Problema Grande Tratado de Bytelândia

Felipe Rodrigues de S. Freitag[1], **Paulo Miranda e Silva Sousa**[1], **Wladimir A. Tavares**[1], **Ricardo Reis Pereira**[1]

[1]Universidade Federal do Ceará (UFC) - Campus Quixadá
Av. José de Freitas Queiroz, 5003 - Quixadá, CE - Brasil

felipefreitag@alu.ufc.br,

{wladimirufc,ricardoreispereira,paulomirandamss12}@gmail.com

Abstract. *This article aims to explain and demonstrate an efficient solution to the Great Treaty of Byteland, problem from the sub-regional phase of the XXVIII SBC Programming Marathon, a qualifying competition for the ACM International Collegiate Programming Contest (ICPC).*

Resumo. *Este artigo tem como objetivo explicar e mostrar como resolver de maneira eficiente a questão Grande Tratado de Bytelândia, problema da fase sub-regional da XXVIII Maratona de Programação da SBC, competição classificatória para o ACM International Collegiate Programming Contest (ICPC).*

Palavras-chave: Algoritmos. Programação competititiva. Geometria computacional.

1. Introdução

A Maratona de Programação é um evento da Sociedade Brasileira de Computação (SBC) que existe desde o ano de 1996. Nasceu das competições regionais classificatórias para as etapas mundiais da competição de programação, o International Collegiate Programming Contest, e é parte da regional sul-americana do evento.

Ela se destina a alunos e alunas de cursos de graduação e início de pós-graduação na área de Computação e afins (Ciência da Computação, Engenharia de Computação, Sistemas de Informação, Matemática, etc.). A competição promove nos estudantes a criatividade, a capacidade de trabalho em equipe, a busca de novas soluções de software e a habilidade de resolver problemas sob pressão. A cada ano, observa-se que as instituições de ensino e, principalmente, as grandes empresas da área têm valorizado os alunos que participam do evento.

O artigo será organizado da seguinte maneira. Na seção 2, apresentaremos o problema Grande Tratado de Bytelândia e faremos uma pequena discussão sobre o problema. Na seção 3, apresentaremos o que é geometria computacional e discutiremos sobre fecho convexo e sobre o algoritmo utilizado na solução do problema. Na seção 4, explicaremos como resolver o problema. Por fim, na seção 5, faremos as nossas considerações finais.

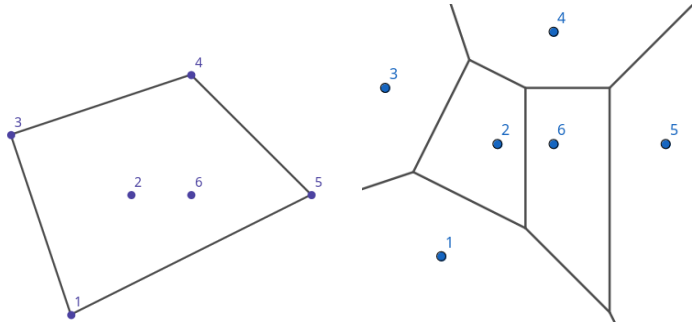
2. Grande Tratado de Bytelândia

Nesta seção, apresentaremos o problema Grande Tratado de Bytelândia presente na fase sub-regional. A fase sub-regional da XXVIII Maratona de Programação, realizada em 2023, continha 13 questões enumeradas de A a M. Cada uma dessas questões aborda um problema do campo da computação e é dever dos participantes conseguir entender o problema, além de criar uma solução para resolvê-lo utilizando o conhecimento prévio sobre algoritmos. Para uma solução

ser aceita pelos juízes da competição, além de responder corretamente o conjunto de casos de testes, é necessário que o algoritmo gere as respostas dentro de um limite de tempo e de memória, limitando assim a complexidade do algoritmo, ou seja, é necessário que o algoritmo seja correto e eficiente. Dentre as 13 questões da fase sub-regional do ano de 2023 estava o problema Grande Tratado de Bytelândia (letra G)[1].

