PMR3201 - Computação para Automação
Exercício Programa 2 - 2023

EP2: Árvores 2-d (Ou árvores k-d bidimensionais)

Prof. Thiago de Castro Martins
Prof. Marcos de Sales Guerra Tsuzuki
Prof. Newton Maruyama
Prof. Larissa Driemeier

Deadline: 26/06/2018 - 23h59min

1 Árvores k-d

1.1 Introdução

Uma árvore k-d (k-d*Tree*) é uma árvore binária de busca que armazena pontos em um espaço de k dimensões. Cada nó em uma árvore armazena um único ponto, definido por suas coordenadas k-dimensionais. Como em uma árvore binária de busca convencional, os filhos de um determinado nó são armazenados ou na sub-árvore esquerda ou na sub-árvore direita de acordo com um critério de comparação entre eles e o nó pai. Este critério é o valor de uma das k componentes das coordenadadas de cada filho. A componente específica é alternada de forma cíclica a cada nível. Assim, por exemplo, em uma árvore k-d bidimensional[1] os filhos do nó raiz são comparados com o pai de acordo com a componente x de suas coordenadas. Os filhos dos nós no 2o. nível são separados de acordo com a componente y. Os filhos dos nós no 3o. nível novamente separados de acordo com a componente x. No quarto nível, volta-se a usar a componente y e assim por diante. Como as coordenadas de cada nó têm múltiplas componentes, é possível que haja igualdade da componente usada na comparação de um filho com o pai sem que os pontos sejam idênticos. Neste caso pode-se colocar o filho arbitrariamente na sub-árvore esquerda ou direita. De fato, para aplicações típicas de árvores k-d (e.g.: nuvens de pontos), considera-se que a coincidência exata de coeficientes seja um evento improvável.

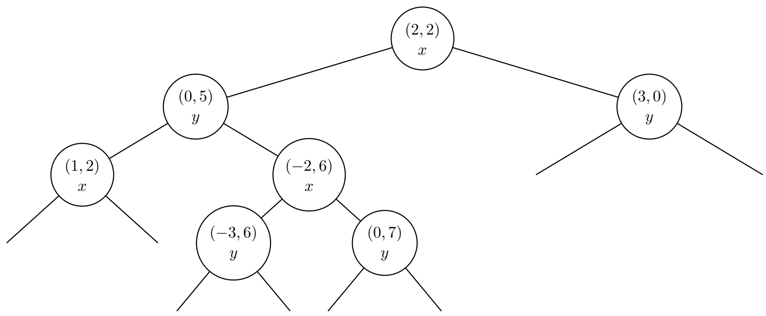

Figura 1: Exemplo de árvore 2-d.

A Figura 1 mostra uma árvore com essa estrutura contendo os pontos $(2, 2)$, $(0, 5)$, $(1, 2)$, $(-2, 6)$, $(-3, 6)$, $(0, 7)$ e $(3, 0)$. Note como todos os pontos da sub-árvore esquerda de $(2, 2)$ têm coordenada x menor *ou igual* a 2. Do mesmo modo, todos os pontos da sub-árvore direita de $(0, 5)$ têm coordenada y maior *ou igual* a 5. Pode-se armazenar no próprio nó a direção segundo a qual as suas sub-árvores estão classificadas, mas isso não é estritamente necessário.

A Figura 2 mostra os pontos armazenados na árvore. As linhas pontilhadas (na cor vermelha quando existe a separação pela coordenada x e na cor azul azul quando existe a separação pela coordenada y) atravessando os nós mostram os limites de cada sub-árvore esquerda/direita.

[1]Embora o "k" do nome seja referente à quantidade de dimensões, curiosamente é tão comum falar-se de "árvore k-d bidimensional" quanto "árvore 2-d"

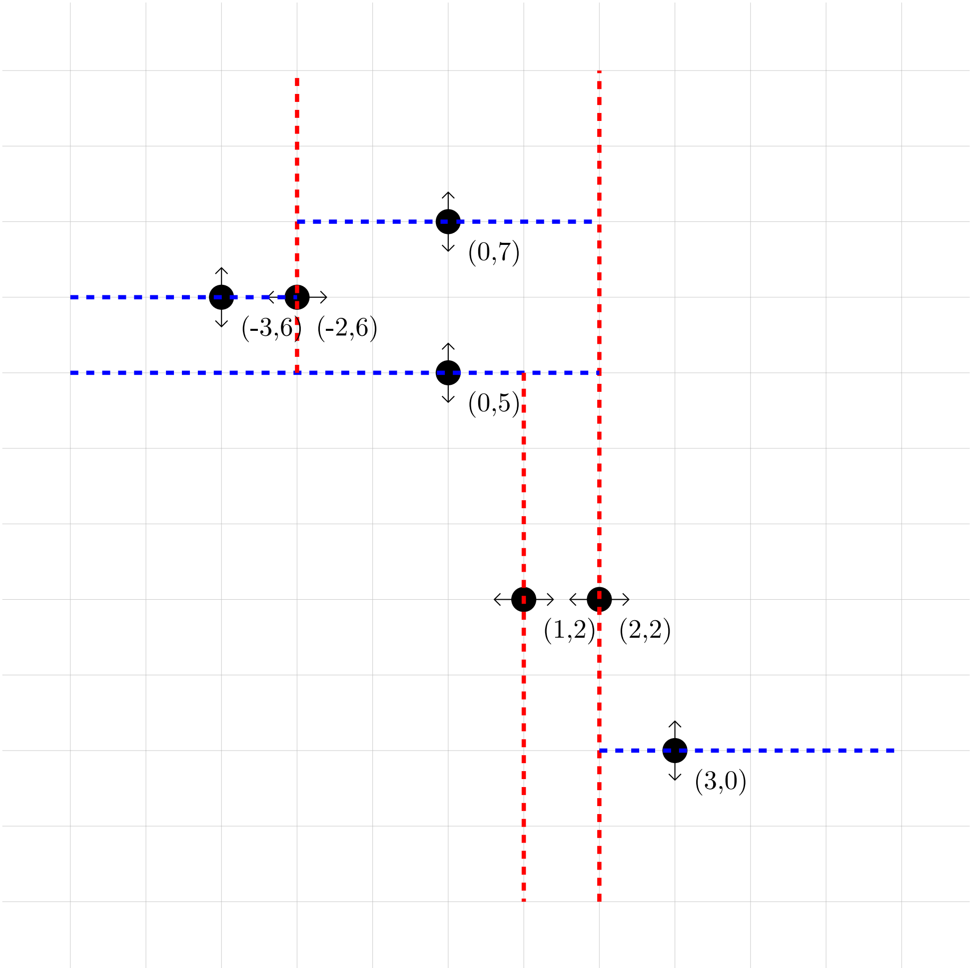

Figura 2: Pontos da árvore da Figura 1.

