DCC207 – Algoritmos 2

Aula 06 – Introdução a Geometria Computacional (Parte 03)

Professor Renato Vimieiro

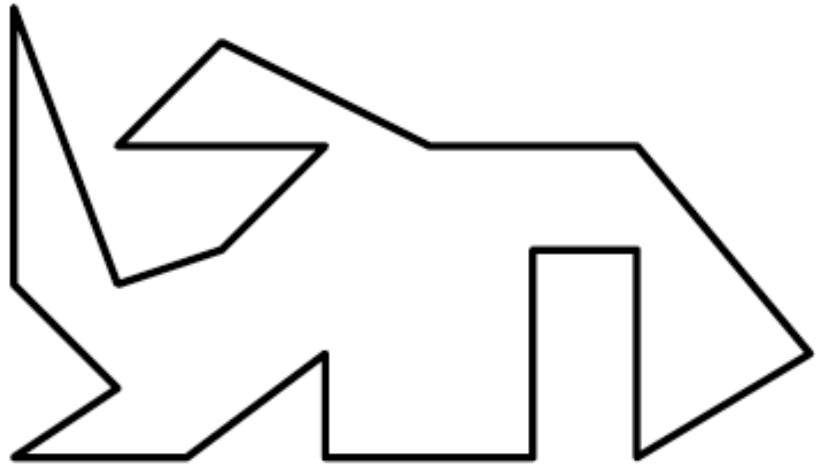
DCC/ICEx/UFMG

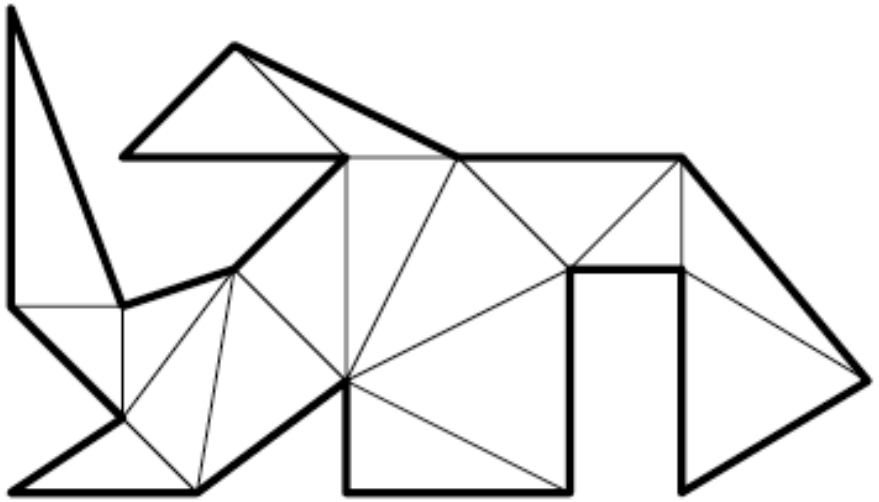
Problema da galeria de arte

- Considere o problema de instalar câmeras de segurança em uma galeria de arte para vigiar todos os cômodos da propriedade
- Deseja-se instalar o menor número de câmeras possível, respeitando-se a condição de que todos os cômodos sejam cobertos
- Embora a galeria seja tridimensional, podemos modelá-la como um polígono no plano
  - As câmeras podem ser instaladas no teto e cobrem toda a área de um cômodo, e ainda alguma parte de outros, exceto pelo bloqueio causado pelas paredes
- Naturalmente, polígonos mais complexos, com mais vértices requerem mais câmeras
  - Mesmo com um mesmo número de vértices, um polígono pode ser mais difícil de vigiar que outro: um polígono convexo requer somente uma câmera
- O problema de determinar o número ótimo de câmeras é computacionalmente difícil
  - Investigaremos limiares para os piores casos, assim a solução é adequada para qualquer problema, mesmo que não seja ótima em alguns casos

Triangulação de polígonos

- Polígonos triviais, como triângulos, podem ser resolvidos com apenas uma câmera
- Portanto, podemos resolver o problema dividindo o polígono em triângulos, e posicionando uma câmera em cada um deles
- Essa tarefa de dividir um polígono em triângulos é chamada de triangulação
- Formalmente, o problema consiste em decompor um polígono em triângulos, usando um conjunto máximo de **diagonais** disjuntas (que não se interceptam)
- Uma diagonal é um segmento conectando dois vértices do polígono que se encontra estritamente dentro do polígono





Triangulação de polígonos

Teorema: Todo polígono simples possui pelo menos uma triangulação, e toda triangulação de um polígono simples com n vértices consiste em exatamente n-2 triângulos.

Prova: Por indução em n. Para n=3, o teorema é trivialmente válido. Para n > 3, suponha como HI que o teorema seja válido para todo m < n. Seja P um polígono com n vértices.

