Aula 16

Buscas ortogonais

Cap 5 do livro de de Berg et al.

Consultas em bancos de dados

BD com informações sobre funcionários de uma empresa:
nome, endereço, data de nascimento, salário, etc.

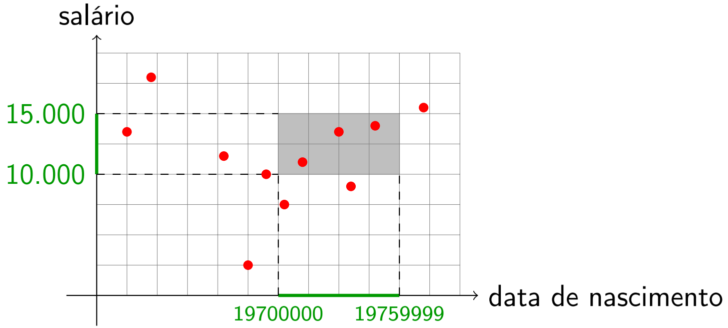
Exemplo de consulta: liste todos os funcionários nascidos entre 1970 e 1975 com salário entre R$$10.000,00$ e R$$15.000,00$.

Consultas em bancos de dados

BD com informações sobre funcionários de uma empresa:
nome, endereço, data de nascimento, salário, etc.

Exemplo de consulta: liste todos os funcionários nascidos entre 1970 e 1975 com salário entre R$10.000,00 e R$15.000,00.

Interpretação geométrica: cada funcionário é um ponto, com data de nascimento, codificada como um número, como x-coordenada, e salário como y-coordenada.

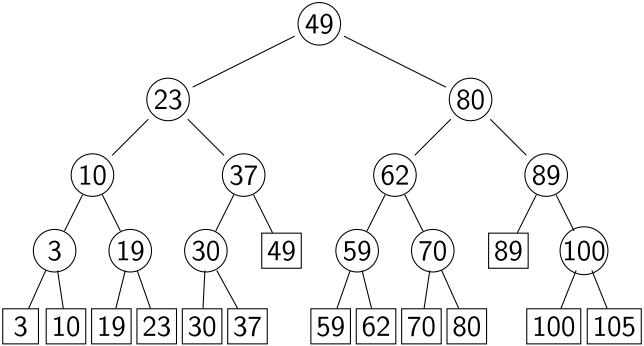


Busca unidimensional por intervalo: 1D range queries

S: conjunto de n números

ABB com números de S nas folhas

Nós internos guardam máximo da subárvore esquerda.



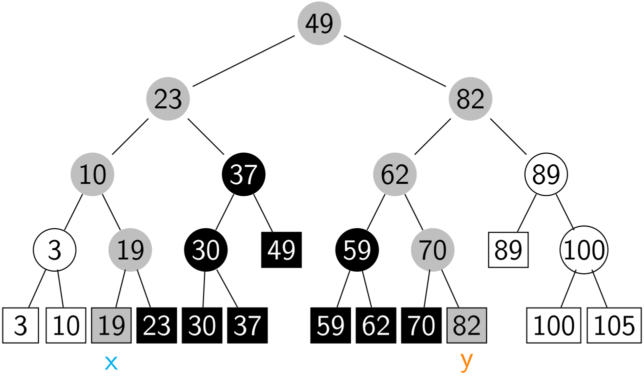
Valor num nó interno divide as folhas aproximadamente ao meio.

Busca unidimensional por intervalo

Busca por números de S no intervalo $[a, b]$:

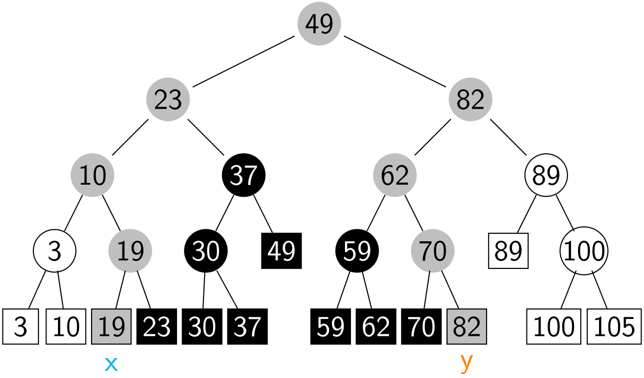
Busque a e b na ABB, terminando nas folhas x e y.

Exemplo: para $a = 15$ e $b = 80$, terminamos com $x = 19$ e $y = 82$.



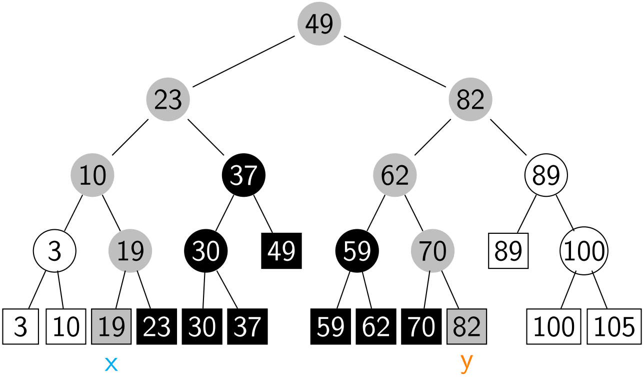
Resposta: folhas entre x e y, possivelmente incluindo x e y

Consumo de tempo



Ache o vértice s onde os caminhos para x e y bifurcam e,
de s para x, imprima todas as folhas de subárvores à direita e,
de s para y, imprima todas as folhas de subárvores à esquerda.

