Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Mateus Barros Rodrigues

Implementação de algoritmos para consultas de segmentos em janelas

São Paulo
Setembro de 2016

Implementação de algoritmos para consultas de segmentos em janelas

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Carlos Eduardo Ferreira

São Paulo
Setembro de 2016

Resumo

Este trabalho de conclusão de curso fundamentou-se na compreensão e implementação em linguagem *python* de um algoritmo para consultas de intersecções de segmentos de retas com janelas retangulares no espaço, um subproblema de geometria computacional conhecido por: buscas em regiões ortogonais. Este algoritmo foi o foco da tese de mestrado de Álvaro Junio Pereira Franco. Além da implementação, foi feita também a adaptação do visualizador de algoritmos geométricos feito por Alexis Sakurai Landgraf para exposição dos resultados obtidos.

Palavras-chave: Geometria, janelas, segmentos, buscas.

Sumário

Capítulo 1

Introdução

Neste trabalho de conclusão de curso foi abordado o problema de *consultas de segmentos em janelas*, um problema de *buscas em intervalos ortogonais*, que é um dos tópicos fundamentais da área de geometria computacional.

Dado um conjunto S de segmentos no espaço (Seja no $\mathbb{R}$, $\mathbb{R}^2$, etc.) e uma janela W de lados paralelos, queremos responder rapidamente a seguinte pergunta: *quais segmentos de S estão contidos na ou intersectam a janela W?*

Este trabalho foi baseado em *Consultas de segmentos em janelas: algoritmos e estruturas de dados* de Álvaro Junio (2009), portanto seguiremos a mesma divisão do problema que foi proposta nessa dissertação: Encontrar pontos contidos em janelas e achar todos os segmentos que intersectam com um dado segmento (Horizontal ou vertical). Seguiremos também a mesma divisão de capítulos: Primeiramente apresentaremos definições e primitivas geométricas, dedicaremos um capítulo para falar de consultas de pontos em janelas, um para falar de encontrar intersecção de segmentos e finalmente um onde agregaremos esses algoritmos para resolver o problema proposto. Todo o código desenvolvido foi escrito em linguagem *python* e está disponível no gitHub.

Capítulo 2

Definições e Primitivas

Explicaremos a seguir algumas das noções fundamentais que serão utilizadas ao longo do trabalho:

2.1 Pontos e Segmentos

Neste trabalho trataremos basicamente com pontos e segmentos de reta no espaço ($\mathbb{R}$ e $\mathbb{R}^2$). Sejam $x, y \in \mathbb{R}$ definimos um **ponto** no $\mathbb{R}^2$ como um par $p = (x, y)$. Um **segmento** s é da forma $s \coloneqq \overline{(x_1, y_1)(x_2, y_2)} \in \mathbb{R}^2 \times \mathbb{R}^2$ onde $u = (x_1, y_1)$ e $v = (x_2, y_2)$ são pontos chamados de **pontos extremos** de s.

2.2 Comparações entre pontos

Uma outra definição que será usada copiosamente ao longo desta monografia é a relação de desigualdade associada à uma dada coordenada. Sejam u, v pontos, dizemos que $u \leq_x v$ caso $x(u) < x(v)$ ou $x(u) = x(v)$ e $y(u) \leq y(v)$ (Simetricamente definido para desigualdades em relação à coordenada y), ou seja, sempre comparamos primeiro a coordenada de maior interesse e desempatamos pela segunda coordenada nas comparações.

2.3 Posição relativa entre ponto e segmento

Usaremos também bastante a noção de posição relativa entre pontos e segmentos, isto é, dado um ponto p e um segmento s, queremos saber se p se encontra **à esquerda**, **à direita** ou **sobre** o segmento s.