Primeiramente demonstramos que sempre existe uma diagonal. Considere o vértice v mais à esquerda no polígono e seus vizinhos u e w. Se o segmento uw estiver integralmente dentro do polígono, então existe tal diagonal. Caso contrário, existe pelo menos um vértice dentro do triângulo formado por uvw. Seja v' o vértice interno mais distante do segmento uw. Como v' é o mais distante, a aresta vv' não intercepta uma aresta do polígono (senão existiria um vértice mais distante que v'). Portanto, vv' é uma diagonal.

Mostrada a existência dessa diagonal, ela divide o polígono em dois P1 e P2. Como o número de vértices nesses polígonos é menor que n, segue dá hipótese que podemos triangulá-los. Consequentemente, P também pode ser triangulado.

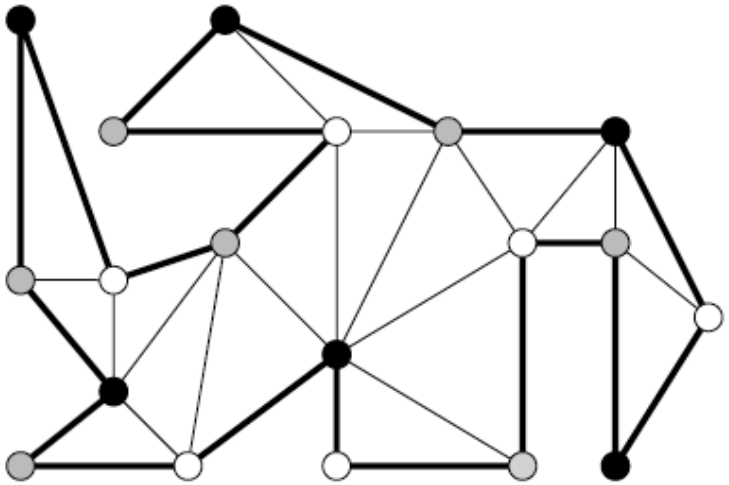
Além disso, cada vértice do polígono pertencerá somente a P1 ou P2, exceto os terminais da diagonal que separam P. Sendo assim, |P1| + |P2| = n+2. Pela hipótese, cada Pi tem |Pi|-2 triângulos. Logo, o número de triângulos em P será |P1|-2 + |P2|-2 = n-2.

Triangulação de polígonos

- Um corolário do teorema anterior é que toda galeria de arte pode ser completamente vigiada por n-2 câmeras
- Esse número, no entanto, parece bastante elevado
- Se posicionarmos a câmera em uma diagonal, conseguimos cobrir dois triângulos ao mesmo tempo. Isso reduz o número de câmeras pela metade
- Podemos reduzir ainda mais esse número se posicionarmos as câmeras em vértices dos triângulos
  - Temos que escolher um subconjunto de vértices de maneira a cobrir todos os triângulos

Triangulação de polígonos

- A escolha será feita com base na coloração dos vértices
- Seja P um polígono e T(P) sua triangulação. Considere o grafo G(P) dual de T(P) em que os vértices são os triângulos e as arestas se eles compartilham uma diagonal
- Como as diagonais dividem o polígono em dois, a remoção de qualquer aresta de G(P) desconecta o grafo. Portanto, G(P) é uma árvore
- Podemos colorir os vértices de T(P), fazendo uma busca em profundidade em G(P)
- Iniciamos a busca em um vértice arbitrário de G(P). Nesse momento, colorimos cada vértice do triângulo equivalente com uma cor (branco, preto e cinza)
- Quando visitarmos o próximo vértice de G(P), teremos apenas uma escolha para a cor do vértice restante do triângulo equivalente, pois os outros dois são compartilhados com o triângulo anterior.
- Ao terminarmos, teremos uma 3-coloração de T(P). Escolhemos a partição com o menor número de vértices para posicionar as câmeras. Essa escolha resulta em um total de n/3 câmeras