1.2 Inserção de nós

Considere a inserção do ponto (-1, 4) nesta árvore. Em primeiro lugar, o seu coeficiente x é comparado com o coeficiente x do nó raiz (2, 1). Como o valor é *menor*, ele é inserido na sub-árvore esquerda. Em seguida, o seu coeficiente y é comparado com o coeficiente y do nó (0, 5). Novamente, o valor é menor e a inserção prossegue na sub-árvore esquerda. Finalmente, o seu coeficiente x é comparado com o coeficiente x do nó (1, 2). As Figuras 3 e 4 mostram, respectivamente, as árvores e os pontos resultantes desta inserção.

Observe as regiões delimitadas pelas linhas pontilhadas da Figura 4. A cada uma corresponde uma sub-árvore *vazia* da árvore da Figura 3. Para melhor evidenciar esta correspondência, vamos ilustrá-la. A Figura 5 mostra a mesma árvore com cada sub-árvore vazia numerada. A Figura 6 mostra as regiões correspondentes numeradas.

Naturalmente, um novo ponto será inserido em alguma das sub-árvores vazias. A sub-árvore será a correspondente à região na qual o ponto se encontra. Então, por exemplo, consideremos uma nova inserção, a do ponto $(3, -1)$. A posição deste ponto corresponde à região 8 da Figura 6. Por outro lado, consideremos a sua inserção na árvore da Figura 5. A sua coordenada x é comparada com a do ponto $(2, 2)$, o que leva à inserção na sub-árvore direita. Em seguida, a sua coordenada y é comparada com a do ponto $(3, 0)$, o que leva à inserção na sub-árvore 8, como esperado. As Figuras 7 e 8 mostram a árvore e os pontos após esta operação.

1.3 Limites de sub-regiões

A cada sub-árvore em uma árvore 2-d corresponde uma região retangular com eventuais limites à esquerda, direita, acima e abaixo. Viu-se esta correspondência de forma explícita para sub-árvores vazias na seção anterior, mas ela é

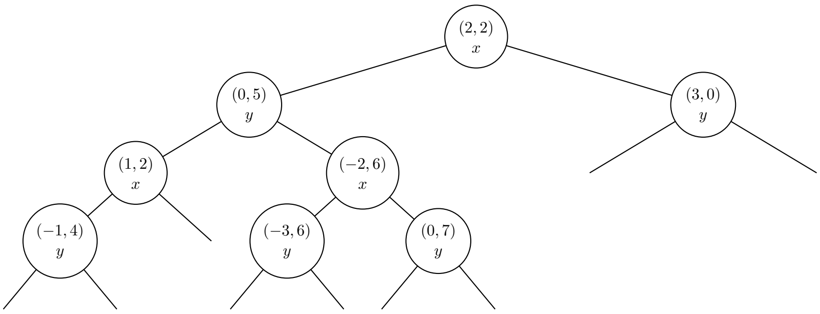

Figura 3: Árvore da Figura 1 após inserção do ponto (-1, 4).

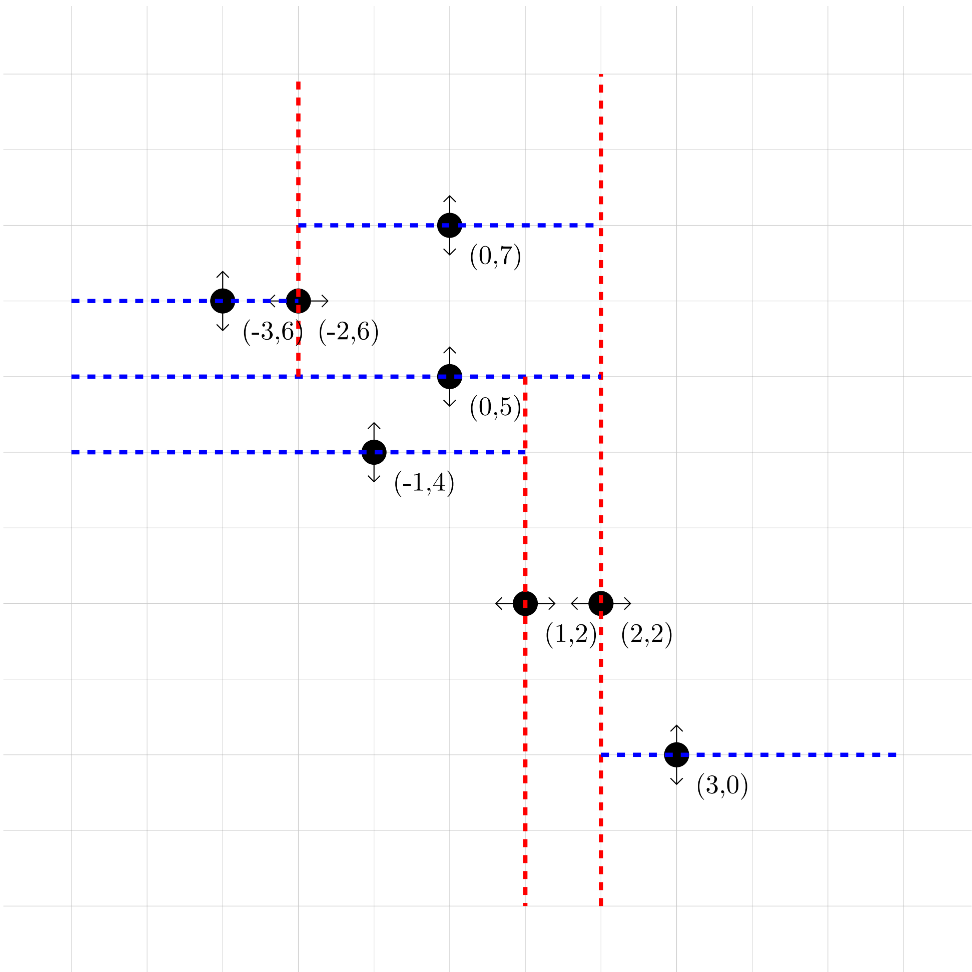

Figura 4: Pontos da árvore da Figura 1 após inserção do ponto (-1, 4).

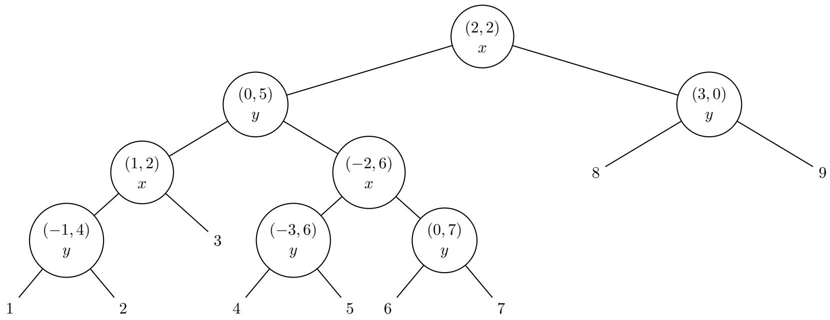

Figura 5: Árvore da Figura 3 com sub-árvores numeradas.

