

Algoritmos e Estrutura de Dados III

Primeiro Trabalho Prático - Hipercampos

Pablo Cecilio Oliveira
Alexander Cristian

1 Introdução

Na Ciência da Computação, o estudo de algoritmos para resolução de problemas geométricos é conhecido como Geometria Computacional. De forma geral, o objetivo deste ramo é resolver de maneira eficiente utilizando o menor número possível de operações sobre os elementos geométricos elementares.[4]

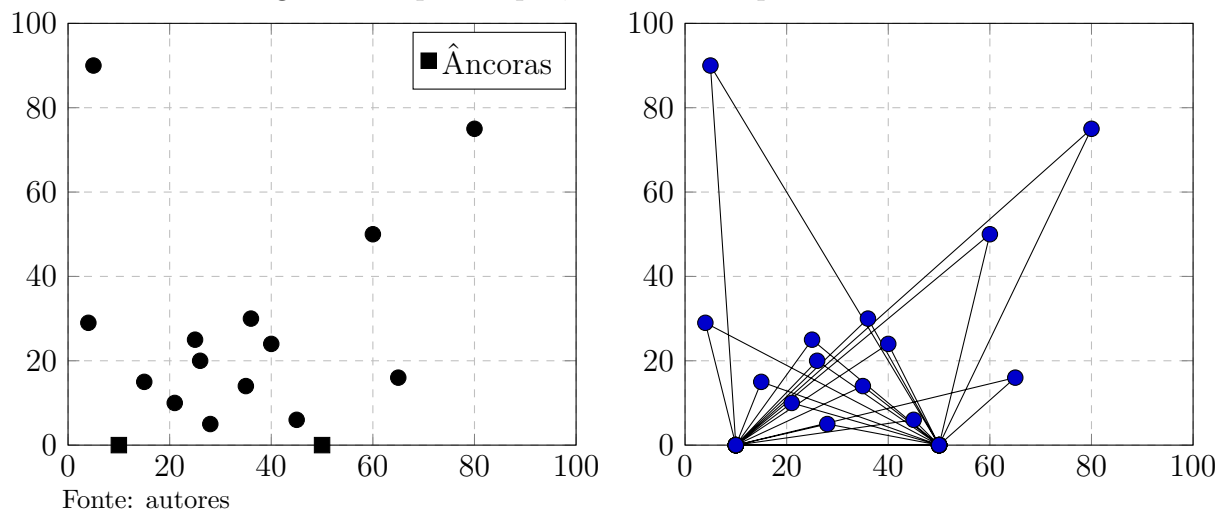
Neste trabalho, é apresentada a solução para o problema denominado como Hipercampos. O qual pode ser visto em diversas maratonas de programação.

1.1 Especificação do Problema

No problema de Hipercampos, um plano cartesiano em \mathbb{R}^2 possui duas "âncoras", dois pontos onde $Y = 0$ e os valores de seus eixo em X variam de X_A até X_B , formando assim um segmento de reta horizontal, tal que $0 < X_A < X_B$.

Ao plano cartesiano também somam-se um conjunto P de N pontos (X, Y) , tal que $0 < X_i, Y_i \leq 10^4$.

Figura 1: Hipercampos, entrada sem processamento.



O objetivo do problema é ligar os pontos contidos em P às âncoras, formando assim um número máximo de triângulos sem que esses se interceptem. E para esse proposito é apresentado um algoritmo contido no programa apresentado neste trabalho.

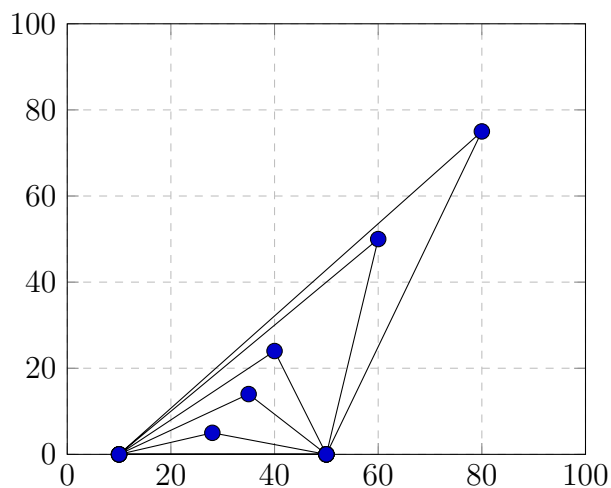
1.2 Entrada

A primeira linha da entrada contém três inteiros, N ($1 \leq N \leq 100$), X_A e X_B ($0 < X_A < X_B \leq 10000$) representando, respectivamente, o número de pontos no conjunto P e as abscissas das âncoras A e B . As N linhas seguintes contêm, cada uma, dois inteiros X_i e Y_i ($0 < X_i, Y_i \leq 10000$), representando as coordenadas dos pontos, para $1 \leq i \leq N$. Não há pontos coincidentes e não há dois pontos u e v distintos tais que A, u, v ou B, u, v sejam colineares.