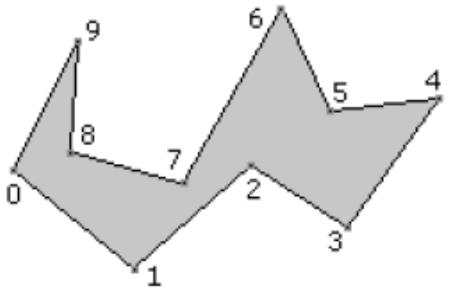


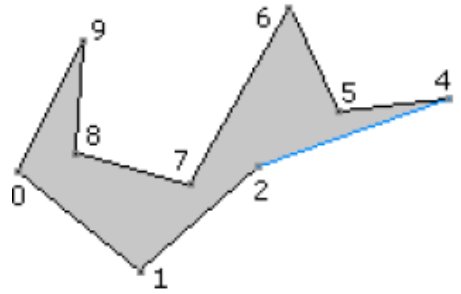
Triangulação de polígonos

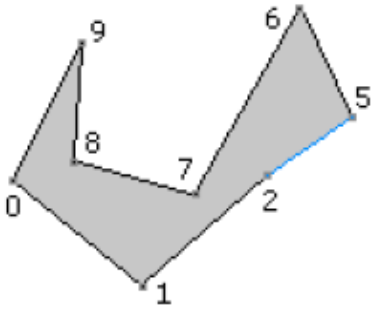
- O problema agora se resume a encontrar uma triangulação do polígono
- A implementação que veremos requer tempo quadrático
  - Existem outras implementações mais elaboradas que rodam em tempo O(n lg n)
- Veremos o algoritmo conhecido como Ear-Clipping (corte de orelhas)
- Uma orelha de um polígono é o triângulo formado pelos vértices consecutivos u, v, w, tal que uw é uma diagonal
  - Nesse caso, v é chamado de ponta da orelha

Triangulação de polígonos

- O algoritmo inicia verificando quais vértices formam orelhas (quais são pontas de orelhas)
  - P_{i-1}-P_i-P_{i+1} gira à esquerda, e o triângulo não contém outro vértice
  - Esse processo tem custo O(n) para cada vértice
- O próximo passo é: Enquanto o número de vértices for maior que três:
  - Remover uma ponta de orelha
  - Atualizar o status dos vértices adjacentes; eles podem ter se tornado pontas de orelhas, ou deixado de ser







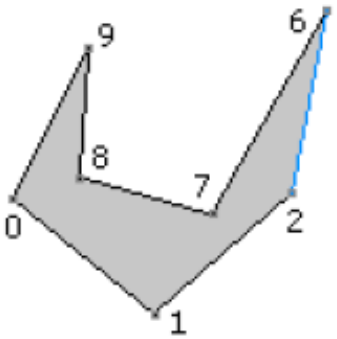


Diagrama de Voronoi

- Considere o seguinte problema: você é o diretor de uma cadeia de supermercados e deseja abrir uma nova loja. Considerando que, por conveniência, as pessoas sempre se dirigem ao supermercado mais perto (às vezes até ignorando diferenças de preço), como estimar o lucro dessa nova loja?
- Esse problema envolve particionar a região considerada (bairro, cidade, etc.) em zonas e estimar o número de potenciais clientes
- Especificamente, queremos dividir a região conforme sítios de interesse (localização das lojas) de tal forma que todo ponto dentro da zona de um sítio esteja mais próximo dele do que qualquer outro

Diagrama de Voronoi

- Esse problema é modelado por uma estrutura matemática chamada diagrama de Voronoi
- Considere que os sítios de interesse sejam pontos no plano. Logo, temos um conjunto P de n pontos
- O diagrama de Voronoi divide o plano em n células, uma para cada ponto, com a propriedade que:
  - Um ponto q está localizado na célula de um ponto p_i sse $dist(q,p_i) < dist(q,p_j)$, para $j \neq i$.
  - A célula do ponto pi é definida por $V(p_i) = \{q \mid dist(q,p_i) < dist(q,p_j) \;\; j \neq i\}$
- Em geral, o diagrama é representado somente por seus vértices e arestas (segmentos e semi-retas)
  - Ele é denotado por Vor(P)

Diagrama de Voronoi

- Para entender melhor a estrutura, considere a situação em que temos somente dois sítios (pontos) de interesse p e q
- A reta perpendicular ao segmento pq que passa por seu ponto médio
- Essa reta divide o plano em 2 semi-planos
  - Denotamos o semi-plano aberto que contém p como h(p,q)
  - E o semi-plano aberto que contém q como h(q,p)
- Logo, um ponto r ∈ h(p,q) sse dist(r,p) < dist(r,q)
- Dessa forma, $V(p_i) = \bigcap_{1 \leq j \leq n} h(p_i,p_j)$