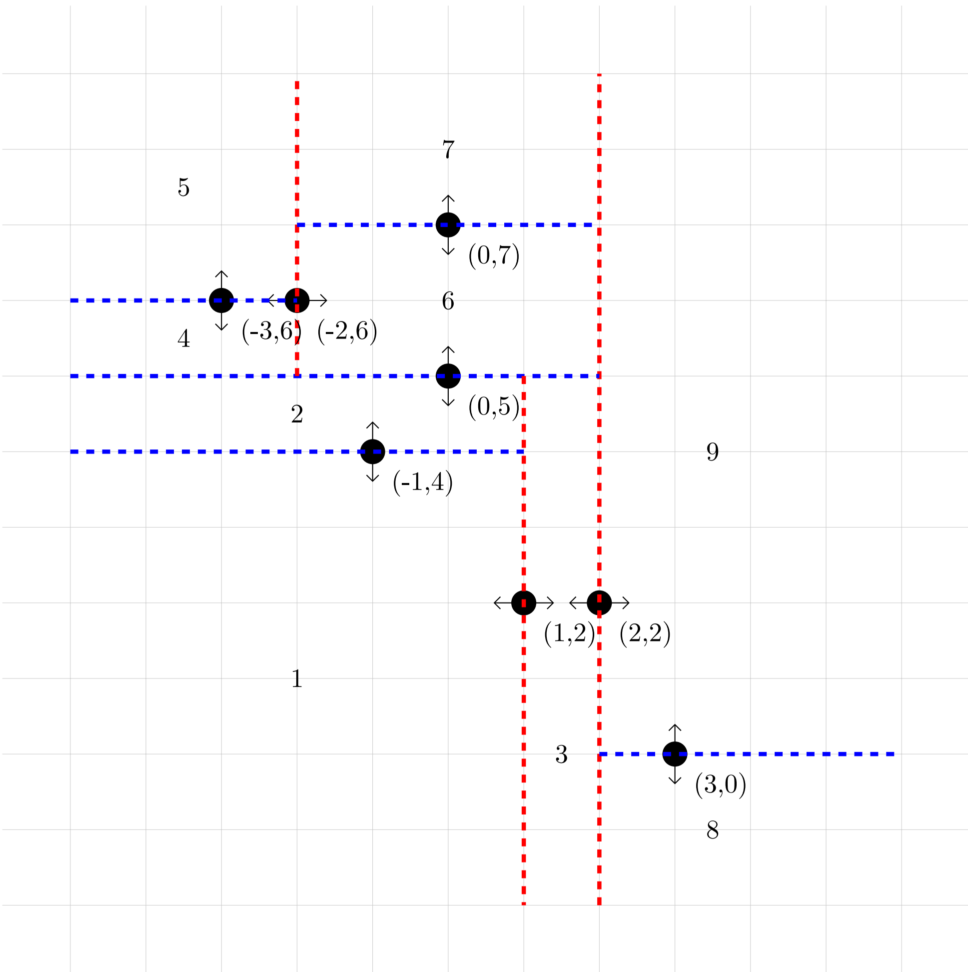

Figura 6: Pontos da árvore da Figura 5 com regiões numeradas.

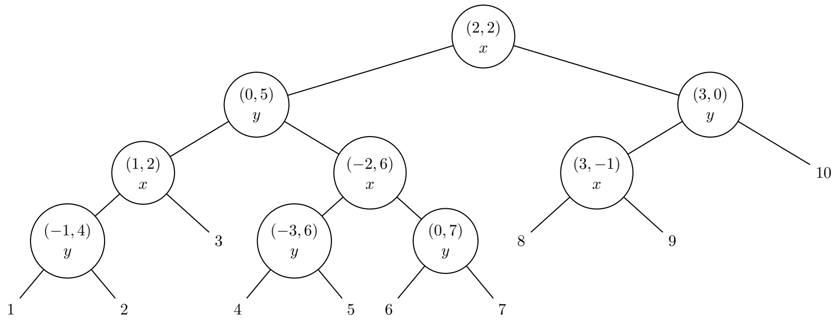

Figura 7: Árvore após inserção do ponto $(3, -1)$.

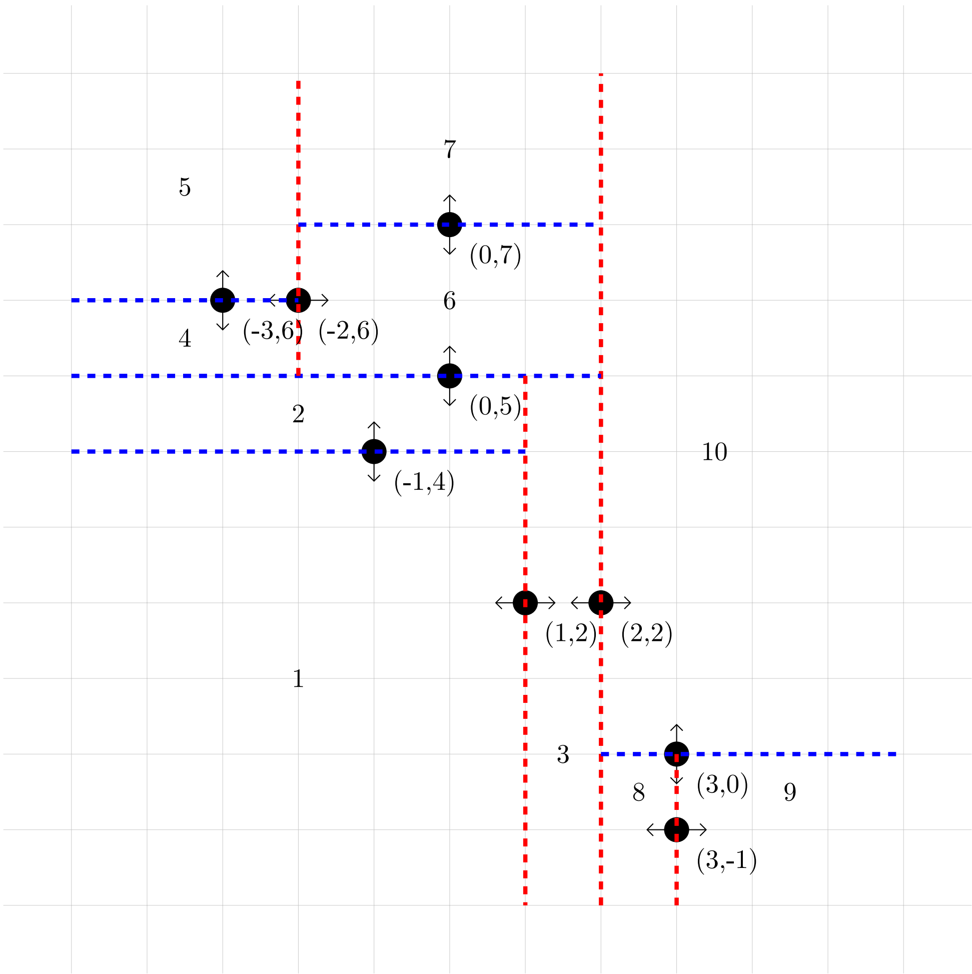

Figura 8: Pontos da árvore após inserção do ponto $(3, -1)$.

válida para qualquer sub-árvore.