Consumo de tempo

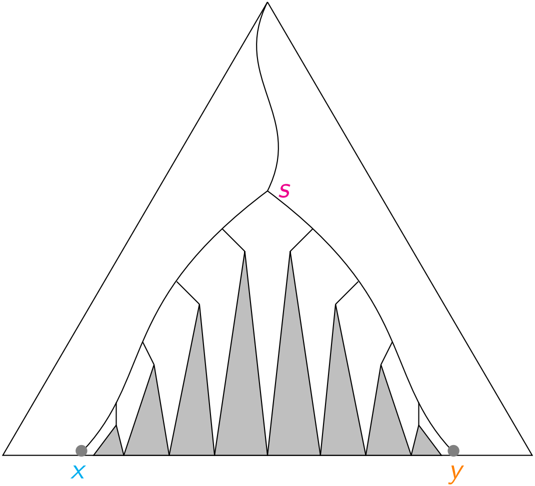


Ache o vértice s onde os caminhos para x e y bifurcam e,
de s para x, imprima todas as folhas de subárvores à direita e,
de s para y, imprima todas as folhas de subárvores à esquerda.

$\mathrm{O}(\lg n + k)$, onde k é o número de elementos de S em $[a, b]$.

Esquema

Ache o vértice s onde os caminhos para x e y bifurcam e,
de s para x, imprima todas as folhas de subárvores à direita e,
de s para y, imprima todas as folhas de subárvores à esquerda.



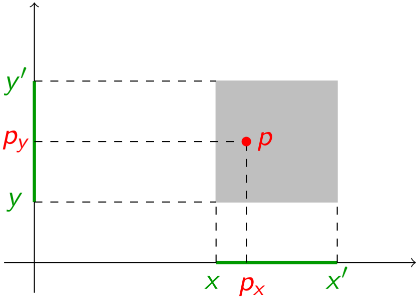
KD-trees

Espaço 2-dimensional, buscas ortogonais.

P: conjunto de pontos (sem empates nas coordenadas x ou y)

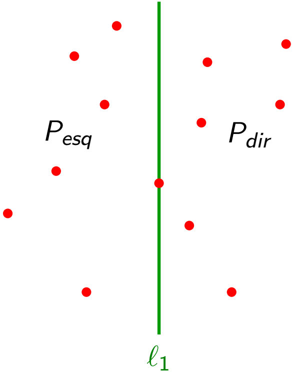
Consulta retangular:
Quais pontos de P estão em $[x : x'] \times [y : y']$?

Ponto p está em $[x : x'] \times [y : y']$ se $p_x \in [x : x']$ e $p_y \in [y : y']$.



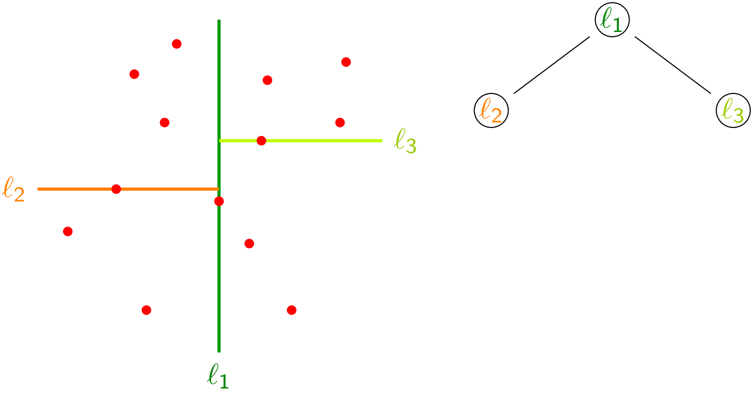
KD-trees

Árvore de busca binária que divide os pontos alternadamente, ora por uma linha vertical, ora por uma linha horizontal.



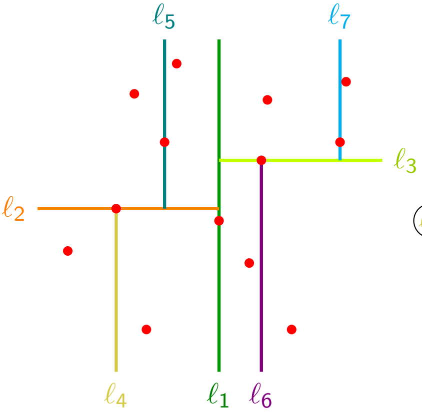
KD-trees

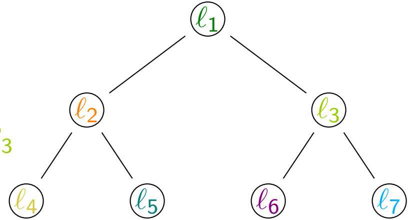
Árvore de busca binária que divide os pontos alternadamente, ora por uma linha vertical, ora por uma linha horizontal.



