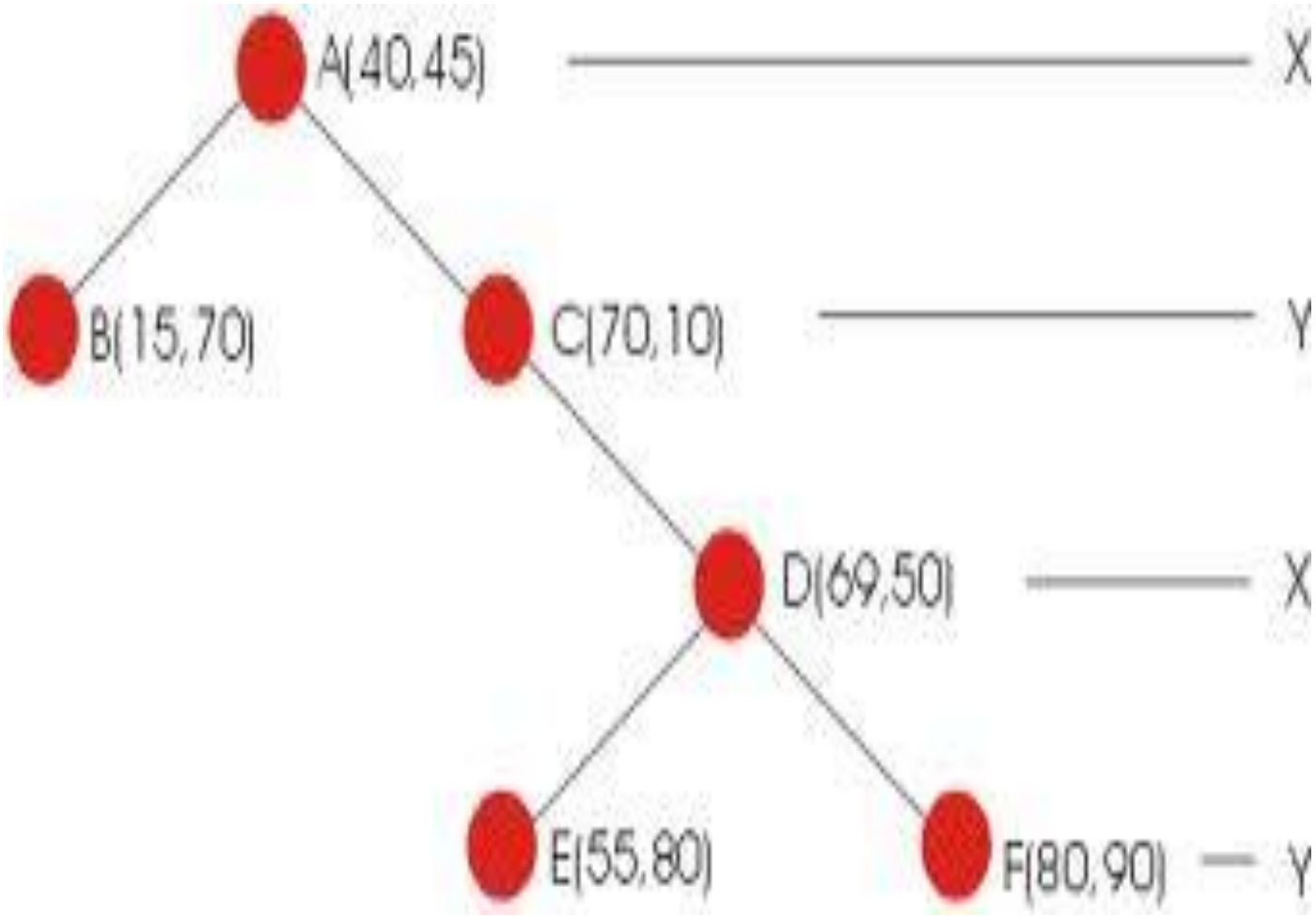
Pesquisa

- Para pesquisar na árvore k-D e semelhante a pesquisa em uma arvore binaria, exceto que cada nível da árvore é associado com um eixo do plano cartesiano.
- No registro que contem uma coordenada x,y de referente ao planto cartesiano.
- Sendo o registro o P(x, y) a ser localizado, iniciamos a comparação de P com o endereço do ponto A armazenado na raiz, se p registro P coincidir com o endereço da raiz, então a pesquisa retornará com sucesso, caso contrario a pesquisa continua a descer de nível.

Pesquisa

- O elemento Ax da raiz é comparado com o elemento Px para determinar em qual ramo seguir.
- Se o valor Ax for menor que o valor pesquisado, nos direcionaremos para a subárvore a esquerda, caso contrario subárvore a direita.
- Todos nodos com valor de x maior ou igual a Ax estão na subárvore da direita.
- A pesquisa sempre se inicia comparando o elemento Px com o elemento x da raiz, assim passando para o próximo nível a comparação será feita com o elemento y, ou seja, em cada nível usa uma das coordenadas para fazer a comparação, iniciando com x, y, x, ...

- Pesquisar o registro P(80,90) ?



Analise de Complexidade

- Construção:

Caso médio: O(nlogn)

Pior caso: O(n)

- Acesso Balanceada:

Caso médio: O(logn).

- Acesso Desbalanceada:

Caso médio: O(n).

- Inserção em arvore balanceada:

Caso médio O(log n).

- Inserção em arvore não balanceada:

Caso médio $O(\log^2 n)$.

- Remoção em arvore balanceada:

Caso médio O(log n).

- Eliminação da raiz:

O(n(k-1)/k)

Aplicação

- Jogos Digitais

- Design interiores

- Astronomia

Vantagem

- Árvores k-d podem ser utilizadas diretamente para todos os 3 tipos de pesquisa: simples, com limites e booleana.
- O tempo médio de acesso a um registro não é pior do que o da árvore binária. Todas as características de tempo de pesquisa, complexidade, etc da árvore binária valem, no que diz respeito à pesquisa, também para a árvore k-d: O(1,4 log2 n).
- Inserção (não balanceada) de um nodo requer também tempo O(log2 n).
- Flexibilidade: Aplicável a qualquer tipo de aplicação onde se queira fazer recuperação de chaves secundárias ou recuperação multichaves.

Desvantagem

- Gera árvores de profundidade extremamente grande. *Solução*: Pode ser resolvido através da paginação.
- Inserção balanceada é extremamente cara.
- Rebalanceamento (apos várias inserções ou exclusões), também extremamente caro.

Referencias

- http://en.wikipedia.org/wiki/K-d_tree
- http://donar.umiacs.umd.edu/quadtree/points/kdtree.html
- http://www.inf.ufsc.br/~ine5384-hp/k-d/arvore-kd.html