UNISINOS

Estruturas Avançadas de Dados I
(QuadTree)

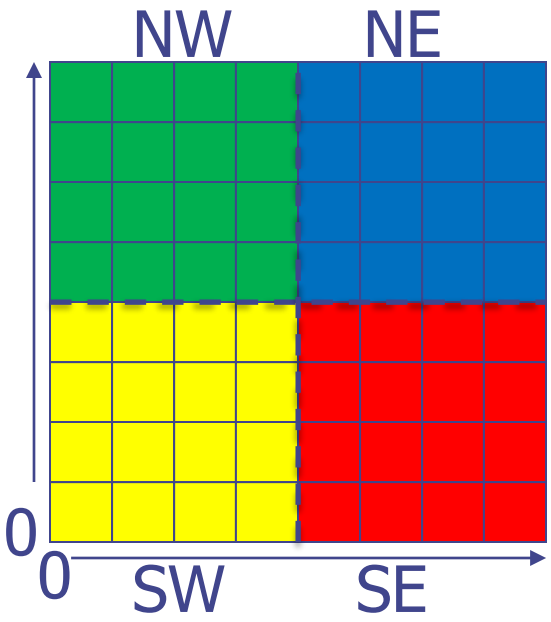
Prof. Gilberto Irajá Müller

Última atualização 18/09/2017

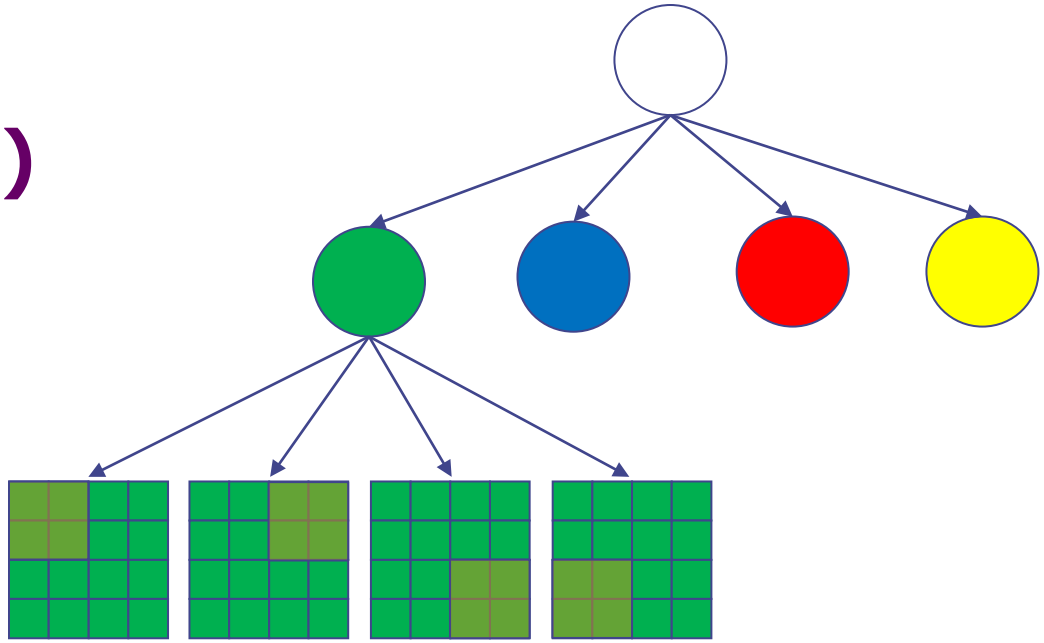
Introdução

- Desenvolvida em 1974 por Raphael Finkel e Jon Louis Bentley com artigo de nome "*Quad Trees: A Data Structure for Retrieval on Composite Keys*";
- É uma estrutura de dados em árvore onde o nó tem exatamente 4 filhos;
- Cada nó representa uma caixa que delimita um espaço indexado, sendo que o nó raiz cobre toda essa área;
- Útil para busca espacial (indexação), compressão de imagens, colisão em jogos (duas dimensões), entre outros.

Introdução (cont.)



NW (North-West (Noroeste))
NE (North-East (Nordeste))
SW (South-West (Sudoeste))
SE (South-East (Sudeste))



Para um conjunto de pontos P em uma área Q = $[x_q, x'_q] \times [y_q, y'_q]$ define uma QuadTree T(P):

- Se $P \leq 1$, então T(P) é uma folha;

- Caso contrário:
- $x_{med} = (x_q + x'_q) / 2$ e $y_{med} = (y_q + y'_q) / 2$
- $P_{NW} = \{p \in P \mid p_x < x_{med} \text{ e } p_y \geq y_{med}\}$
- $P_{SW} = \{p \in P \mid p_x < x_{med} \text{ e } p_y < y_{med}\}$
- $P_{NE} = \{p \in P \mid p_x \geq x_{med} \text{ e } p_y \geq y_{med}\}$
- $P_{SE} = \{p \in P \mid p_x \geq x_{med} \text{ e } p_y < y_{med}\}$

T(P) consiste de um root r que armazena Q com filhos P_i e Q_i ($i \in \{NE, NW, SW, SE\}$).

Estrutura da QuadTree

```
public class QuadTree<K extends Comparable<K>, V> implements QuadTreeADT<K, V> {
   private Node root;

   // Classe Node (slide seguinte)

   @Override
   public void clear() {
      root = null;
   }

   @Override
   public boolean isEmpty() {
      return root == null;
   }
// Continua...
```

```
public interface QuadTreeADT<K extends Comparable<K>, V> {
   public void clear();
   public boolean isEmpty();
   public void insert(K x, K y, V value);
   public void query2D(Interval2D<K> rect);
}
```

Estrutura do Nó

```
private class Node {
    private K x, y;
    private Node NW, NE, SE, SW;
    private V value;

    public Node(K x, K y, V value) {
        this.x = x;
        this.y = y;
        this.value = value;
    }
}
```