KD-trees

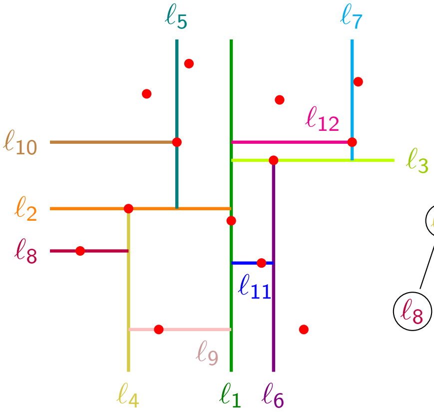
Árvore de busca binária que divide os pontos alternadamente, ora por uma linha vertical, ora por uma linha horizontal.

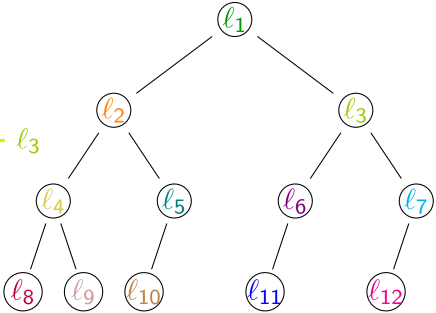




KD-trees

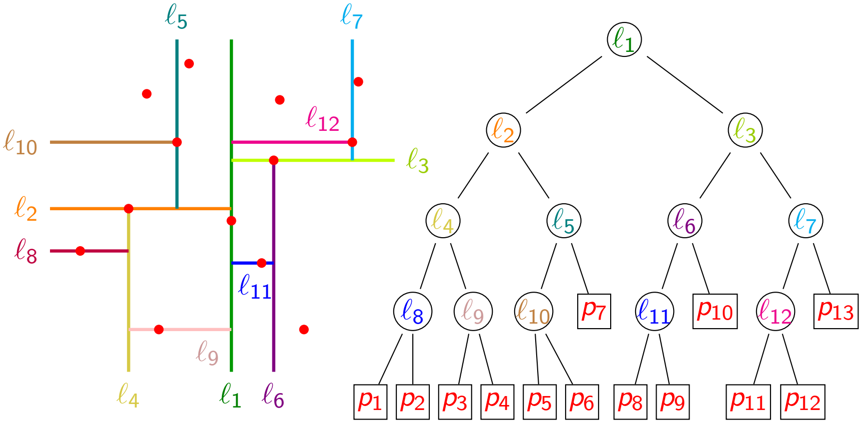
Árvore de busca binária que divide os pontos alternadamente, ora por uma linha vertical, ora por uma linha horizontal.





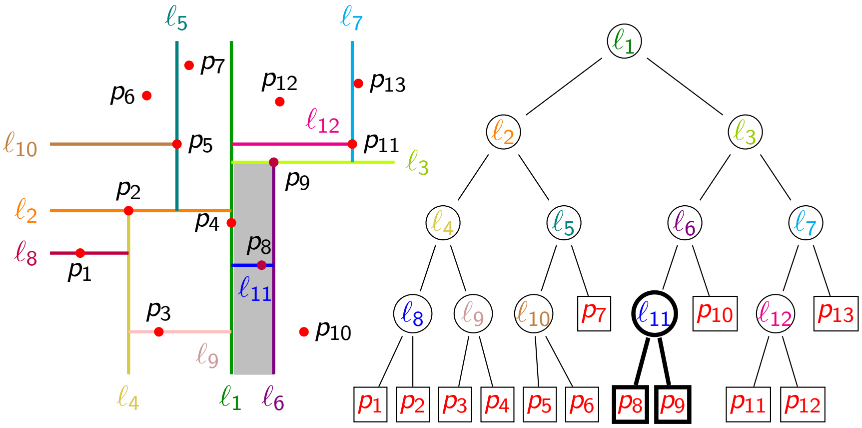
KD-trees

Árvore de busca binária que divide os pontos alternadamente, ora por uma linha vertical, ora por uma linha horizontal.



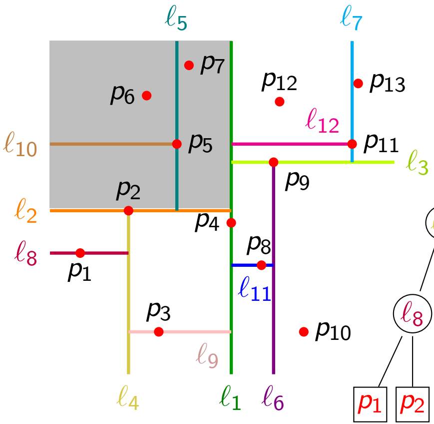
KD-trees

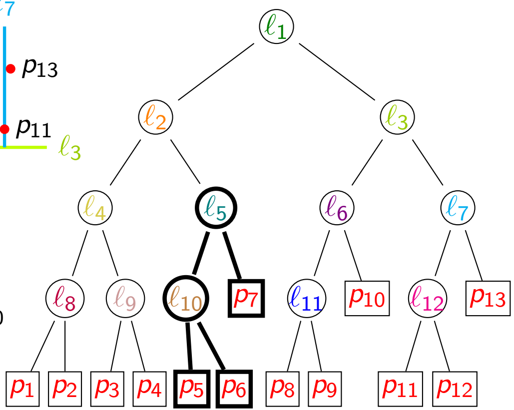
Subárvores e suas regiões



KD-trees

Subárvores e suas regiões





Construção de uma KD-tree

```
BuildKDTree(P, depth)
se |P| = 1
    então cria e retorna uma folha com o único ponto em P
se depth é par
    então divida P em dois conjuntos por uma reta vertical ℓ
          passando pela mediana das x-coordenadas dos pontos de P;
          seja P_1 o conjunto dos pontos de P à esquerda ou
          em cima de ℓ e P_2 os pontos de P à direita de ℓ.
    senão        ▷ análogo para uma reta horizontal ℓ
v_e ← BuildKDTree(P_1, depth + 1)
v_d ← BuildKDTree(P_2, depth + 1)
cria nó v com conteúdo ℓ e filhos v_e e v_d
devolva v
```