Em resumo, o problema nos diz que existem vários reinos e que eles estão discutindo um Tratado de Divisão, que dividirá todas as terras do mundo entre eles. Ademais, todos os reinos possuem uma única capital e o Tratado de Divisão será baseado em suas localizações que serão representadas por meio de coordenadas em um plano bidimensional. Como o Tratado de Divisão se refere não apenas ao mundo conhecido, mas também a quaisquer territórios ainda não descobertos ou habitados, podemos concluir que esse plano é infinito. O Tratado de Divisão também nos diz que cada pedaço de terra pertence ao reino cuja capital é mais próxima em uma linha reta, entretanto, caso ocorra empate entre as distâncias de duas ou mais capitais, esse lugar estará na fronteira entre os reinos. Porém, alguns reinos podem ficar cercados por outros, enquanto outros reinos podem ficar com território ilimitado e, por isso, alguns monarcas estão contestando o Tratado de Divisão. Tendo conhecimento das coordenadas de todas as capitais e sabendo que não existem reinos em uma mesma coordenada e que podemos assumir que toda capital tem tamanho insignificante, nosso dever é responder quais reinos teriam território infinito sob o Tratado de Divisão.

Para uma compreensão mais nítida, analisaremos a seguir o segundo caso de teste, uma vez que a entrada é um pouco mais extensa do que a do primeiro caso de teste e envolve a exclusão de alguns reinos da resposta.



(a) Polígono convexo (b) Diagrama de voronoi

Figura 1. Visualização do segundo caso de teste

Observando o polígono acima (Figura 1.a), fica evidente que os reinos 1, 3, 4 e 5, que compõem a resposta, coincidem exatamente com os vértices que formam o polígono. Enquanto isso, os reinos 2 e 6, que foram excluídos da resposta, estão localizados no interior do polígono. Essa exclusão faz sentido, uma vez que pontos dentro do polígono não podem ser considerados como capazes de ter território infinito considerando o Tratado de Divisão, enquanto os pontos que formam o polígono possuem território infinito. Ademais, observando o diagrama de voronoi (Figura 1.b), também é possível ver que os reinos 2 e 6 não possuem território infinito.

No entanto, o autor da questão oculta um detalhe importante que não é explicitamente mencionado, mas que certamente é testado nos casos de teste: a consideração de pontos colineares. Para ilustrar esse caso, imagine um novo reino, com capital localizada nas coordenadas $(4, 2)$ na aresta formada pelos vértices 1 e 5 do polígono (Figura 1.a). Esse reino também deve estar na

[1] `https://codeforces.com/gym/104555/problem/G`

resposta, pois seu território é igualmente infinito. Portanto, a partir dessa análise, podemos inferir que pontos colineares, que estão nas arestas do polígono, também devem fazer parte da resposta. Então, o que nos resta é saber como encontrar esse polígono e as capitais dos reinos que o formam.

3. Geometria computacional

O campo da geometria computacional é um ramo da Ciência da Computação que estuda algoritmos e estruturas de dados para a resolução computacional de problemas geométricos de maneira eficiente [De Berg 2000]. Vários problemas estão relacionados com a geometria computacional, como interseção de segmentos, triangularização de polígonos, localização de pontos, diagrama de voronoi, entre outros.

3.1. Fecho convexo

O fecho convexo ou envoltória convexa é o menor polígono convexo que engloba todos os pontos de um conjunto dado em um plano ou em um espaço de maior dimensão. Em termos simples, é uma figura geométrica que contém todos os pontos do conjunto, mas é convexa, o que significa que, ao desenhar uma linha entre quaisquer dois pontos dentro dela, essa linha permanece inteiramente dentro da figura. Então, se conseguirmos aprender como utilizar um algoritmo que o encontra, seremos capazes de resolver essa questão. O algoritmo utilizado será o Monotone Chain, que é capaz de encontrar o fecho convexo com complexidade assintótica de $O(n \lg n)$. Além disso, também existe o algoritmo de Graham Scan, que também é capaz de encontrar o fecho convexo com a mesma complexidade assintótica.

3.2. Monotone Chain

O primeiro passo do algoritmo consiste em dividir o fecho convexo em duas partes, uma superior e outra inferior, de forma que possamos construir cada parte do fecho convexo separadamente e, ao final dessa construção, o que restará a ser feito é a união dessas partes.