1.3 Saída

O programa imprime uma linha contendo um inteiro, representando o número máximo de pontos de PP que podem ser ligados com interseção de segmentos apenas nas âncoras.

Figura 3: Solução encontrada pelo algoritmo.



Fonte: autores

1.4 Solução proposta

O método em questão usado para resolver esse problema se baseia na exploração do sistema de coordenadas baricêntricas e na orientação dos segmentos de retas formados pela conexão dos pontos. Portanto em primeiro lugar é verificada a orientação das retas para isolar os casos em que elas se interceptam ou são colineares.

Logo em seguida precisa-se saber quando um ponto está contido em um triângulo. Uma solução simples seria traçar uma reta que segue horizontalmente para a direita, e depois fazendo comparações para saber quantas vezes ela intercepta o polígono formado, se o resultado for um número par o ponto está fora, se for ímpar ele está dentro. Porém isso levaria o programa a executar muitas operações, então evoluindo desse conceito chegamos

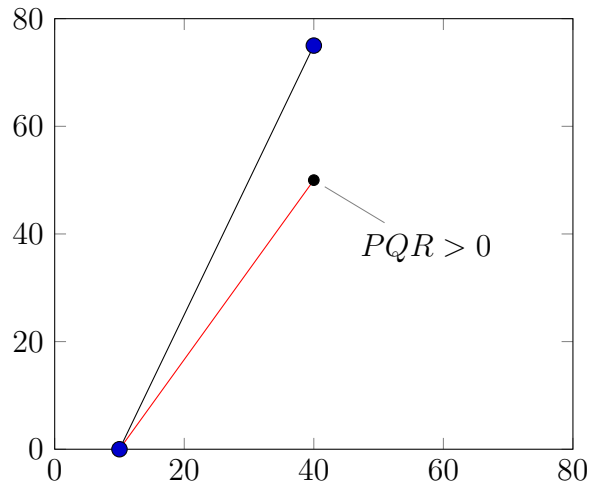
a uma solução usando as coordenadas baricêntricas, onde é verificado em qual lado do meio plano criado pelas arestas está o ponto.

Verifica por meio da expressão de orientação da reta:

$$(y_2 - y_1)(x_3 - x_2) - (y_3 - y_2)(x_2 - x_1)$$

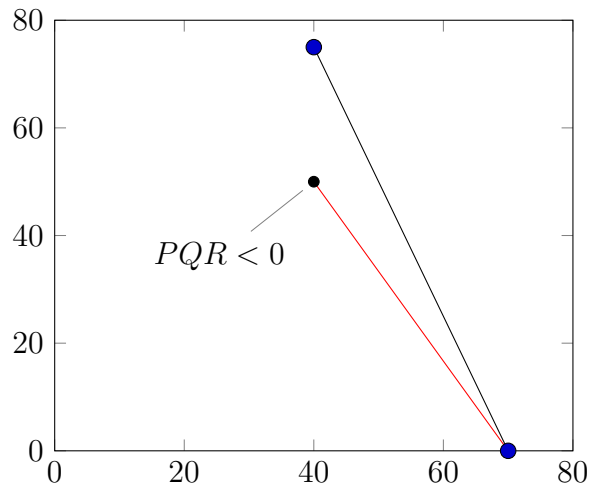
Se os pontos são colineares, e a orientação do triângulo (horário ou anti-horário)