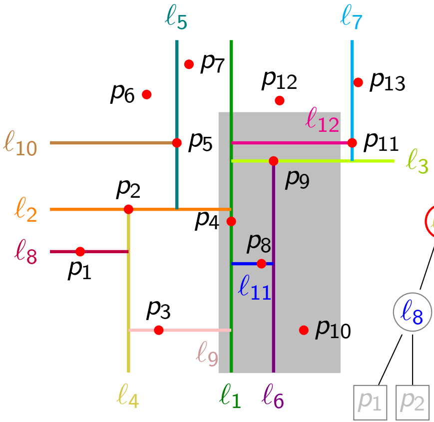
Construção de uma KD-tree

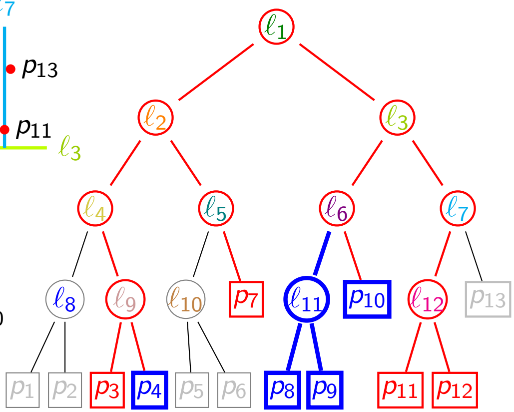
BuildKDTree(P, *depth*)
se $|P| = 1$
    então cria e retorna uma folha com o único ponto em P
se *depth* é par
    então divida P em dois conjuntos por uma reta vertical ℓ
        passando pela mediana das x-coordenadas dos pontos de P;
        seja P_1 o conjunto dos pontos de P à esquerda ou
        em cima de ℓ e P_2 os pontos de P à direita de ℓ.
    senão ▷ análogo para uma reta horizontal ℓ
$v_e \leftarrow$ BuildKDTree(P_1, $depth + 1$)
$v_d \leftarrow$ BuildKDTree(P_2, $depth + 1$)
cria nó v com conteúdo ℓ e filhos v_e e v_d
devolva v

Consumo de tempo: $\mathrm{O}(n \lg n)$, onde $n := |P|$.

Consumo de espaço: $\mathrm{O}(n)$.

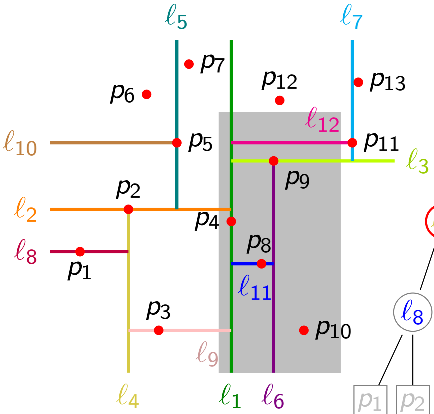
Consultas ortogonais em KD-trees

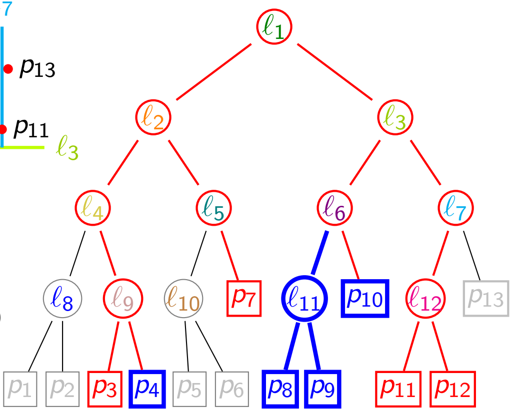




Se R está totalmente à esquerda de ℓ, vá para a esquerda.

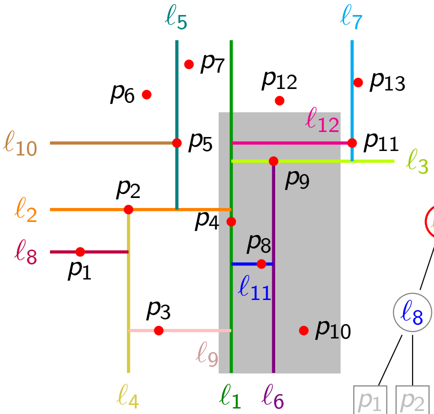
Consultas ortogonais em KD-trees

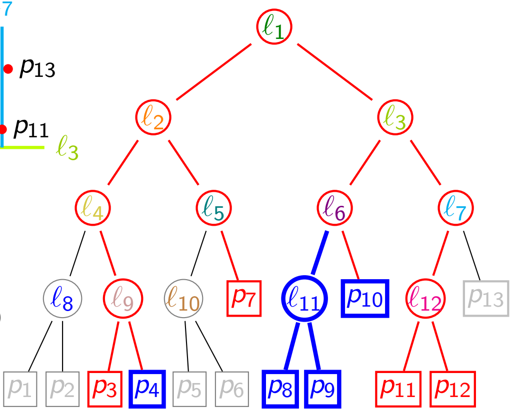




Se R está totalmente à esquerda de ℓ, vá para a esquerda.
Se R está totalmente à direita de ℓ, vá para a direita.