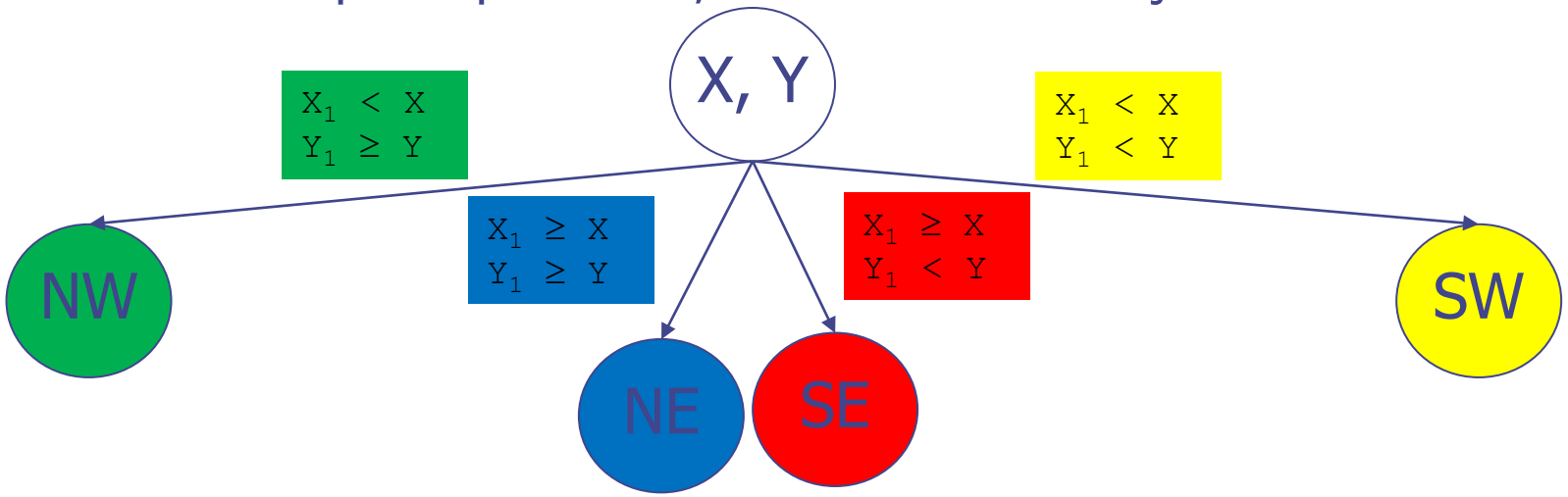
Valores das coordenadas

Quadrantes.

Valor associado à coordenada

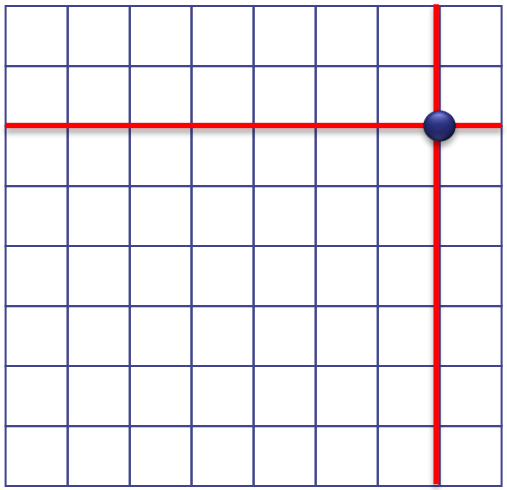
Inserção

- Um novo nó é inserido conforme:
  - Se a QuadTree não tem root definido, então, cria-se um novo root armazenando as coordenadas iniciais, finalizando a inserção;
  - Caso exista um root, então, encontra-se o quadrante com base no nó corrente até encontrar um quadrante que não exista um nó;
  - Se o quadrante não possuir um nó, então, um novo filho é criado naquele quadrante, finalizando a inserção.



Inserção (cont.)

- Inserindo (7,6)



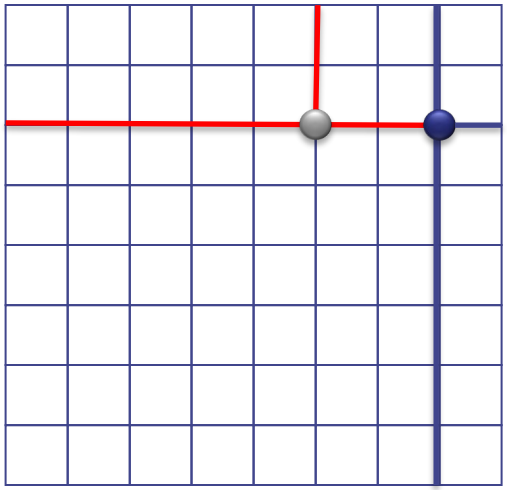
root

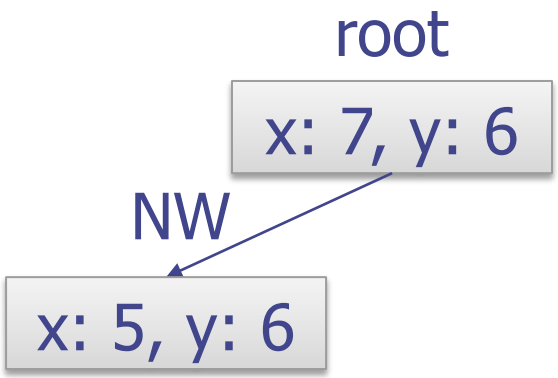
x: 7, y: 6

- Observa-se que estamos delimitando a região (matriz 8 x 8), mas na prática não há esse limite;
- Já o quadrante, estamos limitando para dar uma ideia da **Quad**Tree.

Inserção (cont.)