Os limites de uma região podem ser obtidos pelo *caminho*, partindo do nó raiz até a sub-árvore vazia correspondente. Inicia-se os limites com $\pm\infty$ tanto na horizontal quanto na vertical. Cada nó no caminho modifica um dos limites.

A título de exemplo, consideremos os limites da região 1 na Figura 8. Ela corresponde à sub-árvore 1 da Figura 7. Os limites iniciais são $[(-\infty,+\infty),(-\infty,+\infty)]$ Partindo-se do nó raiz $(2,2)$, caminha-se à esquerda. Este nó separa suas sub-árvores esquerda e direita de acordo com a coordenada x. Assim, o limite *superior* (visto que caminhou-se para a esquerda) da coordenada x é 2. Os limites passam para $[(-\infty,2),(-\infty,+\infty)]$. Em seguida, caminha-se para a esquerda no nó $(0,5)$. Este é um nó que separa as sub-árvores de acordo com a coordenada y. Deste modo os limites passam para $[(-\infty,2),(-\infty,5)]$. O próximo nó, orientado em x, é $(1,2)$ e caminha-se para a esquerda. Os limites passam para $[(-\infty,1),(-\infty,5)]$. Finalmente, passa-se pela esquerda do nó $(-1,4)$ orientado em y. Os limites finais são $[(-\infty,1),(-\infty,4)]$. Como se vê na Figura 8, a região 1 está limitada a direita pela coordenada $x=1$ e superiormente pela coordenada $y=4$.

De modo similar, a sequência esquerda, direita, direita, esquerda leva aos limites $[(-2,2),(5,7)]$ para a região 6.

Os limites para a sub-árvore cuja raiz é o nó $(-3,6)$ são $[(-\infty,-2),(5,+\infty)]$

1.4 Encontrar todos os pontos dentro de um retângulo

Um problema comum em geometria computacional é encontrar todos os pontos da árvore que estão dentro de uma região retangular (por exemplo, considere um programa de desenho que permite ao usuário selecionar pontos traçando um retângulo com o cursor).

A árvore k-d permite acelerar essa busca restringindo-a a sub-árvores que podem conter os pontos de interesse.

De fato, embora potencialmente seja necessário percorrer todos os nós da árvore nessa busca (afinal, o retângulo pode englobar todos os nós da árvore), é possível eliminar sub-árvores cujas regiões não tem intersecção com o retângulo.

A título de exemplo, considere a busca por todos os pontos no interior do retângulo $[(0,1),(3,3)]$. A busca é iniciada pelo nó raiz. O nó raiz $(2,2)$ pertence ao retângulo. Ambas as suas duas sub-árvores (dos nós $(0,5)$ e $(3,0)$) possuem intersecção com o retângulo, então ambas serão consideradas. O nó $(0,5)$ *não* pertence ao retângulo. Ademais, apenas a sua sub-árvore esquerda (do nó $(1,2)$) possui uma intersecção com o retângulo, então a outra (nó $(-2,6)$) será rejeitada. O nó $(1,2)$ pertence ao retângulo. Ambas as suas sub-árvores possuem uma intersecção com o retângulo, mas a sua sub-árvore direita é vazia. Será considerado assim apenas o nó $(-1,4)$. Este não pertence ao retângulo, e ambas as suas sub-árvores são vazias. Voltando-se à sub-árvore direita do nó raiz, o nó $(3,0)$ não pertence ao retângulo. Apenas a sua sub-árvore direita possui intersecção com o retângulo, mas esta é vazia. A busca assim está encerrada, e apenas os pontos $(2,2)$ e $(1,2)$ são selecionados.

1.5 Encontrar o ponto mais próximo a uma coordenada

Outro problema clássico é encontrar o ponto mais próximo a uma dada coordenada. Não é evidente como a árvore k-d pode ajudar nesse problema, afinal, a simples busca pela região que contém a coordenada não assegura que o ponto mais próximo será considerado.

Por exemplo, considere a busca pelo ponto $(3,5)$ (vide Figura 10). A busca pela região que contém este ponto, partindo do nó raiz, leva a visitar os nós $(2,2)$ e $(3,0)$, mas o verdadeiro ponto mais próximo é o $(0,5)$!

A solução é similar a da Seção 1.4. De fato, uma vez que um ponto é inspecionado, é criado um limite superior de distância para considerar outros pontos. Assim, seguindo o exemplo do ponto $(3,5)$. O ponto raiz $(2,2)$ está a $\sqrt{10}$ do ponto $(3,5)$. Assim, só interessam regiões com alguma intersecção entre o círculo centrado em $(3,5)$ com raio $\sqrt{10}$. *Ambas* as sub-árvores do nó raiz têm intersecção com este círculo. A pesquisa é feita inicialmente pela sub-árvore direita, encabeçada pelo nó $(3,0)$, por que é mais plausível que pontos próximos de $(3,5)$ estejam nesta sub-árvore. O ponto $(3,0)$ está mais distante do que o ponto $(2,2)$, então é descartado. A sua sub-árvore esquerda *não* tem interesecção com o círculo, então é descartada. A sua sub-árvore direita é vazia. Esta etapa de busca é ilustrada na Figura 11.

A busca continua na sub-árvore esquerda do nó raiz, encabeçada por $(0,5)$. O nó está a 3 unidades de distância, então é selecionado. Neste momento, o raio da região circular a ser considerada é alterado para 3. Ambas as sub-árvores de $(0,5)$ têm intersecção com o círculo. Ambas são igualmente promissoras (afinal, a coordenada y dos pontos é a mesma). Arbitrariamente, inicia-se a busca pela sub-árvore esquerda, cujo nó raiz é $(1,2)$. Este nó está fora do círculo, então é descartado. A sua sub-árvore direita seria mais promissora, porém é vazia. A sua sub-árvore esquerda contém o nó $(-1,4)$ que é mais distante, então é desconsiderado. Ambas as sub-árvores de $(-1,4)$ são vazias. Volta-se