Consultas ortogonais em KD-trees





Se R está totalmente à esquerda de ℓ, vá para a esquerda.
Se R está totalmente à direita de ℓ, vá para a direita.
Se ℓ intersecta R, vá para os dois lados.

Consultas ortogonais em KD-trees

```
SearchKDTree(v, R)
se v é folha
    então imprima o ponto guardado em v se ele pertence a R
    senão se region(esq(v)) ⊆ R
            então ReportSubtree(esq(v))
            senão se region(esq(v)) intersecta R
                    então SearchKDTree(esq(v), R)
        se region(dir(v)) ⊆ R
            então ReportSubtree(dir(v))
            senão se region(dir(v)) intersecta R
                  então SearchKDTree(dir(v), R)
```

Consultas ortogonais em KD-trees

```
SearchKDTree(v, R)
se v é folha
    então imprima o ponto guardado em v se ele pertence a R
    senão se region(esq(v)) ⊆ R
            então ReportSubtree(esq(v))
            senão se region(esq(v)) intersecta R
                    então SearchKDTree(esq(v), R)
        se region(dir(v)) ⊆ R
            então ReportSubtree(dir(v))
            senão se region(dir(v)) intersecta R
                then SearchKDTree(dir(v), R)
```

Pré-processe a árvore calculando *region*(v) para todo v.

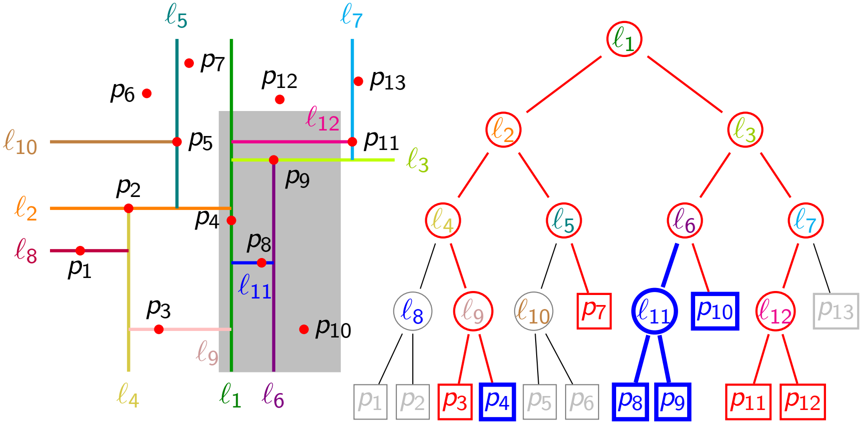
Consultas ortogonais em KD-trees

```
SearchKDTree(v, R)
se v é folha
    então imprima o ponto guardado em v se ele pertence a R
    senão se region(esq(v)) ⊆ R
            então ReportSubtree(esq(v))
            senão se region(esq(v)) intersecta R
                    então SearchKDTree(esq(v), R)
        se region(dir(v)) ⊆ R
            então ReportSubtree(dir(v))
            senão se region(dir(v)) intersecta R
                    então SearchKDTree(dir(v), R)
```

Pré-processe a árvore calculando *region*(v) para todo v.

Consumo de tempo: $\mathrm{O}(\sqrt{n}+k)$, onde n é o número de pontos e k o número de pontos reportados na consulta.

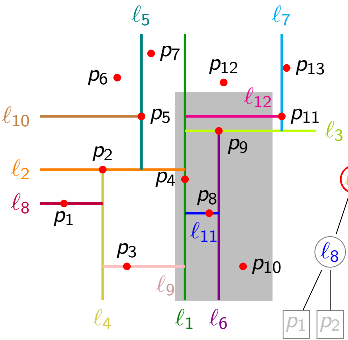
Custo de uma consulta: $\mathrm{O}(\sqrt{n} + k)$

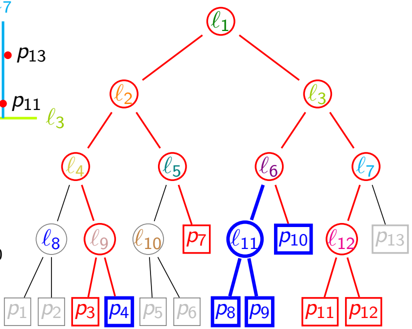


O termo k corresponde às subárvores azuis,
de tamanho proporcional às folhas impressas.

O termo $\mathrm{O}(\sqrt{n})$ é pelas partes vermelhas percorridas no processo.

Custo de uma consulta: $\mathrm{O}(\sqrt{n}+k)$





O termo $\mathrm{O}(\sqrt{n})$ é pelas partes vermelhas percorridas no processo.

R intersecta mas não contém a região de cada nó vermelho.

Isso significa que a borda de R intersecta a região do nó.

Número de nós vermelhos

Considere uma KD-tree com n folhas e uma uma reta vertical ℓ.