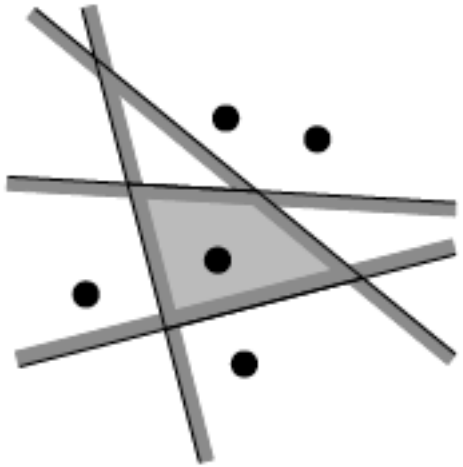


Diagrama de Voronoi

- Uma célula é a interseção de n-1 semi-planos
- Ela é uma região poligonal limitada por no máximo n-1 vértices e n-1 arestas
  - A região pode ser aberta em algum dos lados
  - As arestas podem ser segmentos ou semi-retas
  - No caso degenerado em que os pontos são colineares, arestas são retas paralelas
- Embora cada célula seja limitada por n-1 vértices e arestas, o número total de vértices é, no máximo, 2n-4 e o número de arestas, no máximo, 3n-6
  - Segue da fórmula de Euler para grafos planares: f=e-v+2
  - Demonstração O'Rourke pg. 161

