Algoritmo para gerar invólucros convexos

Artur Queiroz - PG38014
Luís Albuquerque - PG38015

December 14, 2019

1 Introdução

Neste trabalho abordamos um dos principais temas de Geometria Computacional, Invólucros Convexos. Estes são muito usados porque têm aplicações em bastantes àreas como reconhecimento de padrões, processamento de imagem, estatística, etc.

Existem várias formas de construir um, mas neste trabalho vamos nos cingir a implementar o "*merge-hull*" Que é um algoritmo que se usa, umas das tecnicas mais importantes na computação, que se chama "Dividir para conquistar". Que se baseia em dividir um problema complexo, em problemas mais pequenos e mais faceis.

2 Descrição

Para implementarmos o algoritmo apenas temos que seguir os passos a baixo.

- Ordenar os pontos por ordem lexicográfica.
- Remover os pontos repetidos.
- Criar o Invólucro convexo
  - Se o conjunto tem apenas 1 elemento, criar um "polígono" de 1 elemento e acaba-se a função.
  - Separar os pontos em dois conjuntos A e B, onde A contém os pontos da esquerda e B os da direita.
  - Calcular o invólucro convexo de A, $\mathcal{A} = \mathrm{I(A)}$ e o de B, $\mathcal{B} = \mathrm{I(B)}$ recursivamente

- Encontrar o pontos de junção.
- Juntar $\mathcal{A}$ e $\mathcal{B}$, calculando o invólucro convexo de A $\cup$ B.

Agora vamos explicar com mais detalhe todo o processo.

2.1 Ordenar os Pontos

Para ordenar os pontos, pode ser usado qualquer algoritmo de ordenação, tendo em atenção que a escolha do algoritmo de ordenação, pode alterar a complexidade do algoritmo como um todo. Nós optamos por escolher o algoritmo de ordenação *merge sort*, além de ter uma das melhores complexidades $\Theta(n \log n)$, achamos que se enquadra perfeitamente no espirito do algoritmo, "Dividir para conquistar".

2.1.1 Descrição de MergeSort

Input: array, indice esquerdo, indice direito

Começando com o indice esquerdo a 0, e o indice direito a (*tamanho do array*) - 1

- Primeiro encontra-se o indice médio do Array e divide-se em dois. (meio = (esquerda + direita)/2)
- Calcular o MergeSort(array, esquerda, meio) (a lista que fica à esquerda)
- Calcular o MergeSort(array, meio+1, direita) (a lista que fica à direita)
- No final junta os dois de forma ordenada.

2.2 Remover os Pontos Repetidos

Já que neste contexto sabemos que o array está ordenado, podemos fazer este algoritmo com 2 indices, um para percorrer o array, o outro para percorrer o novo array onde todos os elementos são diferentes. Se o valor dos dois indices for diferente então incrementa-se o segundo indice e atribui-se o valor do primeiro indice no valor do segundo indice. Se o valor dos dois indices for igual, apenas incrementa-se o primeiro indice.

2.3 Separar os Pontos em dois conjuntos

No trabalho decidimos usar um array, para que a sua divisão a meio tivesse uma complexidade constante é tão simples como encontrar o índice a meio do array.

2.4 Juntar os invólucros convexos

Liberta-se a memória dos vértices entre as duas ligações e de seguida conecta-se as duas ligações.

3 Correção

Depois de mostrarmos como é o algoritmo, aqui vamos provar, porque é que o algoritmo faz o que diz que faz. Tendo I como a nossa função descrita em cima.

Queremos que no fim de execução do algoritmo, o poligono retornado seja o invólucro convexo do conjunto dado no input.

Para isso vamos usar um Corolário dado nas aulas, descrito como:

"Seja $S \subseteq \mathbb{R}^2$ e P um poligono de vértices positivamente orientados $v_0, ..., v_{n-1} \in S$ tal que cada vértice é estritamente convexo e $S \subseteq P$. Então P é o invólucro convexo de S."

E a definição:

"v_i diz-se estritamente convexo se $\mathcal{A}(v_{i-1}, v_i, v_{i+1}) > 0$."

Agora para provar que o nosso algoritmo retorna o invólucro convexo do conjunto dado no input, basta provar que o poligono retornado, com os vértices positivamente orientados, tem todos os vértices estritamente convexos e contém todo o conjunto de pontos.