A reta ℓ pode intersectar a região de quantos nós?
Digamos que $Q(n)$.

Número de nós vermelhos

Considere uma KD-tree com n folhas e uma uma reta vertical ℓ.

A reta ℓ pode intersectar a região de quantos nós?
Digamos que $Q(n)$.

ℓ ou está à esquerda da linha vertical da raiz da árvore, ou à direita.

Será que $Q(n) = 1 + Q(n/2)$?

Número de nós vermelhos

Considere uma KD-tree com n folhas e uma uma reta vertical ℓ.

A reta ℓ pode intersectar a região de quantos nós?
Digamos que $Q(n)$.

ℓ ou está à esquerda da linha vertical da raiz da árvore, ou à direita.

Será que $Q(n) = 1 + Q(n/2)$?

Não... na verdade $Q(n) = 2 + 2Q(n/4)$,
porque alternamos retas verticais e horizontais nos nós da árvore.

A reta vertical intersecta as regiões dos dois netos
que são filhos do lado em que ela está em relação à reta da raiz.
(A recursão deve aplicar-se apenas a nós cuja reta é vertical.)

Número de nós vermelhos

Considere uma KD-tree com n folhas e uma uma reta vertical ℓ.

A reta ℓ pode intersectar a região de quantos nós?
Digamos que $Q(n)$.

ℓ ou está à esquerda da linha vertical da raiz da árvore, ou à direita.

Será que $Q(n) = 1 + Q(n/2)$?

Não... na verdade $Q(n) = 2 + 2Q(n/4)$,
porque alternamos retas verticais e horizontais nos nós da árvore.

A reta vertical intersecta as regiões dos dois netos
que são filhos do lado em que ela está em relação à reta da raiz.
(A recursão deve aplicar-se apenas a nós cuja reta é vertical.)

A solução desta recorrência é $Q(n) = \mathrm{O}(\sqrt{n})$.

Resumo

P: coleção de n pontos

Objetivo: responder consultas retangulares

Kd-trees:

Consumo de espaço: $\mathrm{O}(n)$

Custo por consulta: $\mathrm{O}(\sqrt{n} + k)$.

Resumo

P: coleção de n pontos

Objetivo: responder consultas retangulares

Kd-trees:

Consumo de espaço: $\mathrm{O}(n)$

Custo por consulta: $\mathrm{O}(\sqrt{n} + k)$.

Range trees:

Consumo de espaço: $\mathrm{O}(n \lg n)$

Custo por consulta: $\mathrm{O}(\lg^2 n + k)$.

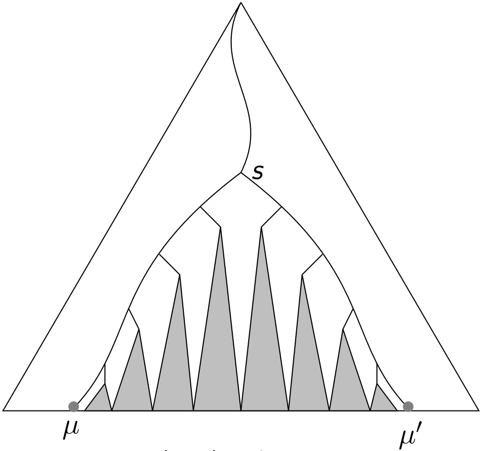
Range trees

Consulta retangular: quais pontos de P estão em $[x : x'] \times [y : y']$?

Range trees

Consulta retangular: quais pontos de P estão em $[x : x'] \times [y : y']$?

Ideia: primeiro buscar pontos cuja x-coordenada está em $[x : x']$.

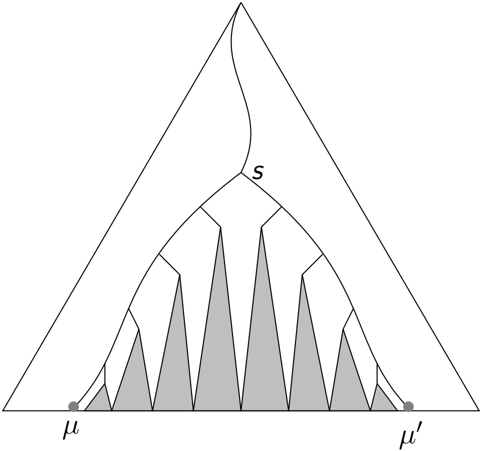


$\mathrm{O}(\lg n)$ subárvores

Range trees

Consulta retangular: quais pontos de P estão em $[x : x'] \times [y : y']$?

Ideia: primeiro buscar pontos cuja x-coordenada está em $[x : x']$.



Buscar nestas subárvores pontos com y-coordenada em $[y : y']$.

Range trees: estrutura de dados com níveis

Como fazer estas segundas buscas eficientemente?

Range trees: estrutura de dados com níveis

Como fazer estas segundas buscas eficientemente?

Cada vértice v da ABB$_x$ tem um conjunto canônico P_v,
que consiste dos pontos que aparecem na subárvore enraizada em v.

Range trees: estrutura de dados com níveis

Como fazer estas segundas buscas eficientemente?

Cada vértice v da ABB_x tem um conjunto canônico P_v, que consiste dos pontos que aparecem na subárvore enraizada em v.

O conjunto dos pontos de P com x-coordenada em $[x : x']$ é a união disjunta de $\mathrm{O}(\lg n)$ conjuntos canônicos.