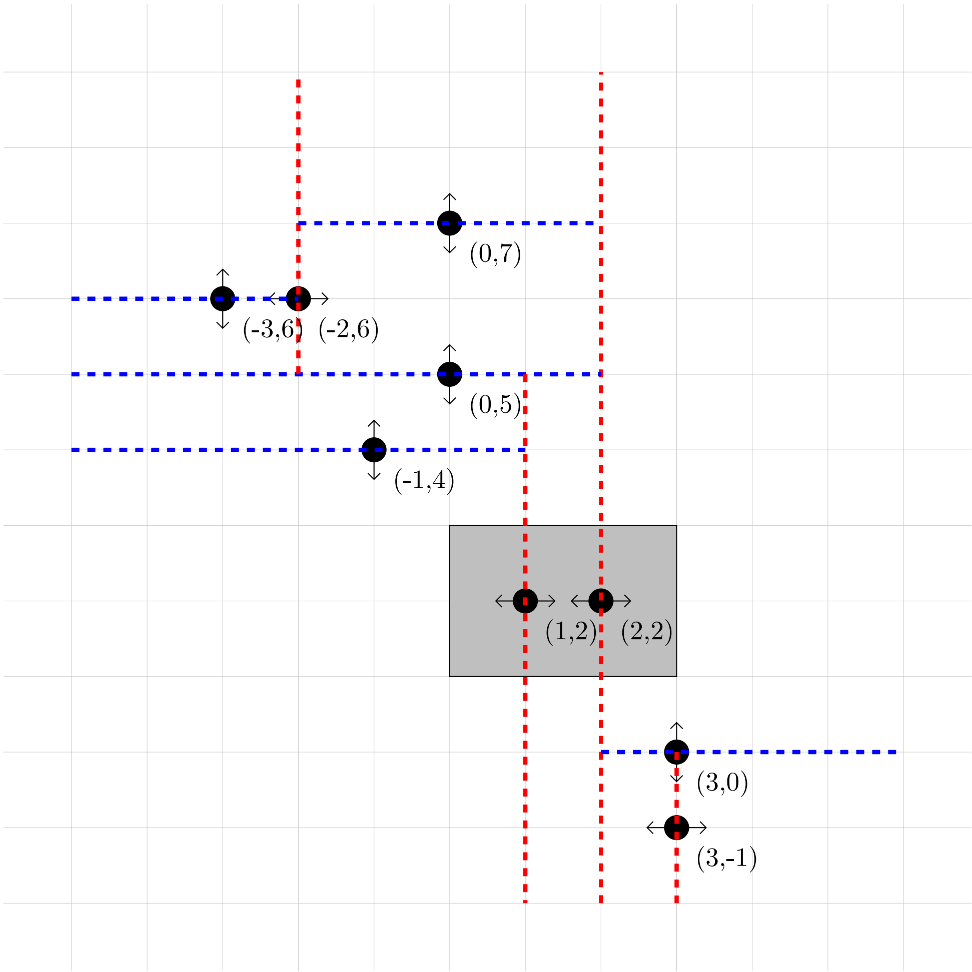

Figura 9: Busca por todos os pontos dentro de um retângulo.

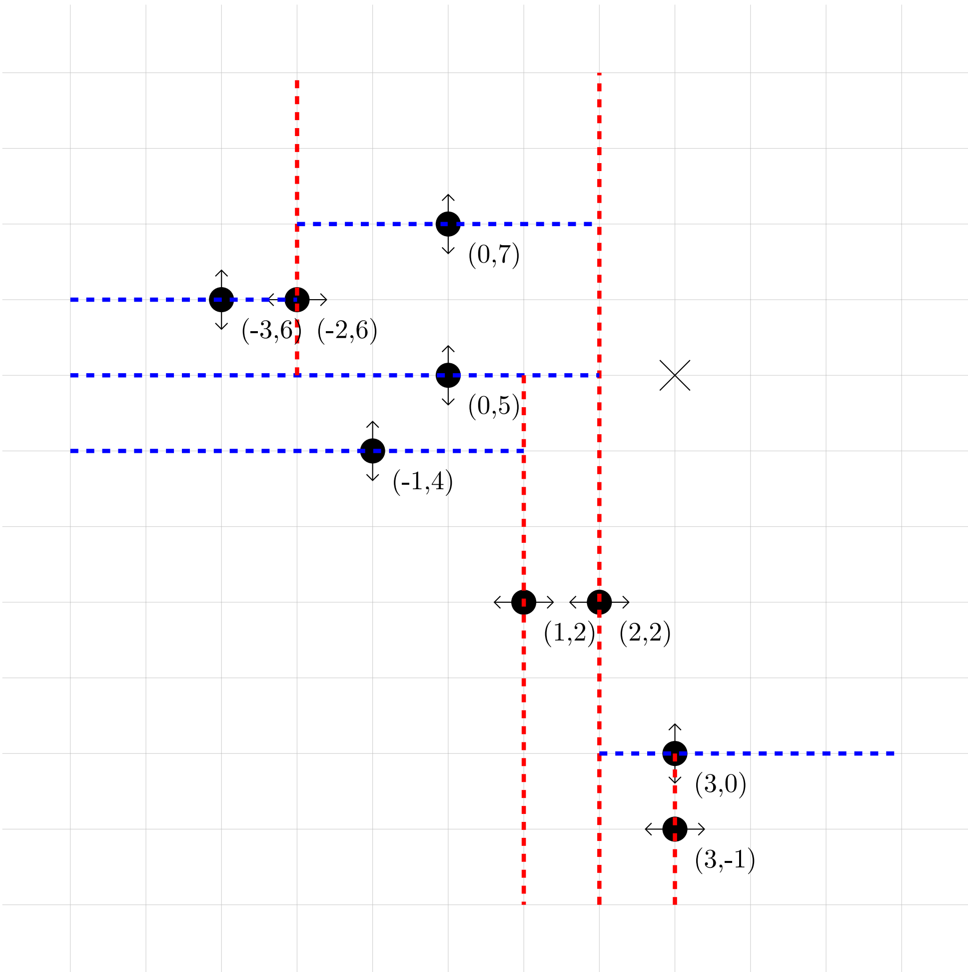

Figura 10: Busca pelos ponto mais próximo a (3,5).