Para realizar essa divisão, é necessário primeiro ordenar os pontos em ordem crescente de coordenadas x e, em caso de empate, em ordem crescente de coordenadas y. Isso garantirá que o primeiro ponto da lista seja o mais à esquerda e, em caso de empate em x, o mais abaixo, enquanto o último ponto será o mais à direita e, em caso de empate em x, o mais acima. Após essa ordenação, temos que passar por todos os pontos, menos o primeiro, e decidir, antes de tudo, se o ponto está na parte superior ou inferior do fecho convexo. Faremos isso utilizando duas variáveis, sendo elas: A, o primeiro ponto, e B, o último ponto. Agora, observe a figura a seguir para um melhor entendimento:

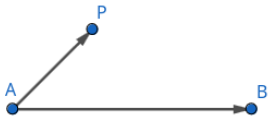


Figura 2. Exemplo para a decisão da parte do fecho convexo

Então, seja P o ponto que queremos saber se está na parte superior ou inferior, ele só estará na parte superior do fecho convexo se $\vec{AP} \times \vec{AB} \leq 0$ e só estará na parte inferior do fecho convexo se $\vec{AP} \times \vec{AB} \geq 0$. Vale ressaltar que, como os pontos A e B fazem a divisão do fecho convexo, eles estarão em ambas as partes e ambas as partes do fecho convexo já devem começar com o ponto A.

Trataremos agora da construção da parte superior do fecho convexo, uma vez que sabemos detectar se um ponto está incluso nessa parte. Durante sua construção iremos guardar os pontos

que atualmente compõem a parte superior em um vetor, pois nele podemos inserir e remover pontos do fim e acessar índices em tempo constante. Tal construção ocorre passando por todos os pontos que já estão ordenados, ignorando o primeiro ponto, pois ele já está na resposta da parte superior. Além disso, a construção se divide em dois casos que serão explicados logo abaixo. Outrossim, considere que em ambos os casos estamos querendo incluir o i-ésimo ponto de uma iteração qualquer na parte superior do fecho convexo e que essa parte já contém dois ou mais pontos, pois caso a parte superior tenha menos que dois pontos o i-ésimo ponto pode ser colocado na parte superior sem problemas.

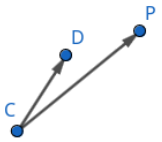


Figura 3. Visualização do primeiro caso

Esse primeiro caso é bem simples. Seja C o penúltimo ponto colocado na parte superior do fecho convexo, D o último ponto colocado e P o ponto que queremos colocar, se $\vec{CD} \times \vec{CP} \leq 0$, então apenas incluiremos P na parte superior, pois não precisamos fazer nenhuma mudança.

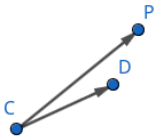


Figura 4. Visualização do segundo caso

Esse segundo caso já é um pouco mais complexo. Seja C o penúltimo ponto colocado na parte superior do fecho convexo, D o último ponto colocado e P o ponto que queremos colocar, se $\vec{CD} \times \vec{CP} > 0$, então isso quer dizer que D não faz parte da resposta e, por isso, podemos removê-lo, pois, se apenas incluíssemos o ponto P sem remover D, o polígono não seria mais convexo. Após a remoção, iremos realizar algumas verificações. Se o primeiro caso for verdadeiro, basta adicionarmos P na resposta. Se o segundo caso for verdadeiro, iremos remover D da resposta e repetir essas verificações. Se a parte superior possuir apenas um ponto, iremos adicionar P na resposta sem nenhuma verificação.

Agora, imagine que estamos na última iteração, então $P = B$, isso quer dizer que podemos colocar P na parte superior sem nenhuma verificação, pois ele, assim como A, deve estar na parte superior do fecho convexo.