Sejam $p \coloneqq (x_1, y_1) \in \mathbb{R}^2$, $s \coloneqq \overline{(x_2, y_2)(x_3, y_3)} \in \mathbb{R}^2 \times \mathbb{R}^2$ e $d \coloneqq \det \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}$

Dizemos que p está **à esquerda** de s caso $d > 0$, que está **sobre** s caso $d = 0$ e que está **à direita** de s caso contrário. Seguem a seguir os trechos de código que foram usados no trabalho para realizarmos essas verificações:

Algoritmo 1 Retorna **TRUE** caso p esteja à esquerda de s

```
1 def left(p,s):
2     b = s.beg
3     c = s.end
4     if b.x == c.x and p.x == b.x: return p.y > c.y
5     if b.y == c.y and p.y == b.y: return p.x < c.x
6     return (b.x-p.x)*(c.y-p.y) - (b.y-p.y)*(c.x-p.x) > 0
```

Algoritmo 2 Retorna **TRUE** caso p esteja à direita de s

```
1 def right(p,s):
2     b = s.beg
3     c = s.end
4     if b.x == c.x and p.x == b.x: return p.y < b.y
5     if b.y == c.y and p.y == b.y: return p.x > c.x
6     return not(left_on(p,s))
```

Algumas ressalvas sobre essas funções:

- A única diferença da função *left_ on* em relação à função *left* é que ela também retorna *true* caso o ponto esteja sobre o segmento dado.

- As modificações presentes nas linhas 4 e 5 foram adicionadas apenas para resolverem os casos degenerados apresentados no capítulo x.

Capítulo 3

Consultas sobre pontos em janelas

Nesse capítulo mostraremos os algoritmos implementados para localizarmos todos os pontos numa dada janela e algumas variações desse problema. Todas as provas de corretude e de eficiência dos algoritmos expostos, tanto deste capítulo quanto dos próximos, poderão ser encontradas na dissertação de Álvaro Junio (2009).

3.1 Janela limitada - Caso unidimensional

Analisaremos primeiramente o problema no espaço $\mathbb{R}$, ou seja, nossos pontos estarão todos contidos na reta. Sejam u, v pontos na reta tais que $u \leq v$, definimos uma **janela** como sendo um *intervalo fechado* com extremos u e v.

3.1.1 Pré-processamento

Para resolvermos rapidamente sucessivas consultas sobre um dado conjunto de pontos, precisaremos armazenar esses dados em uma estrutura de dados apropriada. A estrutura que usaremos será um tipo de *árvore de busca binária balanceada* (ABBB) chamada de árvore limite, a seguir está o trecho de código referente à construção da árvore:

Algoritmo 3 Retorna uma raíz v de uma árvore limite 1D construída sobre um conjunto de pontos ordenados.

```
1 def buildTree(self,points):
2     v = Node(None)
3     l = points[:len(points)//2]
4     r = points[len(points)//2:]
5
6     v.point = points[len(points)//2-1]
7
8     if len(points) == 1:
9         v.l = v.r = None
10    else:
11        v.l = self.buildTree(l)
12        v.r = self.buildTree(r)
13
14    return v
```

3.1.2 Realizando a consulta

Seja P um conjunto de pontos e seja $W = [w_1, w_2]$ uma janela, podemos consultar todos os pontos de $P \subset W$ da seguinte forma:

1. Montamos a ABBB sobre o conjunto P.
2. Achamos o *ponto divisor* de P, este é o ponto que se encontra na raiz da subárvore que contém os pontos $S \coloneqq (v \mid w_1 \leq v \leq w_2)$, chamaremos esse ponto de v_{div}.
3. Percorremos a subàrvore esquerda de v_{div} verificando se o ponto r da raiz é tal que $w_1 \leq r$, caso seja, adicionamos todos os pontos dessa subàrvore na resposta e nos movemos para a subàrvore esquerda, caso contrário vamos para a subárvore direita. Ao chegar na folha apenas verificamos se $w_1 \leq r \leq w_2$ e adicionamos na resposta caso seja verdade.
4. Percorremos a subárvore direita de v_{div} de forma simétrica ao item 3.