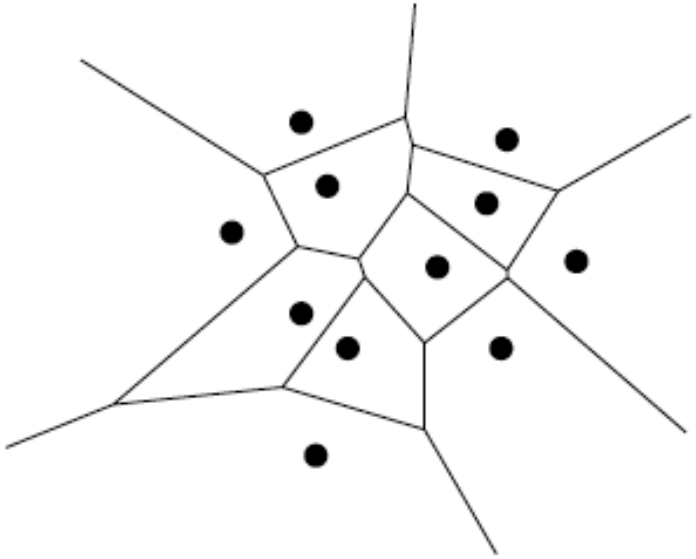


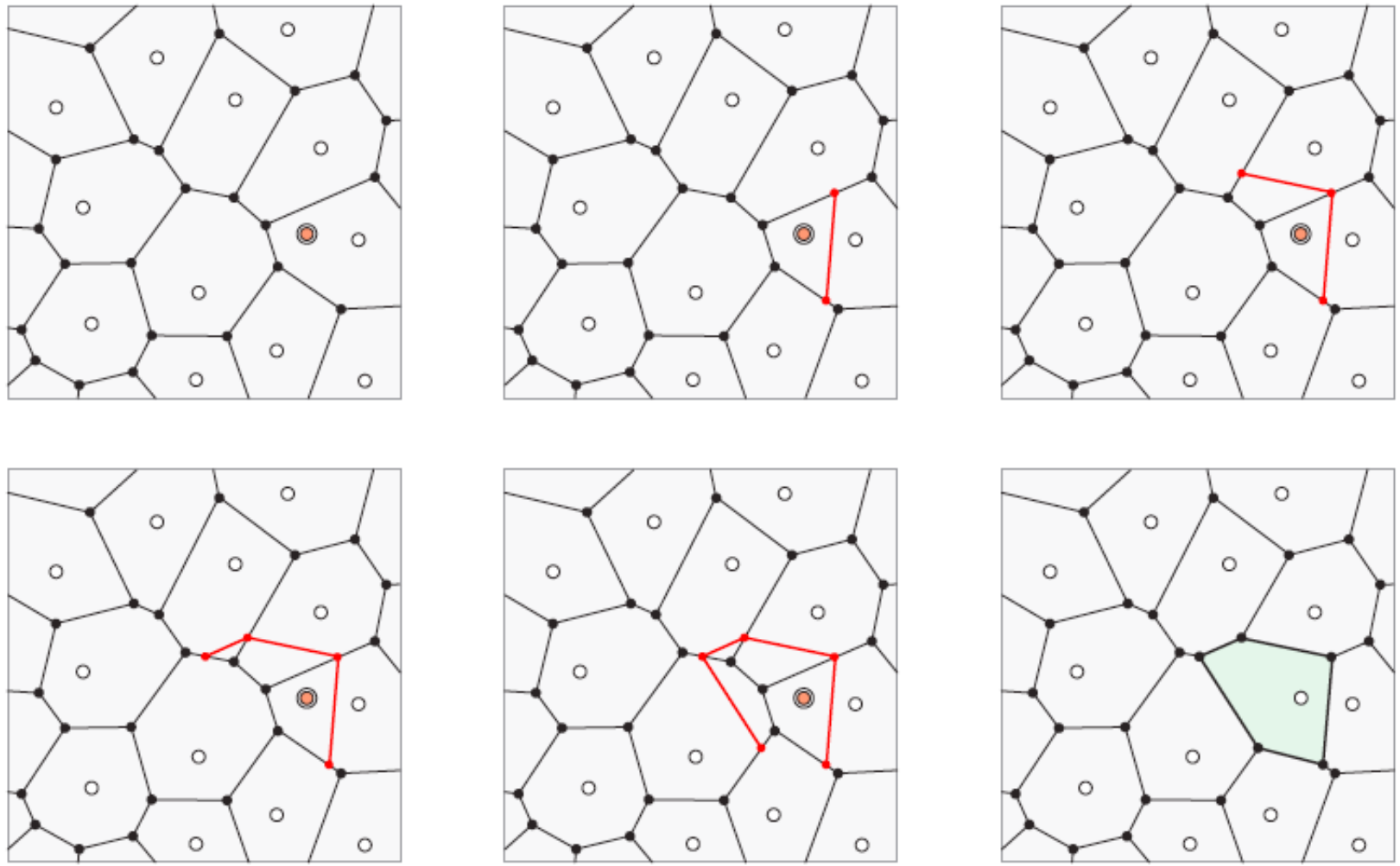
Diagrama de Voronoi

- A definição da célula de Voronoi anterior induz um algoritmo ingênuo para a construção do diagrama
- Para cada sítio de interesse:
  - Compute a interseção dos n-1 semi-planos h(pi,pj)
  - Compute a envoltória convexa dos pontos, representando a região
- Una as regiões resultantes
- Esse algoritmo tem custo $O(n^2 \lg n)$ e sofre com imprecisões numéricas
  - Um vértice compartilhado por duas regiões pode ser levemente deslocado ao ser computado em cada região

Diagrama de Voronoi

- Outra abordagem é usar um algoritmo incremental
- Suponha que Vor(P_{i-1}) seja o diagrama de Voronoi para os i-1 primeiros sítios de interesse
- Como modificar Vor(P_{i-1}) para acomodar o novo sítio p_i?
  - Identificar a região $V(p_j)$ em que p_i se situa
  - A reta que passa sobre o segmento p_ip_j (dividindo a região em duas) intercepta a fronteira de $V(p_j)$ em dois pontos x_1 e x_2 (suponha que x_2 esteja no sentido anti-horário de x_1)
  - O ponto x_2 está na fronteira das regiões $V(p_j)$ e $V(p_k)$. Como a reta que separa p_k e p_i intercepta a fronteira de $V(p_k)$ em dois pontos, sabemos que ela intercepta o ponto x_2, e assim, podemos calcular o outro ponto x_3
  - Repetimos o procedimento acima girando no sentido anti-horário até retornarmos a x_1. Nesse momento, temos a fronteira da região $V(p_i)$. Resta apenas eliminar os vértices do diagrama internos a essa região.

Diagrama de Voronoi



Busca intervalar (Range Search)

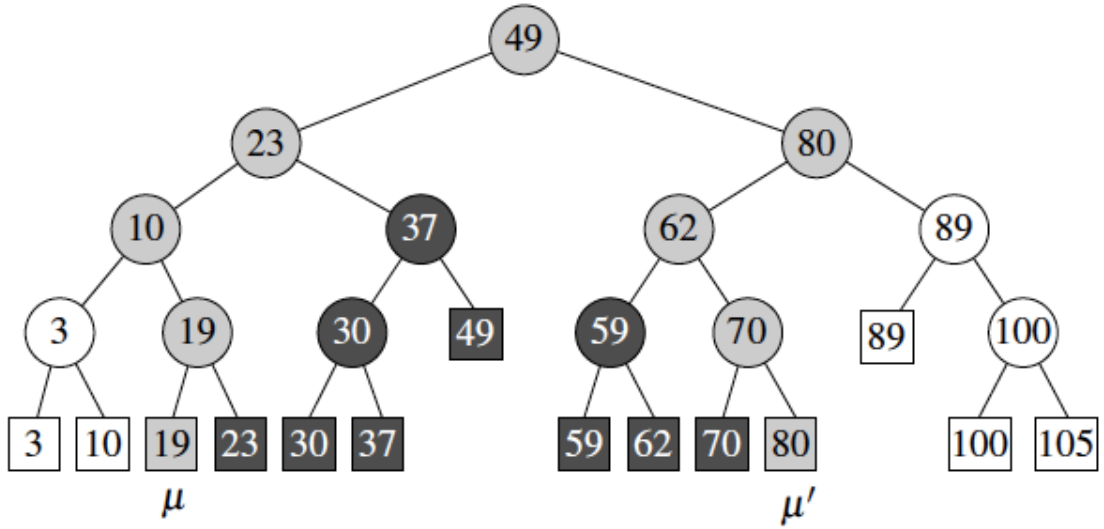
- Considere a seguinte consulta a um banco de dados: retorne todas as entradas cuja variável salário esteja no intervalo [5000;20000] e idade [21;30].
- Em princípio, essa consulta não tem nada a ver com geometria. Porém, podemos interpretar as transações do BD como pontos no espaço n-dimensional (o que é frequentemente feito em várias áreas da computação, e.g. aprendizado de máquina)
- Dessa forma, o problema acima se resume a encontrar um conjunto de pontos no banco de dados que se encontram dentro do retângulo [5000;20000] x [21;30]
- Esse problema é conhecido por busca intervalar (range search) e pode ser resolvido de forma eficiente com o auxílio de estruturas de dados espaciais