A maneira com que a parte inferior é construída é análoga a parte superior e, por isso, não será comentada. Ademais, caso o leitor não queria aceitar pontos colineares, basta alterar um pouco as fórmulas de produto vetorial. No caso em que precisamos decidir se P está na parte inferior ou superior do fecho convexo, se $\vec{AP} \times \vec{AB} = 0$, então ele não estará na resposta de nenhuma das partes. Agora, o que nos resta é alterar as fórmulas do primeiro e do segundo caso, pois, nos casos em que P é colinear ao último e ao penúltimo ponto da parte superior do fecho convexo, o último ponto será removido do vetor, então a fórmula do primeiro caso seria $\vec{CD} \times \vec{CP} < 0$, enquanto a do segundo caso seria $\vec{CD} \times \vec{CP} \geq 0$. Lembre-se que essa alteração também deve ser feita para a construção da parte inferior do fecho convexo.

Após a geração de ambas as partes, a única coisa que resta é unir a parte superior com a inferior. Perceba que, a todo momento que adicionamos novos pontos, estamos indo da esquerda

para a direita no plano, isto é, a parte superior está no sentido horário e a inferior no sentido anti-horário, então, para unirmos a parte inferior à parte superior, teremos que passar pela ordem inversa dos pontos da parte inferior. Entretanto, perceba também que temos que começar do penúltimo ponto até o segundo ponto, pois esses pontos estão tanto na parte superior quanto na inferior, já que se tratam dos pontos A e B, e, sendo assim, evitaremos repetições.

Para finalizarmos, temos que nos atentar com um problema: como aceitamos pontos colineares, temos que tratar um caso em que todos os pontos da entrada são colineares, pois isso ocasionaria um caso degenerado em que o algoritmo iria colocar pontos repetidos no fecho convexo. Portanto, torna-se necessário verificar se todos os pontos da entrada são colineares e, caso essa verificação seja verdadeira, iremos considerar como resposta todos os pontos ordenados como mencionado anteriormente só que na ordem inversa, que seria o que o algoritmo de Graham Scan retornaria.

3.3. Complexidade do algoritmo

Falando brevemente sobre a complexidade, note primeiro que as operações nos vetores e as operações de produto vetorial levam tempo constante pare serem executadas. Agora, veja que temos uma ordenação e alguns laços. A ordenação ocorre em $O(n \lg n)$, já o laço que checa o caso degenerado tem complexidade $O(n)$, pois faz uma única iteração por todos os pontos, e o laço que faz a união das partes do fecho convexo também possui complexidade $O(n)$, pois apenas estamos passando pelos pontos da parte inferior e unindo os pontos dela aos pontos da parte superior para formar o fecho convexo. Portanto, o que nos resta é observar os laços que constroem as partes do fecho convexo. Sabemos que eles passam por todos os pontos, o que nos entregaria uma complexidade também de $O(n)$, mas, enquanto estamos no segundo caso, temos que ir corrigindo a parte do fecho convexo até que cheguemos no primeiro caso ou o até que vetor tenha somente um ponto. Como iremos adicionar e remover cada ponto no máximo uma vez, isso nos dará uma complexidade amortizada de $O(n)$. Por consequência, o Monotone Chain terá complexidade de $O(n \lg n)$ ocasionado pela etapa de ordenação.

4. Resolvendo o problema

Para resolvermos o problema, basta guardarmos em cada reino o seu índice e as coordenadas de sua capital e utilizarmos o Monotone Chain para descobrirmos os reinos, com base em suas capitais, que compõem o fecho convexo em $O(n \lg n)$. Após essa etapa, iremos ordenar esses reinos pelos seus índices, também em $O(n \lg n)$, e imprimiremos os índices na resposta em tempo linear, que serão os reinos cujo território é infinito. Consequentemente, nosso algoritmo terá uma complexidade final de $O(n \lg n)$.

5. Conclusão e resultados

Neste artigo foi apresentado uma forma eficiente de solucionar o problema Grande Tratado de Bytelândia da fase sub-regional da Maratona de Programação da SBC 2023.

A solução utilizando o algoritmo descrito acima foi implementada em C++17 e submetida na plataforma Codeforces, no qual recebeu o veredito de aceito. O tempo de execução para os dados casos de teste que tinham entradas de até 10^5 reinos foi de 46 milissegundos, respeitando o limite imposto de 500 milissegundos. Em relação ao custo de memória, a solução submetida utilizou 4,7 megabytes dos 1024 megabytes permitidos.

Referências

De Berg, M. (2000). *Computational geometry: algorithms and applications*. Springer Science & Business Media.