- Inserindo (5,6)

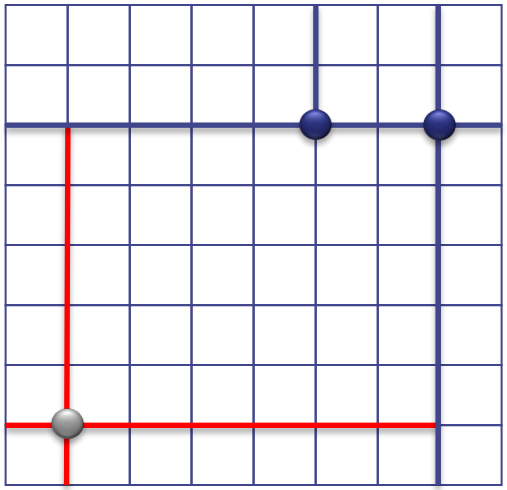


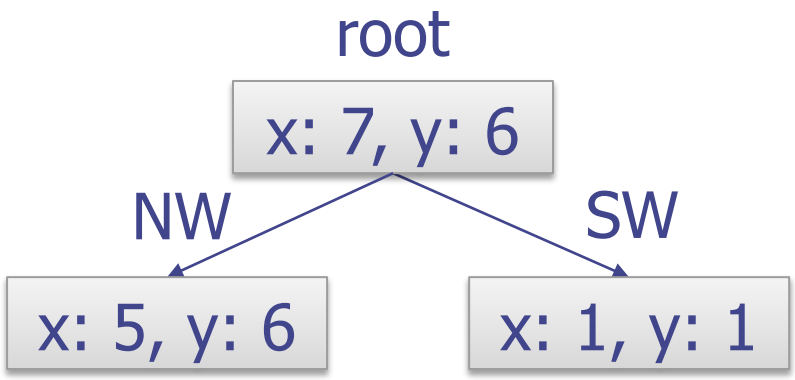


- Observa-se que o ponto (5, 6) pertence ao quadrante NW do root.

Inserção (cont.)

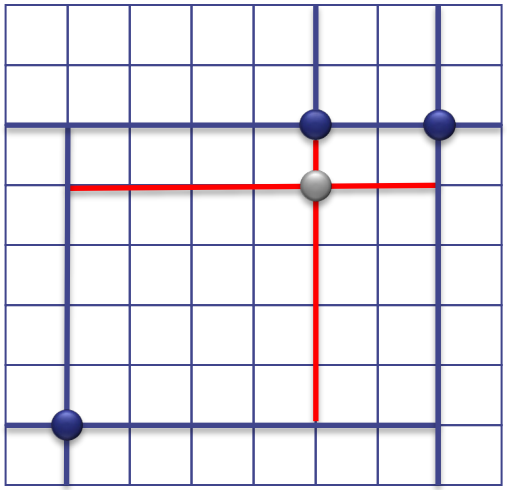
- Inserindo (1,1)

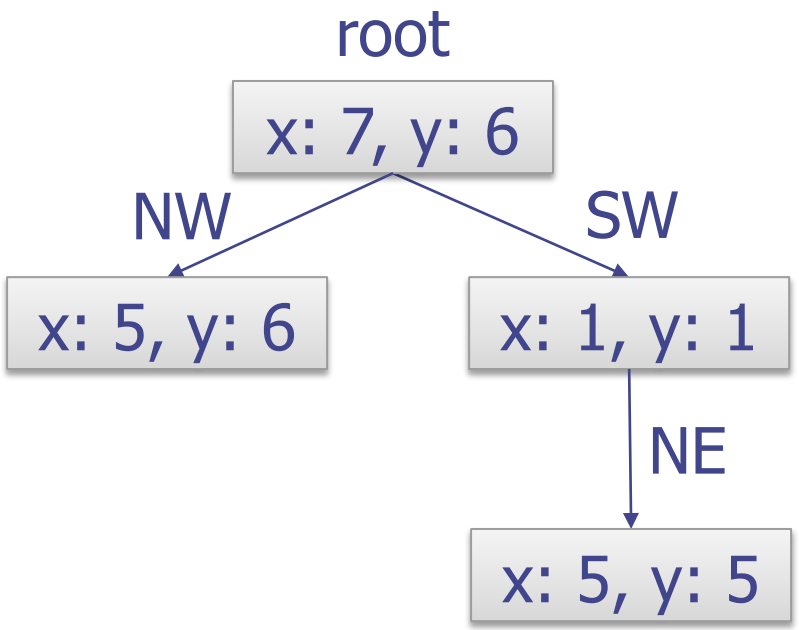




Inserção (cont.)

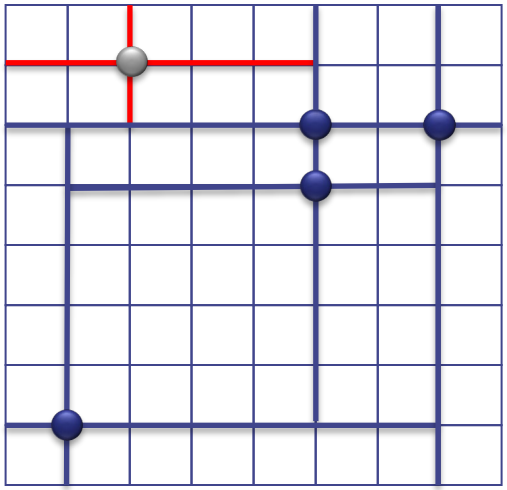
- Inserindo (5,5)

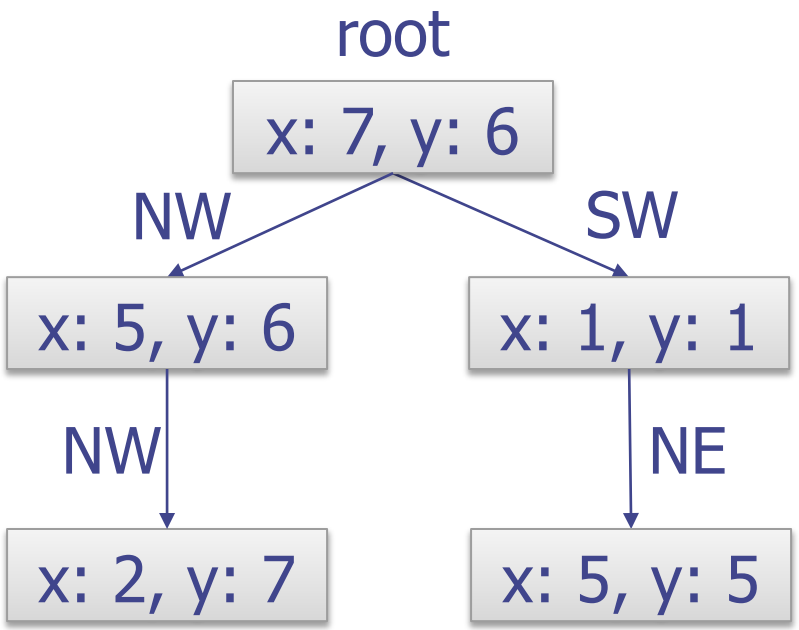




Inserção (cont.)