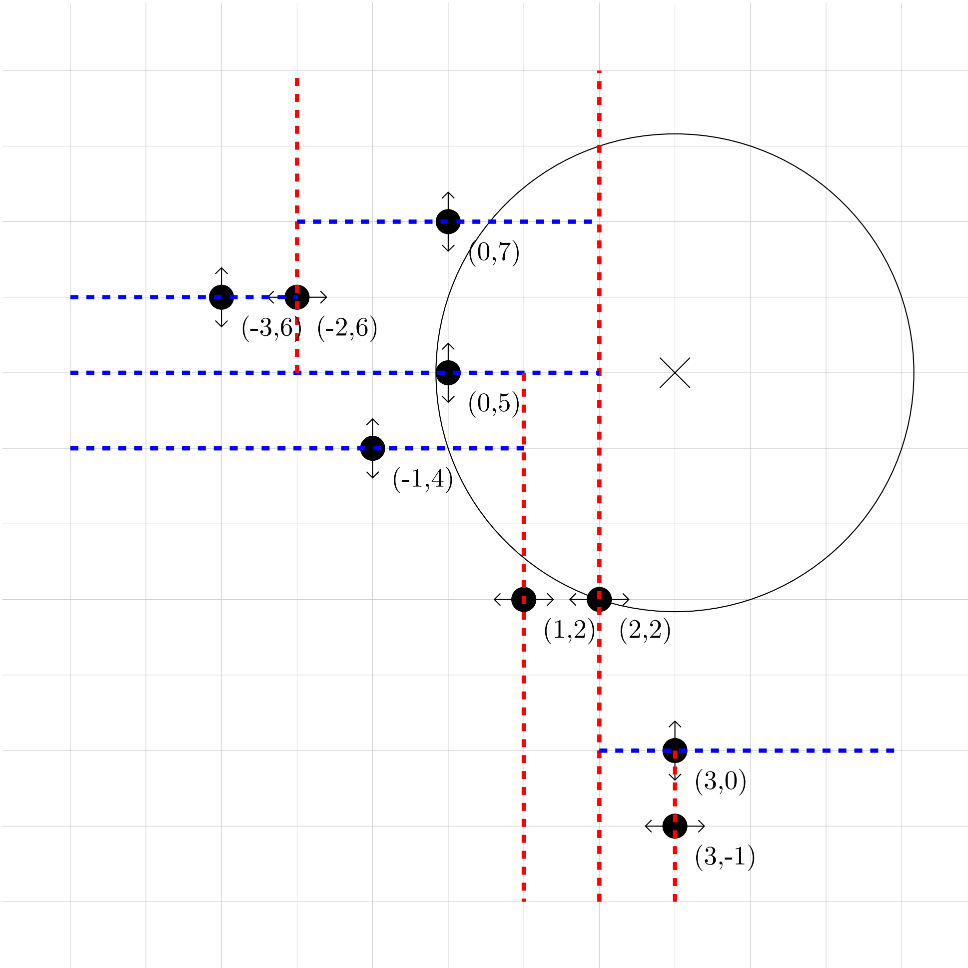

Figura 11: Busca pelos ponto mais próximo a (3,5), etapa 1.

à sub-árvore direita de $(0, 5)$, encabeçada por $(-2, 6)$. Este nó está fora do círculo, portanto é descartado. A sua sub-árvore direita, encabeçada por $(0, 7)$, tem intersecção com o círculo. O ponto $(0, 7)$ está fora do círculo, então é descartado. Ambas as suas sub-árvores são vazias. A sub-árvore esquerda de $(-2, 6)$ não possui intersecção com o círculo, portanto é descartada e a busca acaba.

2 Para você fazer

A sua solução deve ser desenvolvida no arquivo jupyter notebook `PMR3201EP2V2023.ipynb` que contém uma célula com o código base de definição da classe `NoArvore2D`. Cada tarefa proposta deve ser desenvolvida em células independentes que já se encontram definidas.

2.1 Código base: A classe `NoArvore2D`

A classe `NoArvore2D`, cuja listagem está na Fig. 13, mostra a implementação de uma classe que modela o nó de uma árvore 2d com uma função de inserção de novo nó.

Cada objeto desta classe possui os seguintes atributos:

- `_x`: Componente x das coordenadas do ponto do nó.
- `_y`: Componente y das coordenadas do ponto do nó.
- `_e`: Uma referência à sub-árvore esquerda. Contém `None` caso a sub-árvore esquerda esteja vazia.

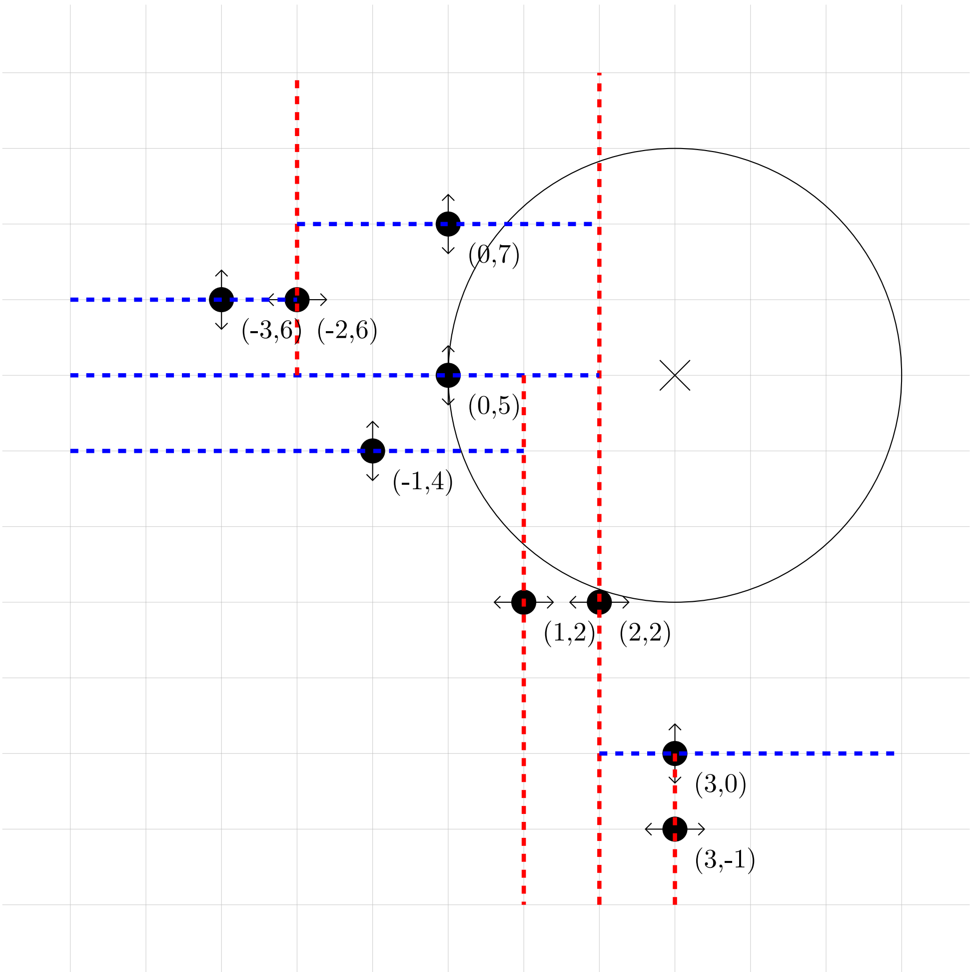

Figura 12: Busca pelos ponto mais próximo a (3,5), etapa 2.

```python
class NoArvore2D:
  """ Implementa um nó de árvore 2d (k-d bidimensional)
  """
  def __init__(self, x, y):
    """ Cria um novo nó. x e y são as coordenadas do nó.
    """
    self._x = x        # Coordenada x
    self._y = y        # Coordenada y
    self._e = None     # Sub-árvore esquerda
    self._d = None     # Sub-árvore direita

  # Este é o método recursivo que faz a inserção de um novo nó
  # Ele é chamada pelo método insere
  def _insere_rec(self, x, y, horizontal):
    if (horizontal and self._x > x) or ((not horizontal) and self._y > y):
      if self._e is None:
        self._e = NoArvore2D(x, y)
      else:
        self._e._insere_rec(x, y, not horizontal)
    else:
      if self._d is None:
        self._d = NoArvore2D(x, y)
      else:
        self._d._insere_rec(x, y, not horizontal)

  def insere(self, x, y):
    """ Insere um novo nó. x e y são as coordenadas do nó a ser inserido na
    árvore."""
    self._insere_rec(x, y, True)
```

Figura 13: Listagem da classe `NoArvore2D`.