Figura 4a: Determinando o ponto interno ao triângulo



Fonte: autores

Figura 4b: Determinando o ponto interno ao triângulo



Fonte: autores

2 Implementação

Tabela 1: Funções do programa

Funções	Finalidade	Complexidade*
<i>debug()</i>	Função que verifica a condição para retorno de possíveis bugs no programa.	$O(1)$
<i>create()</i>	Inicializa a Lista encadeada.	$O(1)$
<i>insere()</i>	Insere os dados em uma lista encadeada.	$O(1)$
<i>printCJT()</i>	Imprime uma Lista encadeada.	$O(n)$
<i>sizeCJT()</i>	Retorna o tamanho da lista encadeada.	$O(n)$
<i>dump()</i>	Libera a memória alocada pela lista.	$O(n)$
<i>isEmpty()</i>	Verifica se uma lista encadeada está vazia.	$O(1)$
<i>openFILE()</i>	Abre o arquivo solicitado e transfere os dados para uma lista encadeada.	$O(n)$
<i>saveFILE()</i>	Salva a solução do problema em um arquivo.	$O(1)$
<i>chkFILE()</i>	Verifica por possíveis erros de entrada em um arquivo.	$O(1)$
<i>showerro()</i>	Retorna possíveis erros no arquivo de entrada.	$O(1)$
<i>ask()</i>	Solicita a confirmação do usuário caso erros de entrada sejam encontrados.	$O(1)$
<i>cpyCJT()</i>	Copia os dados de uma lista encadeada para outra lista encadeada.	$O(1)$
<i>PQR()</i>	Algoritmo de orientação do ponto em relação a reta da ancora.	$O(1)$
<i>findMAX()</i>	Função recursiva que determina o maior conjunto de pontos que se encontram dentro do triângulo formado pelas ancoras e um ponto (x, y) .	$O(n)$
<i>soluciona()</i> <i>solucao()</i>	Funções de chamada e retorno para a execução do algoritmo	$O(1)$
<i>plotGraph()</i>	PIPE para o gnuplot com a finalidade de renderizar os arquivos .svg contendo respectivamente, a entrada e saída da solução do problema.	$O(n)$

Fonte: autores

3 Análise de Complexidade

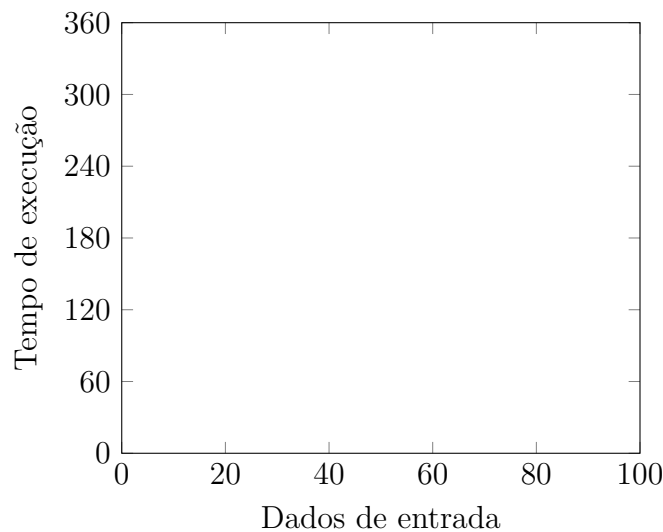
$$f(n) = 11 + (n + 1) * \frac{n}{n - 1} + \frac{n}{2} * \frac{1}{2n} + n + \frac{n}{2} + 4n$$

$$f(n) = 11 + \frac{n^2 + n}{n - 1} + \frac{n}{4n} + \frac{n}{2} + 5n$$

$$f(n) = \frac{26n^2 + 27n - 45}{4(n - 1)}$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Figura 5: Tempo de execução do algoritmo



Fonte: autores

4 Considerações finais

O Trabalho computacional 1 da disciplina foi uma grande oportunidade para aprender sobre grafos e LCS, que rodeiam o algoritmo ótimo para a solução desse problema, o que é a introdução para programação dinâmica e acreditamos ser o intuito desse trabalho, também proporcionou um contato maior com a análise de complexidade do algoritmo.

Um dos maiores problemas no desenvolvimento foi encontrar um algoritmo que possuísse um comportamento adequado quando a entrada de valores é muito grande. Apesar da forte base matemática de nossos métodos, em alguns casos eles podem levar a uma falta de precisão, porque o sistema de números de ponto flutuante tem tamanho limitado e na maioria das vezes lida com aproximações. O problema ocorre às vezes quando um ponto p deve estar exatamente na borda de um triângulo, as aproximações levam a falhar no teste.

Para a construção de gráficos que auxiliam em uma melhor visualização do trabalho foi necessário o gnuplot.

Referências

- [1] et al. Elin, Kisielewicz. How to determine if a point is in a 2d triangle?
[https://stackoverflow.com/questions/2049582/
how-to-determine-if-a-point-is-in-a-2d-triangle](https://stackoverflow.com/questions/2049582/how-to-determine-if-a-point-is-in-a-2d-triangle). [Acesso em:
23-Agosto-2018].
- [2] Cédric Jules. Accurate point in triangle test. [http:
//totologic.blogspot.com/2014/01/accurate-point-in-triangle-test.html](http://totologic.blogspot.com/2014/01/accurate-point-in-triangle-test.html).
[Acesso em: 23-Agosto-2018].
- [3] Patrick Prosser. Geometric algorithms.
<http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>. [Acesso em:
23-Agosto-2018].
- [4] Wikipedia contributors. Computational geometry — Wikipedia, the free
encyclopedia. [https://en.wikipedia.org/w/index.php?title=Computational_
geometry&oldid=841504892](https://en.wikipedia.org/w/index.php?title=Computational_geometry&oldid=841504892), 2018. [Acesso em: 3-Setembro-2018].