[Provar para os casos base 0, 1 e 2]

3.1 Todos os vértices são estritamente convexos

Seja S o conjunto de pontos dado no input.

caso indutivo:
Seja $S = S_1 \cup S_2$ tal que todos os pontos de S_1 sejam menores que os pontos de S_2 (lexicograficamente).
Hipoteses Indutivas:

1. $< \forall p_i : p_i \in I(S_1) : \mathcal{A}(p_{i-1}, p_i, p_{i+1}) > 0 >$
2. $< \forall p_i : p_i \in I(S_2) : \mathcal{A}(p_{i-1}, p_i, p_{i+1}) > 0 >$

Na função *find_lower_lim* começa-se com a linha, do maior vértice de $I(S_1)$ ao menor vértice de $I(S_2)$ (lexicograficamente). A partir do vértice atual de $I(S_1)$ segue-se para o anterior, o mais longe possível, até que o vértice anterior esteja estritamente à esquerda da linha, de seguida fazemos o dual para o vértice de $I(S_2)$. Repetindo este processo até não se conseguir mais.

Quando não conseguirmos mais, significa que encontramos 2 vértices p_i e q_i, tais que:

$left(\overline{p_i q_i}, p_{i-1}) \wedge left(\overline{p_i q_i}, q_{i+1})$
$\equiv$ { definição de left }
$\mathcal{A}(p_i, q_i, p_{i-1}) > 0 \wedge \mathcal{A}(p_i, q_i, q_{i+1}) > 0$
$\equiv$ { regra dada na aula }
$\mathcal{A}(p_{i-1}, p_i, q_i) > 0 \wedge \mathcal{A}(p_i, q_i, q_{i+1}) > 0$

A função *find_higher_lim* é análoga à *find_lower_lim*, por isso arranjamos 2 vértices p_i e q_i, tais que:

$left(\overline{q_i p_i}, p_{i+1}) \wedge left(\overline{q_i p_i}, q_{i-1})$
$\equiv$ { definição de left }
$\mathcal{A}(q_i, p_i, p_{i+1}) > 0 \wedge \mathcal{A}(q_i, p_i, q_{i-1}) > 0$
$\equiv$ { regra dada na aula }
$\mathcal{A}(q_i, p_i, p_{i+1}) > 0 \wedge \mathcal{A}(q_{i-1}, q_i, p_i) > 0$

Com isto temos que os vértices encontrados que foram usados para juntar os dois invólucros são estritamente convexos no $I(S)$. Como por indução sabemos que os outros vértices são estritamente convexos, significa que todos os vértices em $I(S)$ são estritamente convexos.

■

3.2 Proposição Tangente

Seja I um invólucro convexo e l uma linha que interseta num vértice de I, p_i, no sentido contrário dos ponteiros do relógio. Seja p_{i-1} o vértice anterior a p_i e p_{i+1} o vértice posterior a p_i. Temos que:

$$leftOn(l, p_{i-1}) \wedge leftOn(l, p_{i+1}) \Rightarrow < \forall p : p \in I : leftOn(l, p) >$$

3.2.1 Demonstração

Suponhamos que $leftOn(l, p_{i-1}) \wedge leftOn(l, p_{i+1})$ e que existe um $p \in I$ tal que $right(l, p)$ isso significa que a linha $\overline{p\ p_i}$ não está no invólucro convexo. Que é uma contradição, ou seja, $< \forall p : p \in I : leftOn(l, p) >$

4 Complexidade

A nossa implementação não foi exatamente igual ao algoritmo original, apesar de não alterar na conta da complexidade assintoticamente. Por isso vamos avaliar a correção da nossa implementação, e quando achamos pertinente, vamos fazer a ressalva, mensionando as diferenças em relação ao algoritmo original.

- Ordenar os pontos pela cordenada x, tem Complexidade $\Theta(n \log n)$
- Separar os pontos em dois conjuntos A e B, onde A contém os pontos da esquerda e B os da direita.
- Calcular o invólucro convexo de A, $\mathcal{A}$ = I(A) e o de B, $\mathcal{B}$ = I(B) recursivamente
- No final juntar $\mathcal{A}$ e $\mathcal{B}$, calculando o invólucro convexo de A $\cup$ B.

5 Conclusão