Busca Intervalar (Range Search)

- Vamos considerar primeiro o problema de encontrar pontos no espaço 1D, depois generalizaremos para mais dimensões
- Nesse caso, temos um conjunto P com n pontos, e o intervalo (fechado) de consulta [xmin;xmax].
- Esse problema pode ser facilmente resolvido usando uma árvore binária balanceada
- Podemos usar qualquer árvore balanceada, mas assumiremos que somente as folhas armazenam pontos
  - Os nós internos terão valores para guiar a busca
  - Cada nó interno v, divide o conjunto de pontos em xi <= xv (esq), e xi > xv (dir)

Busca Intervalar (Range Search)

- Assim, tendo uma árvore T, construída como descrito anteriormente, podemos realizar consultas da seguinte forma
- Buscamos por xmin e xmax individualmente em T. Isso nos leva a duas folhas μ e μ'. O conjunto solução são todas as folhas entre μ e μ' (potencialmente as duas inclusas).
- Exemplo [18;77]



Busca Intervalar (Range Search)

- O problema agora é como encontrar as folhas que satisfazem a consulta
- O primeiro passo é encontrar o nó (v_{split}) em que a busca se parte em esquerda e direita. Isso é, encontrar a raiz da sub-árvore que possivelmente contenha o intervalo
- O algoritmo para executar esse procedimento é simples:
  - Inicie da raiz
  - Enquanto o intervalo estiver à esquerda/direita do valor do nó atual, desça para a sub-árvore adequada

Busca Intervalar (Range Search)

FINDSPLITNODE($\mathcal{T}$,x,x')
Input. A tree $\mathcal{T}$ and two values x and x' with $x \leqslant x'$.
Output. The node v where the paths to x and x' split, or the leaf where both paths end.

1. $v \leftarrow root(\mathcal{T})$
2. **while** v is not a leaf **and** ($x' \leqslant x_v$ **or** $x > x_v$)
3. **do if** $x' \leqslant x_v$
4. **then** $v \leftarrow lc(v)$
5. **else** $v \leftarrow rc(v)$
6. **return** v

Busca Intervalar (Range Search)

- Tendo encontrado o nó de partida, os nós que devem ser reportados podem ser encontrados nas sub-árvores da esquerda e direita
- Todas folhas das sub-árvores da direita no percurso até encontrar xmin na sub-árvore da esquerda devem ser reportadas
- O inverso se aplica ao percurso na sub-árvore da direita para encontrar xmax
- Finalmente, as folhas que deveriam conter xmin e xmax devem ser avaliadas para verificar se satisfazem a consulta

Busca Intervalar (Range Search)

Algorithm 1DRANGEQUERY($\mathcal{T}, [x : x']$)
Input. A binary search tree $\mathcal{T}$ and a range $[x : x']$.
Output. All points stored in $\mathcal{T}$ that lie in the range.

```
1.  v_split ← FINDSPLITNODE(T, x, x')
2.  if v_split is a leaf
3.     then Check if the point stored at v_split must be reported.
4.     else (* Follow the path to x and report the points in subtrees right of the
            path. *)
5.          v ← lc(v_split)
6.          while v is not a leaf
7.             do if x ⩽ x_v
8.                   then REPORTSUBTREE(rc(v))
9.                        v ← lc(v)
10.                  else  v ← rc(v)
11.         Check if the point stored at the leaf v must be reported.
12.         Similarly, follow the path to x', report the points in subtrees left of
            the path, and check if the point stored at the leaf where the path
            ends must be reported.
```

Busca Intervalar (Range Search)