Segue a implementação das rotinas supracitadas juntamente com suas funções auxiliares:

Algoritmo 4 Retorna *true* caso o ponto p esteja contido no intervalo *rng*.

```
1 def inRange(self,rng,p):
2     w1,w2 = rng
3     return w1 <= p and p <= w2
```

Algoritmo 5 Retorna o ponto divisor v_{div} de uma ABBB referente à uma dada janela *rng*.

```
1 def findDividingNode(self,rng):
2     w1,w2 = rng
3     div = self.root
4
5     while(not div.isLeaf() and (w1 > div.point or w2 <= div
          .point)):
6         if w2 <= div.point:
7             div = div.l
8         else:
9             div = div.r
10    return div
```

Algoritmo 6 Devolve uma lista com as folhas de uma dada árvore.

```
def listSubTree(self):
    l = []
    self.findLeaves(l)
    return l

def findLeaves(self,l):
    if self.isLeaf():
        l.append(self.point)

    if self.l is not None: self.l.findLeaves(l)
    if self.r is not None: self.r.findLeaves(l)
```

Algoritmo 7 Retorna uma lista com todos os pontos contidos numa dada janela *rng*.

```
def query(self,rng):
    div = self.findDividingNode(rng)
    p = []

    if div.isLeaf():
        if self.inRange(rng,div.point):
            p.append(div.point)
    else:
         v = div.l
         while(not v.isLeaf()):
             if w1 <= v.point:
                 subtree = v.r.listSubTree()
                 p += subtree
                 v = v.l
             else:
                 v = v.r

         if self.inRange(rng,v.point):
             p.append(v.point)

         v = div.r

         while(not v.isLeaf()):
             if w2 > v.point:
                 subtree = v.l.listSubTree()
                 p += subtree
                 v = v.r
             else:
                 v = v.l
         if self.inRange(rng,v.point):
             p.append(v.point)

     return p
```

3.1.3 Análise

- O pré-processamento requer que seja feita uma ordenação sobre o conjunto de pontos de entrada, portanto tem complexidade $\Theta(n \log n)$.

- A árvore terá altura $\mathcal{O}(\log n)$ e visitaremos $\mathcal{O}(\log n)$ pontos em cada subàrvore de v_{div}, além disso, consumiremos tempo $\mathcal{O}(k)$ para visitar os k pontos das folhas que estão contidos no intervalo e devem aparecer na resposta final. Portanto a complexidade final da consulta é da ordem $\mathcal{O}(n \log n + k)$.

Capítulo 4

Desenvolvimentos

Embora neste exemplo tenhamos apenas um capítulo, entre a introdução e a conclusão de uma monografia podemos ter uma sequência de capítulos descrevendo o trabalho e os resultados. Estes podem descrever fundamentos, trabalhos relacionados, método/modelo/algoritmo proposto, experimentos realizados, resulatdos obtidos.

Cada capítulo pode ser organizado em seções, que por sua vez pode conter subseções.

Um exemplo de figura está na figura 4.1.

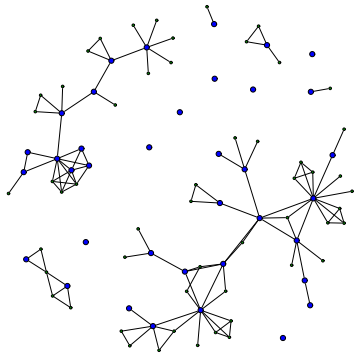

Figura 4.1: *Exemplo de uma figura.*

Capítulo 5

Conclusões

Texto texto[1].

[1]Exemplo de referência para página Web: www.vision.ime.usp.br/~jmena/stuff/tese-exemplo

Referências Bibliográficas

Álvaro Junio(2009) Álvaro Junio. Consultas de segmentos em janelas: algoritmos e estruturas de dados. Dissertação de Mestrado, Instituto de Matemática e Estatística, Universidade de São Paulo, Brasil. Citado na pág. 1, 5