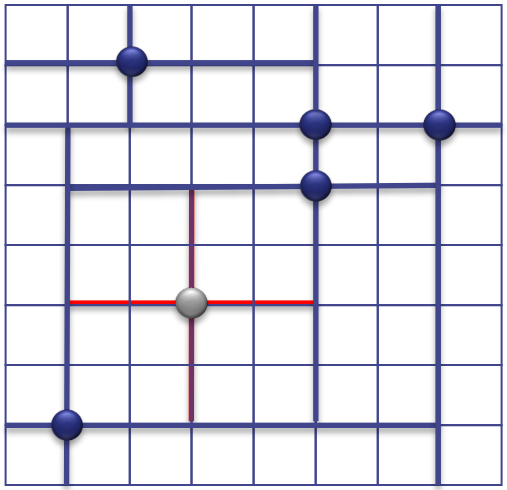
- Inserindo (2,7)

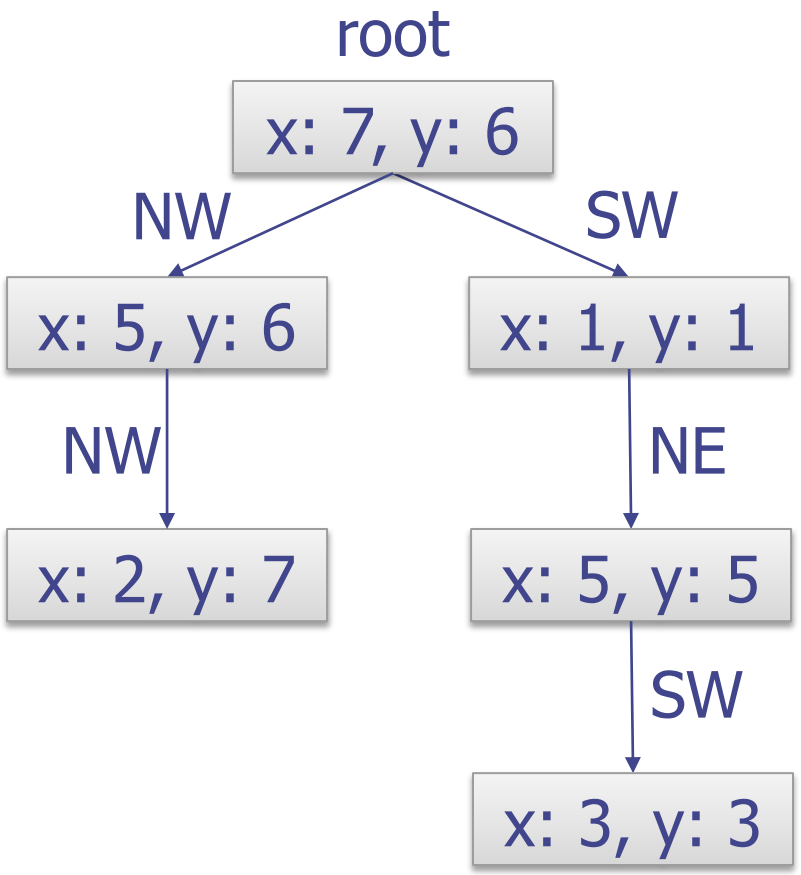




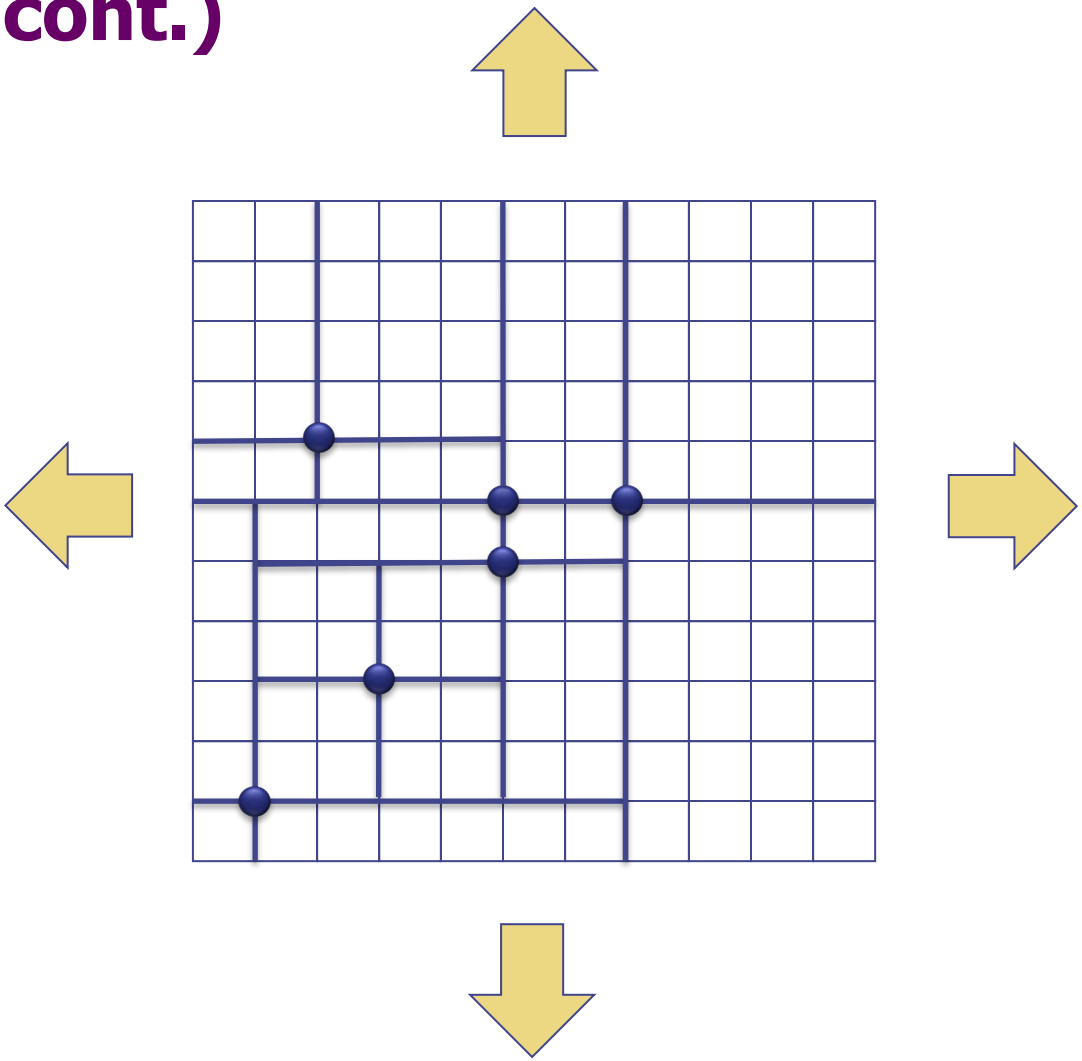
Inserção (cont.)

