Algorítimos e Estrutura de Dados III
Primeiro Trabalho Prático - Hipercampos

Pablo Cecilio Oliveira
Alexander Cristian

1 Introdução

A computação é um instrumento muito útil para encaminhar a resolução de um problema. Podendo ele ser uma coisa simples, ou então, até mesmo algo mais elaborado. Tendo isso em vista, o primeiro trabalho prático da disciplina nos desafia a resolver o problema dos Hipercampos. O qual é visto em diversas maratonas de programação, e pode ser resolvido de varias formas. Neste trabalho será aborada a solução julgada de melhor implementação por parte dos discentes.

1.1 Especificação do Problema

Em uma visão mais detalhada do problema, temos um plano cartesiano onde são dadas duas âncoras, dois pontos onde Y= 0 e os valores de X variam de A até B formando assim um segmento de reta horizontal, tal que $0 < X_A < X_B$. E também se recebe como entrada um conjunto P de N pontos na forma (X, Y) sendo X e Y maiores do que 0.

Ao ligar um dos pontos contidos em P às âncoras, usando segmentos de reta, formamos um triangulo, deve-se ligar vários pontos, mas de modo que eles se interceptem apenas nas ancoras. Se expressando de uma maneira mais simples, deve-se achar o maior número de triângulos contidos um dentro do outro, que se cruzam apenas na base.

Portanto o algoritmo trabalhado computa o número máximo de pontos que é possível ligar com interseção de segmentos apenas nas ancoras, de acordo com as entradas do usuário.

1.2 Entrada

A primeira linha da entrada contém três inteiros, $N(1 \leqslant N \leqslant 100)$, X_A e X_B $(0 < X_A < X_B \leqslant 10000)$ 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 Xi e Yi $(0 < X_i, Y_i \leqslant 10000)$,representando as coordenadas dos pontos, para $1 \leqslant i \leqslant N$. 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.

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 esta contido em um triangulo. 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 impar ele está dentro. Porem 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.

2 Implementação

-mergesort Foi usado o mergesort para ordenar a lista encadeada pelo maior Y dos pontos. Esse algoritmo de ordenação segue o estilo dividir e conquistar, possuindo complexidade de $O(n \log n)$.

-plotGraph Cria um aquivo "data.temp" onde são armazenados os valores de entrada, e logo em seguida escreve os comandos do "gnuplot" e manda executá-lo.

-dump Essa função usa um vetor de nomes de objetos R e produz representações de texto dos objetos em um arquivo ou conexão.

-soluciona Chama as funções para formar o gráfico e achar o ponto de maior valor do conjunto.

-PQR Verifica por meio da expressão de orientação da reta:

$$(y2 - y1)(x3 - x2) - (y3 - y2)(x2 - x1)$$

S5e os pontos são colineares, e a orientação do triangulo (horário ou anti-horário)

-solucao Imprime a saída do problema

3 Análise de Complexidade

$O(N^2)$

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 analise 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`. [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`. [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].