- `_d`: Uma referência à sub-árvore direita. Contém `None` caso a sub-árvore direita esteja vazia.

Os métodos da classe são:

- `__init__(self, x, y`: Cria um novo nó cujas coordenadas são dadas pelo par `x` e `y`.
- `insere(self, x, y, usax=True)`: Cria e insere um novo nó na árvore cuja raiz é `self`. As coordenadas do novo nó são dadas através dos parâmetros `x` e `y`. Percebe-se que o eixo usado para classificar as sub-árvores de um nó *não* é armazenado na árvore. Usa-se a própria estrutura da mesma e a profundidade das operações recursivas para determinar se as sub-árvores serão classificadas segundo a sua coordenada x ou y. Assim, há um 4o. parâmetro neste método, `usax`, que é verdadeiro quando o nó sob o qual este método está sendo invocado usa o eixo x para classificar as suas sub-árvores e falso quando usa o eixo y. Na árvore proposta, o nó *raiz* deve usar o eixo x. Não há uma maneira trivial de se determinar qual o eixo usado pelos outros nós da árvore. Assim, a maneira robusta de se usar o método insere é invocá-lo diretamente *apenas* no nó raiz[2] e com o valor padrão do parâmetro (ou seja, omitindo-o).

Tarefa 1: Verificação de intersecção entre dois retângulos

Construa uma função em Python que determina se dois retângulos paralelos aos eixos coordenados possuem uma intersecção. Os retângulos são definidos por suas mínimas e máximas coordenadas ao longo dos eixos x e y. Os retângulos devem ser considerados conjuntos *compactos*, ou seja, a sua borda *faz* parte do retângulo. Assim, por exemplo, há intersecção entre o retângulo delimitado por $[(-3,1),(0,2)]$ e $[(1,3),(1,3)]$ no segmento de reta que liga o ponto $(1,1)$ ao ponto $(1,2)$.

[2] Uma maneira de deixar esta interface mais robusta seria criar uma nova classe `NoRaizArvore2D` e apenas nesta oferecer um método público `insere`

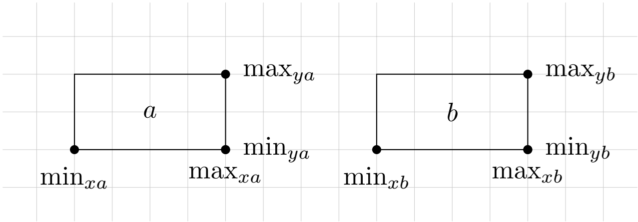

Figura 14: Retângulos a e b e suas coordenadas mínimas e máximas.

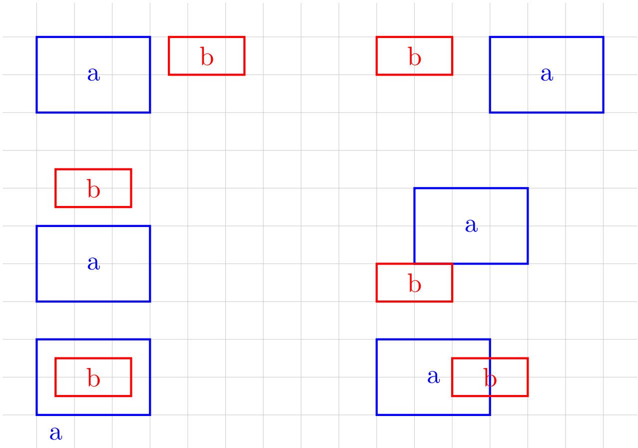

Figura 15: Exemplos de algumas posições relativas que os retângulos a e b podem assumir.

Use a seguinte assinatura:

```
def ha_interseccao(minxa, maxxa, minya, maxya, minxb, maxxb, minyb, maxyb):
```

onde `minxa`, `maxxa`, são respectivamente os limites inferiores e superiores das coordenadas x do primeiro retângulo a, `minya`, `maxya` são os limites inferiores e superiores das coordenadas y do primeiro retângulo a, `minxb`, `maxxb`, `minyb` e `maxyb` são os valores correspondentes do segundo retângulo b. A Figura 14 ilustra os retângulos a e b e suas coordenadas mínimas e máximas.

A Figura 15 ilustra algumas posições relativas que os retângulos a e b podem assumir.

Sua função deve retornar `True` caso exista alguma intersecção entre os retângulos (ainda que somente nas bordas) e `False` caso contrário.

Teste seu código com os retângulos definidos pelos limites:

1. $[(-3,1),(0,2)]$ e $[(1,3),(1,3)]$
2. $[(-2,-1),(-2,-1)]$ e $[(1,2),(1,2)]$
3. $[(-3,3),(-1,1)]$ e $[(-1,1),(2,3)]$
4. $[(-3,1),(-2,2)]$ e $[(2,3),(0,3)]$
5. $[(0,1),(0,1)]$ e $[(1,2),(1,2)]$
6. $[(-4,4),(-2,2)]$ e $[(-1,1),(-1,1)]$
7. $[(-4,4),(-2,2)]$ e $[(-2,2),(-4,4)]$
8. $[(-2,1),(-2,1)]$ e $[(-1,2),(-1,2)]$

Tarefa 2: Busca por pontos dentro de um retângulo

Adicione à classe `NoArvore2D` o método ProcuraRect que encontra todos os pontos de uma árvore 2d que estão dentro de uma região retangular paralela aos eixos coordenados.

O algoritmo a ser adotado é:

- Verifique se o nó raiz pertence ao retângulo.
- Verifique se as sub-árvores possuem alguma intersecção com o retângulo (use para isso a função da seção 2.1 e o processo de delimitação de região de sub-árvores descrito na seção 1.3).
- Repita o processo em cada sub-árvore cuja região possui intersecção com o retângulo.

Use a constante `math.inf` para expressar as regiões de árvores que não estão limitadas em alguma direção.