- Inserindo (3,3)





Inserção (cont.)



- Quando não há limite, as bordas são infinitas...

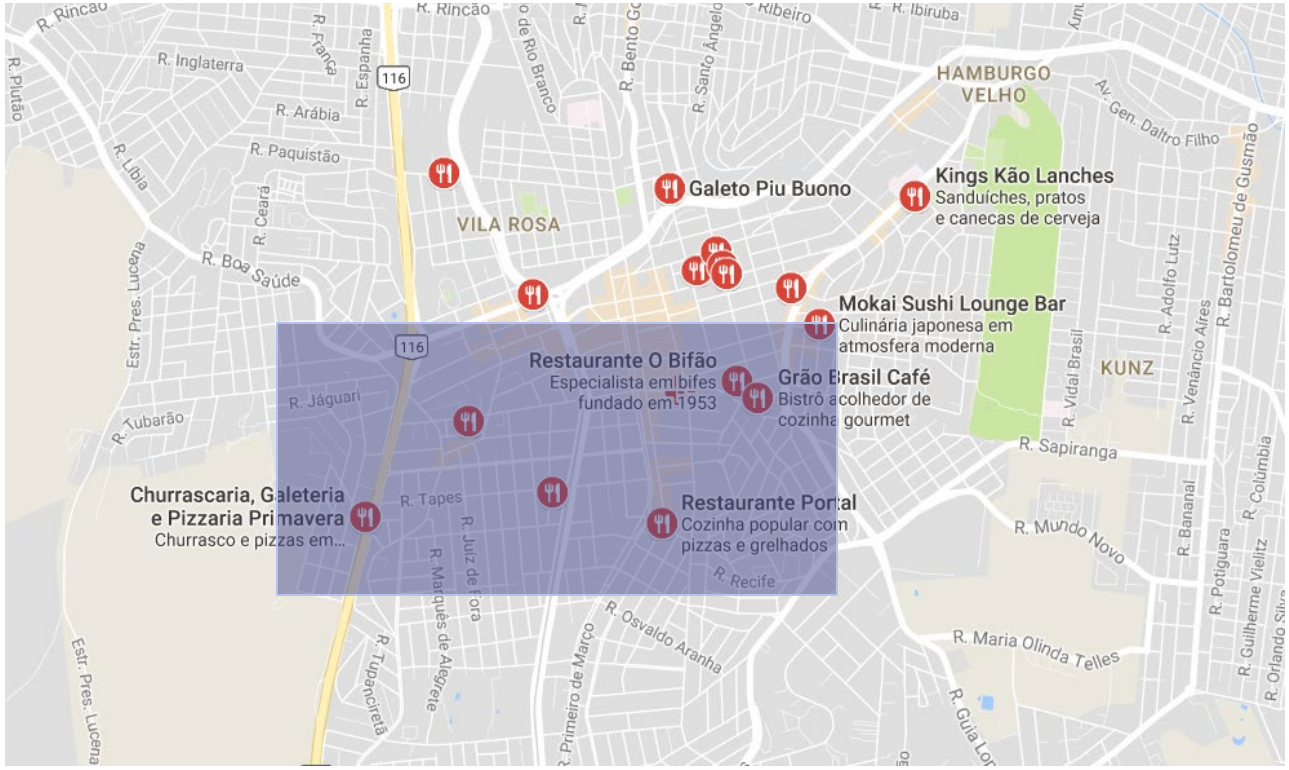
Inserção (cont.)

```
public void insert(K x, K y, V value) {
   root = insert(root, x, y, value);
}

private Node insert(Node node, K x, K y, V value) {
   if (node == null)
      return new Node(x, y, value);
   else if (less(x, node.x) && !less(y, node.y))
      node.NW = insert(node.NW, x, y, value);
   else if (less(x, node.x) && less(y, node.y))
      node.SW = insert(node.SW, x, y, value);
   else if (!less(x, node.x) && !less(y, node.y))
      node.NE = insert(node.NE, x, y, value);
   else if (!less(x, node.x) && less(y, node.y))
      node.SE = insert(node.SE, x, y, value);

   return node;
}
```

```
private boolean less(K k1, K k2) {
   return k1.compareTo(k2) < 0;
}
```

Consultas por Região

- Um dois objetivos da QuadTree é o de fornecer uma consulta eficiente por intervalo, ou seja, selecionando uma região.



Consultas por Região (cont.)