Range trees: estrutura de dados com níveis

Como fazer estas segundas buscas eficientemente?

Cada vértice v da ABB_x tem um conjunto canônico P_v,
que consiste dos pontos que aparecem na subárvore enraizada em v.

O conjunto dos pontos de P com x-coordenada em $[x : x']$
é a união disjunta de $\mathrm{O}(\lg n)$ conjuntos canônicos.

Manteremos, para cada v, o conjunto P_v
em uma ABB secundária, ordenada pela y-coordenada dos pontos.

Range trees: estrutura de dados com níveis

Como fazer estas segundas buscas eficientemente?

Cada vértice v da ABB_x tem um conjunto canônico P_v,
que consiste dos pontos que aparecem na subárvore enraizada em v.

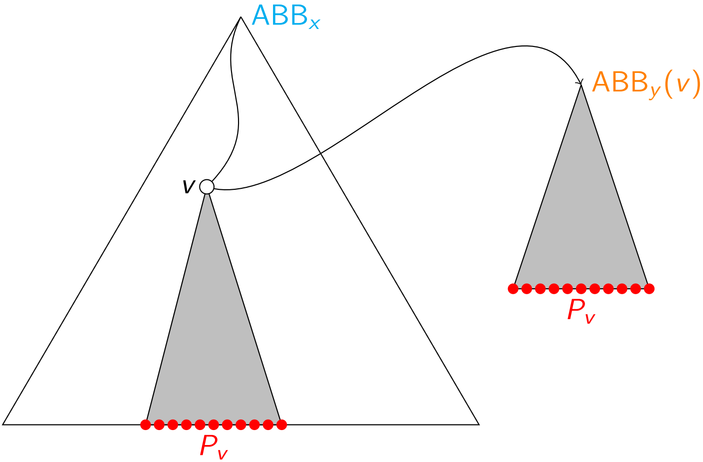
O conjunto dos pontos de P com x-coordenada em $[x : x']$
é a união disjunta de $\mathrm{O}(\lg n)$ conjuntos canônicos.

Manteremos, para cada v, o conjunto P_v
em uma ABB secundária, ordenada pela y-coordenada dos pontos.

Chamemos essa ABB de $\mathrm{ABB}_y(v)$ para cada v.

$\mathrm{ABB}_y(v)$: árvore para busca unidimensional por intervalo.

Range trees: estrutura de dados com níveis



Construção de uma range tree 2D

BuildRangeTree(P)

Construa uma ABB_y T para P em relação a y

se $|P| = 1$

    então cria uma folha v para este ponto

    senão calcule x_{meio} para P

        divida P em P_{esq} e P_{dir} em relação a x_{meio}

        $v_{esq} \leftarrow$ BuildRangeTree(P_{esq})

        $v_{dir} \leftarrow$ BuildRangeTree(P_{dir})

        cria nó v com chave x_{meio} e filhos v_{esq} e v_{dir}

$\text{ABB}_y(v) \leftarrow T$

devolva v

Construção de uma range tree 2D

BuildRangeTree(P)
Construa uma ABB_y T para P em relação a y
se $|P| = 1$
    então cria uma folha v para este ponto
    senão calcule x_{meio} para P
        divida P em P_{esq} e P_{dir} em relação a x_{meio}
        $v_{esq} \leftarrow$ BuildRangeTree(P_{esq})
        $v_{dir} \leftarrow$ BuildRangeTree(P_{dir})
        cria nó v com chave x_{meio} e filhos v_{esq} e v_{dir}
$\text{ABB}_y(v) \leftarrow T$
devolva v

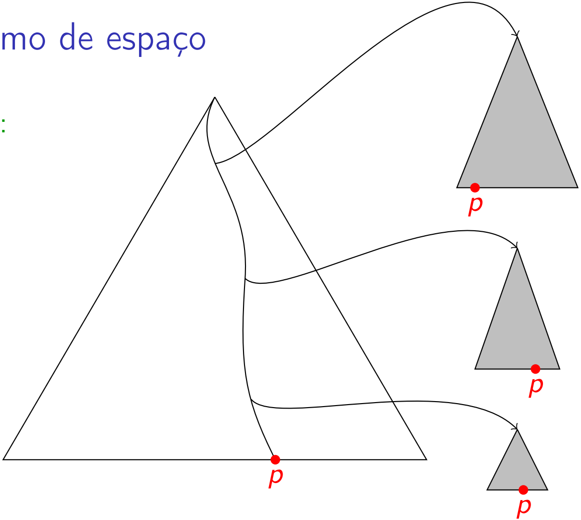
Consumo de tempo:

Se pré-ordenarmos por y e garantirmos que as chamadas a BuildRangeTree sempre recebem os pontos ordenados por y, o consumo de tempo será $\mathrm{O}(n \lg n)$, onde $n := |P|$, pois a construção de cada ABB_y custará tempo linear neste caso.