Use a seguinte assinatura:

```
def procura_rect(self, minx, maxx, miny, maxy, func):
```

onde `self` é o nó raiz da árvore, `minxb`, `maxxb`, `minyb` e `maxyb` os limites do retângulo e `func` é uma *função* que recebe dois argumentos. Esta função deve ser invocada pelo método procuraRect para cada ponto encontrado dentro do retângulo e deve receber as coordenadas x e y de cada ponto encontrado.

Sugestão: Lembre-se de que o eixo usado na classificação *não* está armazenado nos nós da árvore. Você pode presumir que sua função só será chamada no nó raiz (cuja classificação é feita usando a componente x) e adicionar um método auxiliar recursivo com um parâmetro extra nos moldes do método `insere`.

Reproduza o exemplo da Fig. 9 (**Atenção com a ordem de inserção dos nós na árvore!**).

Tarefa 3: Interseção entre círculo e retângulo

Construa uma função em Python que determina se um retângulo paralelo aos eixos coordenados e um círculo fechado possuem alguma intersecção.

Use a seguinte assinatura:

```
def int_circ_rect(minx, maxx, miny, maxy, cx, cy, r2):
```

onde `minxb`, `maxxb`, `minyb` e `maxyb` são limites do retângulo, `cx` e `cy` são as coordenadas do centro do círculo e `r2` é o *quadrado* do raio do círculo (usar o quadrado do raio elimina uma complexa operação de raiz quadrada no próximo exercício).

Sua função deve retornar `True` caso exista alguma intersecção entre o retângulo e o círculo (ainda que somente nas bordas) e `False` caso contrário.

Teste seu código com os seguintes exemplos:

1. Limites do retângulo: $[(-3, 1), (0, 2)]$

   Centro do círculo: $(0, -1)$

   Raio ao quadrado do círculo: 4

2. Limites do retângulo: $[(0, 1), (0, 1)]$

   Centro do círculo: $(2, 2)$

   Raio ao quadrado do círculo: 2

3. Limites do retângulo: $[(-5, 5), (0, 1)]$

   Centro do círculo: $(0, 4)$

   Raio ao quadrado do círculo: 25

4. Limites do retângulo: $[(-5, 5), (0, 5)]$

   Centro do círculo: $(0, 2)$

Raio ao quadrado do círculo: 1

5. Limites do retângulo: $[(-1,1),(-1,1)]$
   Centro do círculo: $(0,0)$
   Raio ao quadrado do círculo: 9

6. Limites do retângulo: $[(-3,-1),(-3,-1)]$
   Centro do círculo: $(2,2)$
   Raio ao quadrado do círculo: 16

7. Limites do retângulo: $[(3,4),(-1,1)]$
   Centro do círculo: $(1,0)$
   Raio ao quadrado do círculo: 1

8. Limites do retângulo: $[(-2,2),(-4,-3)]$
   Centro do círculo: $(-1,0)$
   Raio ao quadrado do círculo: 1

Tarefa 4: Busca por ponto mais próximo

Adicione à classe `NoArvore2D` o método `buscaMaisProximo` que encontra o ponto na árvore mais próximo a um outro ponto.

Este método deve usar um procedimento recursivo para encontrar o ponto mais próximo dentre os armazenados na sub-árvore até outro ponto, chamado aqui de "alvo".

A função recursiva recebe o nó raiz de uma árvore, as coordenadas do alvo, as coordenadas do nó mais próximo ao alvo até então encontrado e a sua distância ao quadrado até o alvo (estes valores são inicializados com $+\infty$ na primeira chamada). A função retorna o ponto mais próximo até o alvo encontrado, dentre todos os pontos armazenados na árvore e o ponto mais próximo encontrado antes da chamada recursiva (note que é possível que a função simplesmente retorne o ponto original caso a sub-árvore sobre a qual ela foi encontrada não contenha nenhum ponto mais próximo), bem como sua distância ao quadrado (o uso da distância ao quadrado aqui elimina a necessidade do cálculo de uma raiz quadrada).

O algoritmo recursivo é como se segue:

1. Caso o nó raiz da árvore atual esteja mais próximo do que o ponto encontrado até então, este passa a ser o ponto mais próximo e sua distância ao quadrado passa a ser a nova distância ao quadrado considerada.

2. Determine em que sub-árvore o ponto desejado seria inserido. Esta será chamada de sub-árvore "próxima" e a outra de sub-árvore "distante".

3. Se há intersecção entre o círculo centrado no ponto alvo com raio igual à distância atualmente considerada, invoque a função recursivamente na sub-árvore próxima. Use a função desenvolvida na seção 2.1.

4. Verifique se há intersecção entre a sub-árvore "distante" e o círculo centrado no ponto alvo. O raio do círculo é potencialmente *diferente* do usado na etapa anterior, visto que a distância ao quadrado do ponto mais próximo pode ter sido alterada pelo resultado da chamada recursiva naquele item. Se há intersecção, invoque a função recursivamente na sub-árvore "distante".

5. Retorne as coordenadas e a distância ao quadrado do ponto mais próximo encontrado.

O seu método deve usar a seguinte assinatura:

```
def busca_mais_proximo(self, x, y):
```

onde `self` é o nó raiz da árvore 2d sobre o qual será invocada a função e `x, y` são as coordenadas do ponto alvo. A função deve retornar as coordenadas do ponto mais próximo da árvore até o alvo.

Observe que esta assinatura é insuficiente para passar todos os parâmetros do algoritmo descrito anteriormente (em particular o eixo usado para classificar as sub-árvores de cada nó, as coordenadas do ponto mais próximo encontrado até então e sua distância ao quadrado até o alvo). Você deverá implementar um método auxiliar recursivo para este algoritmo.

Teste o seu código com os pontos $(0,0)$, $(-3,2)$, $(3,3)$, $(-3,0)$, $(-6,5)$, $(1,3)$, $(2,5)$ (-7,3),(2,-1),(3,0)*inseridos nesta ordem.Pro*
1, 4).*Mostrequaisospontosvisitadospelabuscaeemqueordem.*

Avalie o desempenho de sua função em uma árvore de nós gerados aleatoriamente no espaço ± 10 em x e y. Plote a quantidade total de chamadas da função `int_circ_rect` *versus* a quantidade total de nós. Avalie a função árvores de 10, 100, 1000 e 10000 nós. Para cada uma destas árvores, selecione 10 pontos alvo aleatórios e escolha a *média* da quantidade de chamadas a `int_circ_rect` obtida para um dado tamanho de árvore. Embora o pior caso do algoritmo seja linear, como você estima empiricamente a sua complexidade média?

Sugestão: Como é trivial percorrer *todos* os nós da árvore, implemente antes uma versão "trivial" do algoritmo que enumera todos os nós e pega simplesmente o nó mais próximo encontrado. Use os resultados desta função trivial para verificar o resultado da sua função recursiva.