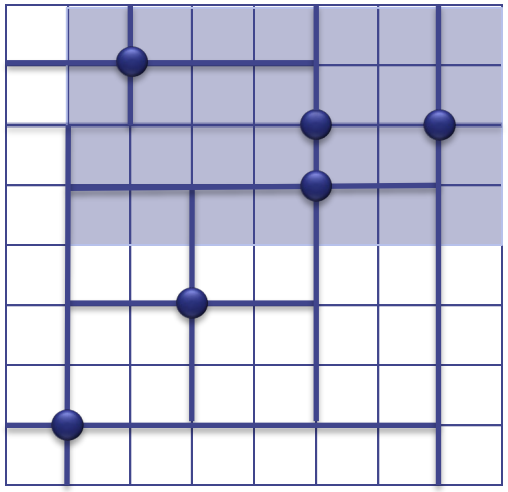
```
public void query2D(Interval2D<K> rect) {
  query2D(root, rect);
}

private void query2D(Node node, Interval2D<K> rect) {
  if (node == null)
     return;
  K xMin = rect.intervalX.min();
  K yMin = rect.intervalY.min();
  K xMax = rect.intervalX.max();
  K yMax = rect.intervalY.max();
  if (rect.contains(node.x, node.y))
      System.out.println("    (" + node.x + ", " + node.y + ") " + node.value);
  if ( less(xMin, node.x) && !less(yMax, node.y)) query2D(node.NW, rect);
  if ( less(xMin, node.x) &&  less(yMin, node.y)) query2D(node.SW, rect);
  if (!less(xMax, node.x) && !less(yMax, node.y)) query2D(node.NE, rect);
  if (!less(xMax, node.x) &&  less(yMin, node.y)) query2D(node.SE, rect);
}
```

Consultas por Região (cont.)

```
public class Interval<K extends Comparable<K>> {
  private final K min;
  private final K max;

  public Interval(K min, K max) {
    this.min = min;
    this.max = max;
  }

  public K min() { return min; }
  public K max() { return max; }

  public boolean contains(K x) {
    return !less(x, min) && !less(max, x);
  }

  private boolean less(K x, K y) { return x.compareTo(y) < 0; }
}
```

```
public class Interval2D<K extends Comparable<K>> {
    public final Interval<K> intervalX;
    public final Interval<K> intervalY;

    public Interval2D(Interval<K> intervalX, Interval<K> intervalY) {
        this.intervalX = intervalX;
        this.intervalY = intervalY;
    }

    public boolean contains(K x, K y) { return intervalX.contains(x) && intervalY.contains(y); }
}
```

Consultas por Região (cont.)

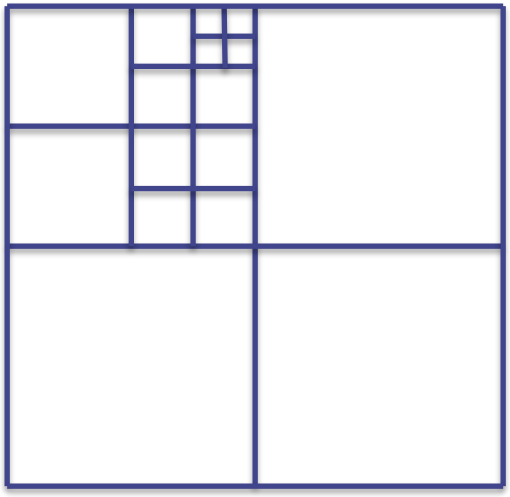


```
(7, 6) 76
(5, 5) 55
(5, 6) 56
(2, 7) 27
```

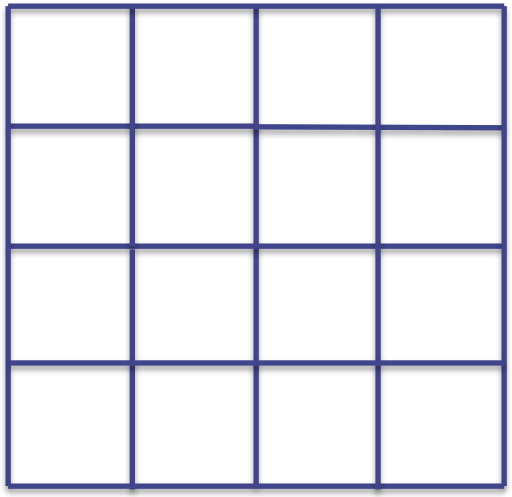
```
public class QuadTreeTest {

  public static void main(String[] args) {
     QuadTree<Integer, Integer> quad = new QuadTree<>();
     quad.insert(7, 6, 76); quad.insert(5, 6, 56);
     quad.insert(1, 1, 11); quad.insert(5, 5, 55);
     quad.insert(2, 7, 27); quad.insert(3, 3, 33);

     Interval2D<Integer> rect = new Interval2D<>(new Interval<>(1, 8), new Interval<>(4, 8));
     quad.query2D(rect);
  }

}
```

Balanceamento

- Uma QuadTree é dita balanceada se qualquer dois nós vizinhos diferem a profundidade em apenas um;



Árvore Desbalanceada



Árvore Balanceada

Complexidade

- Uma QuadTree de profundidade d que armazena um conjunto de n coordenadas pode ser construída em O((d +1)n);

- Em uma QuadTree de profundidade d pode ser encontrada uma vizinhança dado um nó v em O(d + 1);

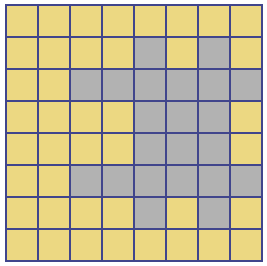
- No melhor caso o tempo de inserção é O(nlog(n)) e, no pior caso, a inserção chega em $O(n^2)$.

QuadTree - Compressão

- Compressão de imagem é uma forma de codificar uma imagem para reduzir o espaço que ocupa em memória ou disco. Similar à árvore de Huffman;
- A compressão com QuadTree assume que há grande probabilidade de regiões vizinhas terem a mesma cor;
- Assim, uma região que tem todos os pixels (elemento da imagem) com a mesma cor, é codificada com apenas 2 bits.

QuadTree – Compressão (cont.)