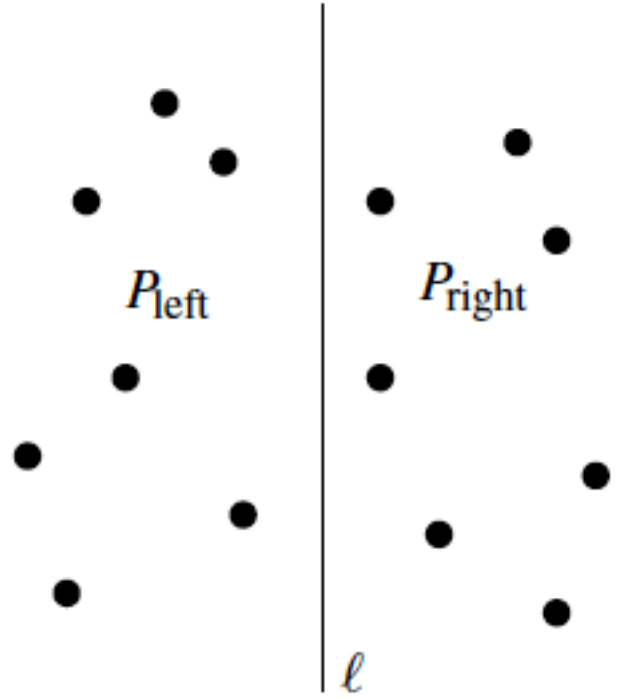
- O custo de armazenamento dos pontos é O(n)
- O tempo de construção da estrutura é O(n lg n)
- No pior caso, o tempo de consulta é O(n) (temos que reportar todas as folhas)
  - Esse custo não é representativo, pois o tempo, na verdade, depende do número de pontos que satisfazem a consulta
- O tempo para reportar todos pontos que satisfazem a consulta é O(k)
- O tempo para encontrar as folhas que deveriam conter xmin e xmax é O(lg n) (árvore balanceada)
- Portanto, o custo total do algoritmo é O(k + lg n)

Kd-Trees (Árvores k-dimensionais)

- Vamos considerar a generalização do problema anterior para duas dimensões
- Agora, temos um conjunto, P, de n pontos no plano; e uma consulta intervalar sobre as duas dimensões (variáveis, no caso mais geral)
- A consulta será o par de intervalos ([xmin;xmax], [ymin; ymax])
- Um ponto p = (x,y) satisfaz a consulta sse
  - $xmin \leq x \leq xmax$ e $ymin \leq y \leq ymax$
- Dessa forma, a consulta em 2D é a composição das consultas 1D em x e y
- O problema é como generalizar a árvore binária para acomodar esse tipo de busca

Kd-Trees (Árvores k-dimensionais)

- A ideia fundamental da estrutura que usamos na busca 1D é que cada nó separava o espaço de busca em dois
- Em 2D, temos duas variáveis a considerar. Dessa forma, podemos dividir os pontos alternadamente conforme a variável x e y.
- Especificamente, na raiz, dividimos os pontos em duas metades conforme a coordenada x
  - A sub-árvore da esquerda contém os pontos com coordenadas menores ou iguais ao ponto de corte
  - A sub-árvore da direita contém os pontos com coordenadas maiores
  - Dividimos o plano em dois com uma reta vertical (mediana das coordenadas)



Kd-Trees (Árvores k-dimensionais)

- No segundo nível, passamos a dividir os pontos conforme a coordenada y
- Novamente, dividimos os pontos em duas metades segundo uma reta horizontal
- Os pontos abaixo ou sobre a reta são colocados na sub-árvore da esquerda
- Os pontos acima da reta são colocados na sub-árvore da direita
- O processo se repete nos níveis abaixo. Particionamos conforme a coordenada x nos níveis ímpares, e conforme a y nos níveis pares.