Range trees: consumo de espaço

Consumo de espaço:

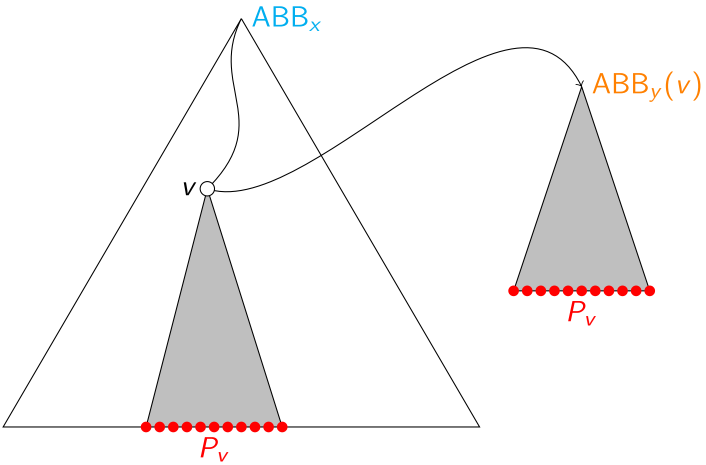
Pela ABB_x: $\mathrm{O}(n)$



Cada ponto aparece em $\text{ABB}_y(v)$ para $\mathrm{O}(\lg n)$ vértices v da ABB_x.
No total então $\mathrm{O}(n \lg n)$ pelas $\text{ABB}_y(v)$ para todo v da ABB_x.

Consumo total de espaço: $\mathrm{O}(n \lg n)$

Range trees: consulta



Consulta: busca na ABB_x e, para cada $\mathrm{ABB}_y(v)$ encontrada na consulta por x, busca na $\mathrm{ABB}_y(v)$ por y.

Range trees: consulta

Query2DRangeTree($T, [x, x'] \times [y, y']$)

$s \leftarrow$ FindSplitNode(T, x, x')
se s é folha
    então imprime o ponto em s se está em $[x, x'] \times [y, y']$
    senão ▷ siga o caminho até x e processe subárvores à direita
        $v \leftarrow esq(s)$
        enquanto v não é folha faça
            se $x \leq x_v$
                então Query1DRangeTree($T^y(v), [y, y']$)
                    $v \leftarrow esq(v)$
                senão $v \leftarrow dir(v)$
        ▷ siga o caminho até x' e processe subárvores à esquerda
        ...

Range trees: consulta

Query2DRangeTree(T, $[x, x'] \times [y, y']$)

$s \leftarrow$ FindSplitNode(T, x, x')
se s é folha
    então imprime o ponto em s se está em $[x, x'] \times [y, y']$
    senão $\triangleright$ siga o caminho até x e processe subárvores à direita
        $v \leftarrow esq(s)$
        enquanto v não é folha faça
            se $x \leq x_v$
                então Query1DRangeTree($T^y(v)$, $[y, y']$)
                    $v \leftarrow esq(v)$
                senão $v \leftarrow dir(v)$
        $\triangleright$ siga o caminho até x' e processe subárvores à esquerda
        ...

Consumo de tempo por consulta: $\mathrm{O}(\lg^2 n + k)$
$\mathrm{O}(\lg n)$ pela busca unidimensional em T por $[x : x']$ e
$\mathrm{O}(\lg n)$ buscas unidimensionais por $[y : y']$, custo $\mathrm{O}(\lg n + k')$ cada.

Range trees: análise

Consumo de tempo por consulta:

$\mathrm{O}(\lg n)$ pela busca unidimensional na ABB_x por $[x : x']$ e

$\mathrm{O}(\lg n)$ buscas unidimensionais por $[y : y']$,
uma em cada $\mathrm{ABB}_y(v)$ encontrada na busca acima.

Tempo total por consulta: $\mathrm{O}(\lg^2 n + k)$

Range trees: análise

Consumo de tempo por consulta:

$\mathrm{O}(\lg n)$ pela busca unidimensional na ABB_x por $[x : x']$ e

$\mathrm{O}(\lg n)$ buscas unidimensionais por $[y : y']$,
uma em cada $\mathrm{ABB}_y(v)$ encontrada na busca acima.

Tempo total por consulta: $\mathrm{O}(\lg^2 n + k)$

Consumo de espaço:

Pela ABB_x: $\mathrm{O}(n)$

Cada ponto aparece em $\mathrm{ABB}_y(v)$ para $\mathrm{O}(\lg n)$ vértices v da ABB_x.
No total então $\mathrm{O}(n \lg n)$ pelas $\mathrm{ABB}_y(v)$ para todo v da ABB_x.

Consumo total de espaço: $\mathrm{O}(n \lg n)$

Hipótese simplificadora

Como garantir que todas as x-coordenadas são distintas
e todas as y-coordenadas são distintas?

Hipótese simplificadora

Como garantir que todas as x-coordenadas são distintas
e todas as y-coordenadas são distintas?

Seja $p = (x, y)$.

Considere que p é na verdade $((x, y), (y, x))$.

Defina $(a, b) < (c, d)$ sse $a < b$ ou $a = b$ e $c < d$.

Hipótese simplificadora

Como garantir que todas as x-coordenadas são distintas
e todas as y-coordenadas são distintas?

Seja $p = (x, y)$.

Considere que p é na verdade $((x, y), (y, x))$.

Defina $(a, b) < (c, d)$ sse $a < b$ ou $a = b$ e $c < d$.

Ou seja, dados dois pontos $p = (x, y)$ e $q = (w, z)$,

$p_x < q_x$ sse $x < w$ ou $x = w$ e $y < z$

$p_y < q_y$ sse $y < z$ ou $y = z$ e $x < w$