- Um bitmap é uma matriz de bits que representa uma imagem



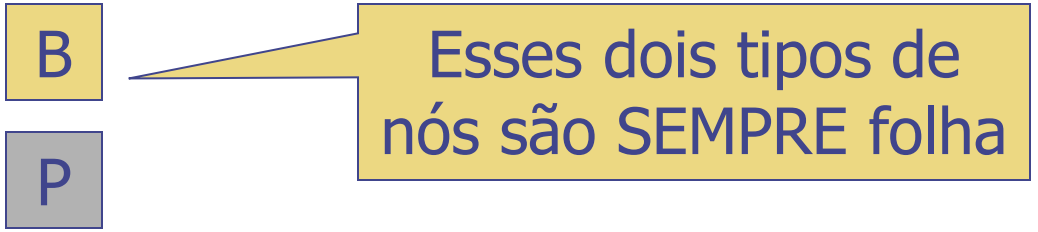
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Podemos definir

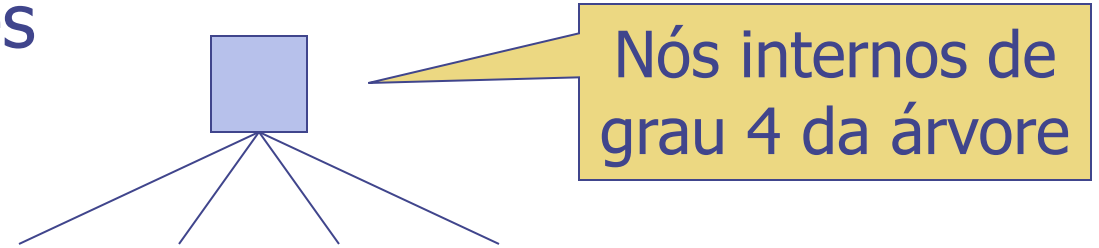
1=preto e
0=branco

QuadTree – Compressão (cont.)

- Há dois tipos de nós na QuadTree:
  - Os nós que correspondem a regiões de uma única cor
    - Branco
    - Preto