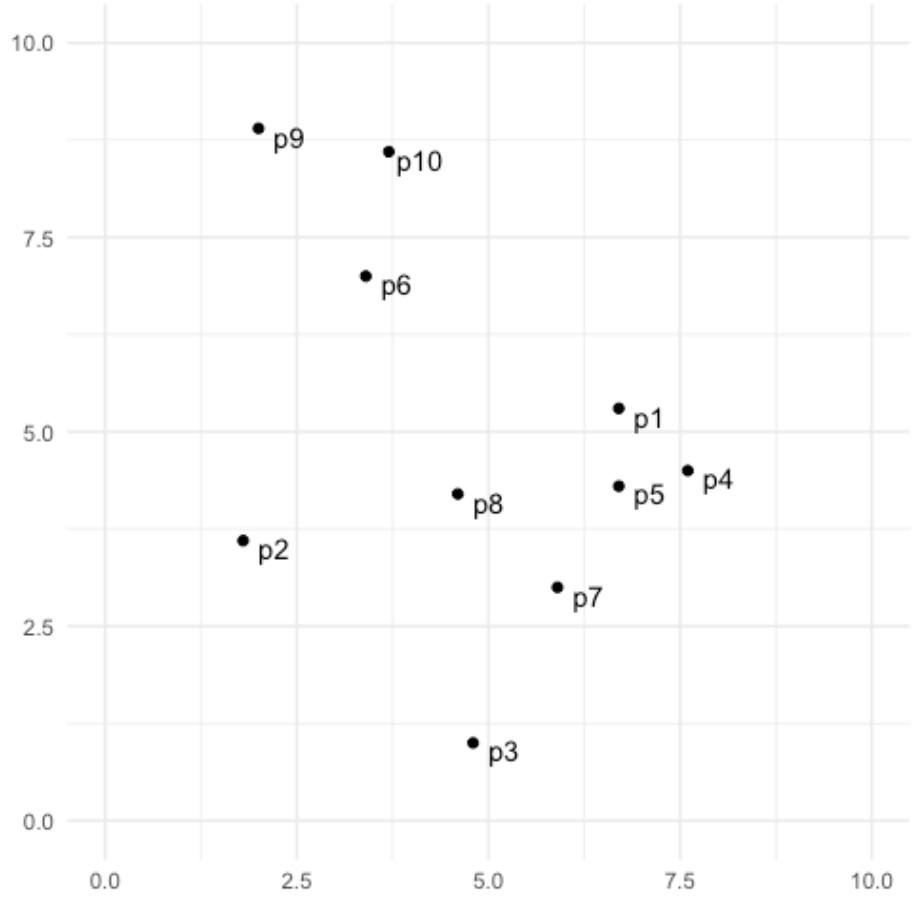
Kd-Trees (Árvores k-dimensionais)

Algorithm BUILDKDTREE($P, depth$)
Input. A set of points P and the current depth $depth$.
Output. The root of a kd-tree storing P.

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** $depth$ is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let P_2 be the set of points to the right of ℓ.
5. **else** Split P into two subsets with a horizontal line ℓ through the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE($P_1, depth + 1$)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE($P_2, depth + 1$)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

Kd-Trees (Árvores k-dimensionais)

- Exemplo:



Kd-Trees (Árvores k-dimensionais)

- O custo para construir a árvore é O(n lg n)
- O passo mais custoso em cada iteração (recursão) é a computação da mediana, a qual pode ser feita em tempo O(n)
  - Também podem ser criados índices ordenados dos pontos conforme as coordenadas x e y. Isso permite a computação da mediana em O(1).
- Cada nó na k-d tree representa uma região no plano
- O neto à esquerda da raiz representa a região abaixo da reta horizontal de seu pai, e à esquerda da reta vertical de seu avô
- Logo, precisamos visitar uma sub-árvore somente se a região definida por ela possui interseção com a consulta
  - Se ela estiver integralmente dentro da consulta, devemos reportar todas as suas folhas
- As regiões podem ser computadas durante o percurso na árvore

Kd-Trees (Árvores k-dimensionais)

Algorithm SEARCHKDTREE(v, R)
Input. The root of (a subtree of) a kd-tree, and a range R.
Output. All points at leaves below v that lie in the range.

```
1.  if v is a leaf
2.      then Report the point stored at v if it lies in R.
3.      else if region(lc(v)) is fully contained in R
4.              then REPORTSUBTREE(lc(v))
5.              else if region(lc(v)) intersects R
6.                      then SEARCHKDTREE(lc(v), R)
7.           if region(rc(v)) is fully contained in R
8.              then REPORTSUBTREE(rc(v))
9.              else if region(rc(v)) intersects R
10.                     then SEARCHKDTREE(rc(v), R)
```

Kd-Trees (Árvores k-dimensionais)

- Pode ser demonstrado que o algoritmo executa as consultas em tempo $O(n^{1/2} + k)$
- A estrutura pode ser facilmente adaptada para k dimensões
- No i-ésimo nível, os pontos são particionados conforme a dimensão (i mod k) + 1
- O algoritmo possui um custo relativamente alto se o número de pontos retornados for pequeno
- É possível reduzir o custo da consulta, usando estruturas multiníveis (árvores de intervalos)
  - Essencialmente, as árvores são como no caso 1D, e cada nó tem um link para uma outra árvore 1D construída sobre a outra dimensão

Leitura

- Triangulação de polígonos
  - Capítulos 1(Computational Geometry in C, 2nd ed., J. O'Rourke)
  - Seções 5.1, 5.2, 5.4, 5.5 (O'Rourke)
  - Seção 3.1 (De Berg et al.)
- Diagrama de Voronoi
  - Seção 7.1 (De Berg et al.)
  - Seção 7.2 (De Berg et al., opcional -- computação eficiente do diagrama de Voronoi)
- Busca intervalar
  - Capítulo 5 (Computational Geometry: Algorithms and Applications, 3rd ed. De Berg et al.)
  - Seção 21.3 (Goodrich e Tamassia)