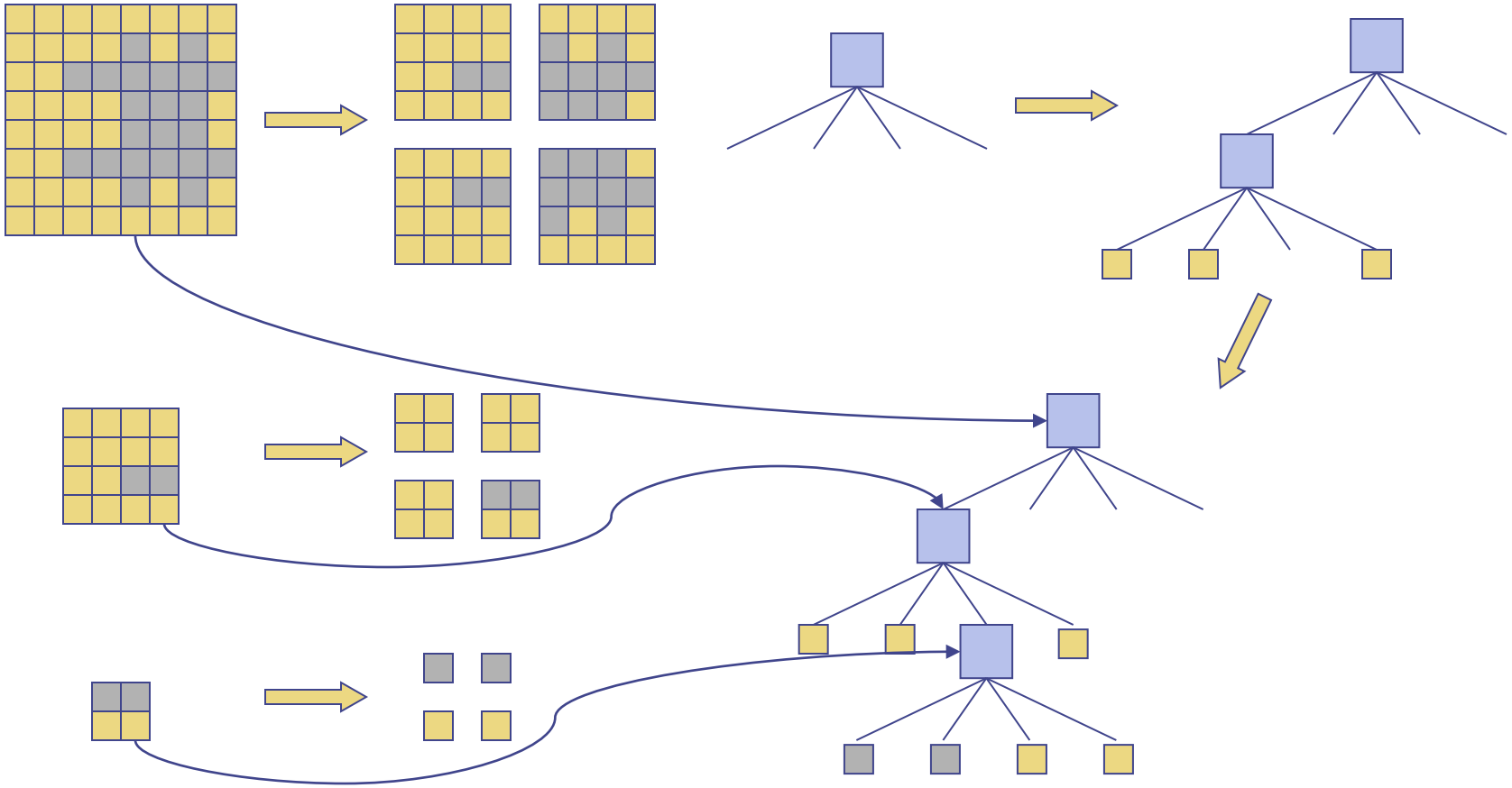


  - Nós que correspondem a regiões com as duas cores

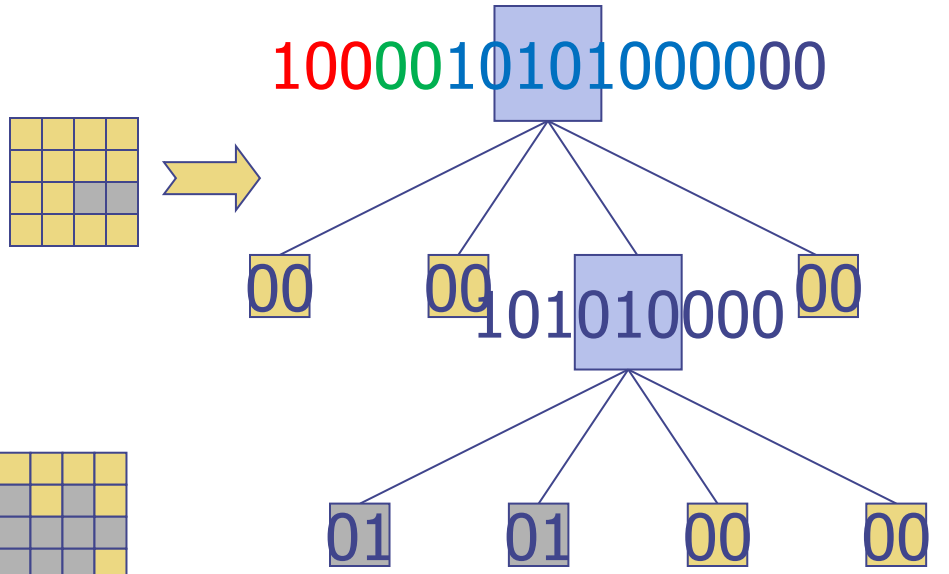


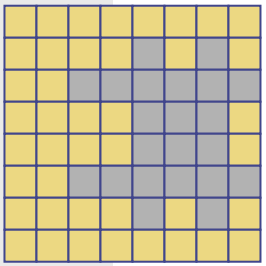
QuadTree – Compressão (cont.)

- O bitmap é particionado sempre que não for de uma só cor



QuadTree – Compressão (cont.)





Codificação:
100001010100000

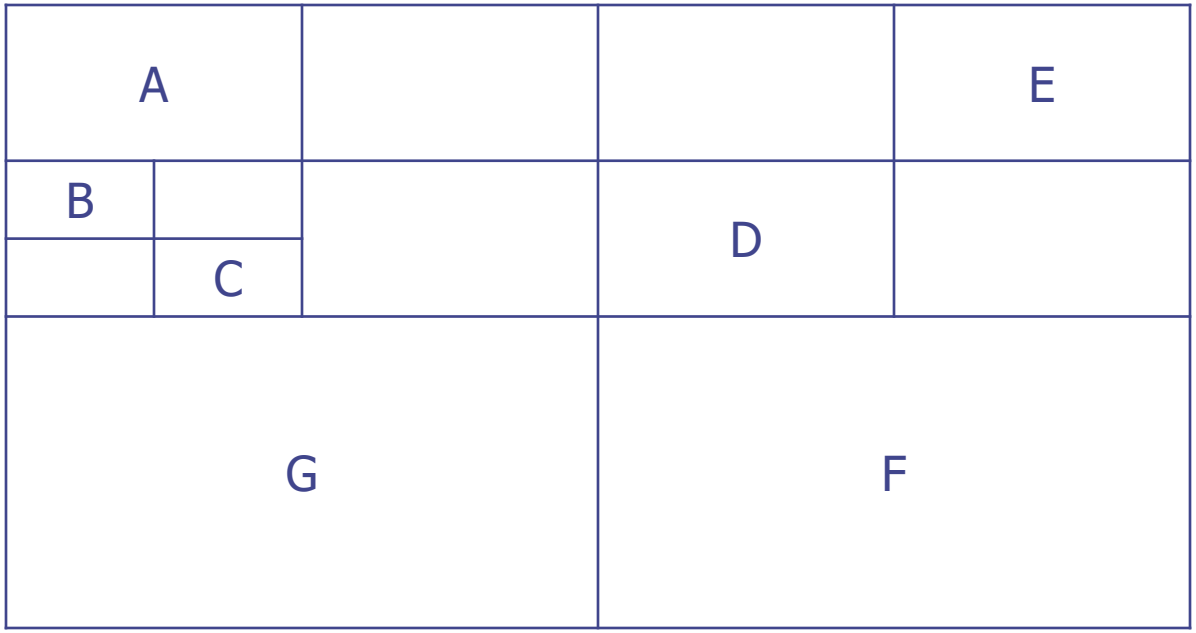
Exercício Prático

- Com base na classe QuadTree, desenvolva os seguintes contratos:

Contrato	Considerações
V search(K x, K y)	**Parâmetro**: coordenada 2D. **Retorno**: valor associado à coordenada. Se não existir, retornar null.
V search(Interval<K> interval)	**Parâmetro**: objeto do tipo intervalo contendo a coordenada. **Retorno**: valor associado à coordenada. Se não existir, retornar null. **Observação**: é uma sobrecarga do primeiro método.
Interval<K> min()	**Parâmetro**: sem parâmetro. **Retorno**: objeto do tipo Interval contendo a menor coordenada da QuadTree. Se não existir nenhum coordenada, retornar null.

Exercícios Teóricos

- **Exercício 1**. Com base na representação 2D da QuadTree abaixo, desenhe a árvore que a representa. Assuma que as informações estarão todas nas folhas.



Exercícios Teóricos

- **Exercício 2**. Suponha que desejamos armazenar a localização das cidades em uma QuadTree. Desenhe o plano 2D (10 x 10) conforme ordem de inserção das seguintes cidades: Porto Alegre (5, 5), Canoas (4, 1), São Leopoldo (6, 8), Novo Hamburgo (9, 6), Campo Bom (8, 2) e Sapiranga (2, 8).

Referências Bibliográficas

- Samet, Hanan. ***The Quadtree and Related Hierarchical Data Structures***. Computing Surveys, Vol.16, No 2, 1984.

- http://algs4.cs.princeton.edu/92search/. Acessado em 19/09/2017.

- Slides Dados Espaciais - QuadTree para Pontos. Estruturas de Dados II. Prof. Jairo Francisco de Souza.