7. O Algoritmo de Graham

Segundo O'Rourke (cf. [12], pg. 80) talvez o primeiro artigo na área de geometria computacional tenha sido o de Graham [7] que apresenta um algoritmo para encontrar o fecho convexo de um conjunto de pontos no plano.[1]

O algoritmo de Graham constrói o fecho convexo de um conjunto de pontos S em duas fase. A primeira fase consiste de um pré-processamento onde primeiro um ponto extremo p_0 de S é selecionado e depois os pontos restantes são ordenados angularmente ao redor de p_0. Na segunda fase o algoritmo iterativamente processa os vértices produzindo uma seqüência de fechos convexos que converge para $\text{conv}(S)$.

Observação. O material desta seção foi extraído do Capítulo 35 de [3], Capítulo 3 de [6], Capítulo 6 de [10], Capítulo 3 de [14], Capítulo 4 de [4], e Capítulo 3 de [12].

7.1. **Pré-processamento: 'Presorting phase'.** Dado um conjunto de pontos S o pré-processamento do algoritmo de Graham consiste em encontrar um ponto extremo p_0 de S. Escolheremos p_0 como sendo o ponto em S com a menor y-coordenada (mais baixo) e em caso de empate escolheremos, entre os pontos com a menor y-coordenada, o ponto com a maior x-coordenada (mais a direita). Este passo do algoritmo pode ser feito em tempo $O(n)$.

Numa segunda etapa os pontos de $S \setminus \{p_0\}$ são ordenados em relação ao ângulo polar que estes formam ao redor do ponto p_0. Se dois pontos tem o mesmo ângulo polar (i.e. p_0 e esses dois pontos são colineares) então o ponto que está mais próximo de p_0 pode ser descartado. Veja Figura 7.

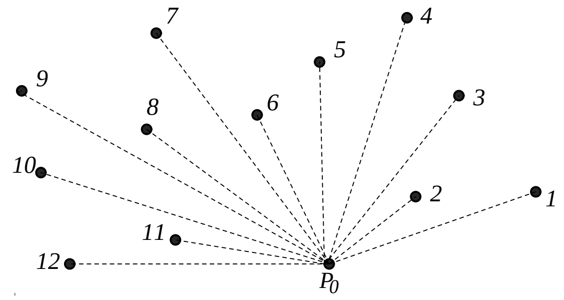

Figura 7. Pontos de S ordenados em relação ao ângulo polar que estes formam ao redor do ponto p_0.

O custo total desta fase de pré-processamento é dominado pelo custo da ordenação dos pontos que é $O(n \log n)$.

7.2. **Construção do fecho convexo: 'The hull finding fase'.** O pré-processamento tem o efeito de simplicar a fase de construção do fecho convexo. Nesta fase os pontos são examinado na ordem que estes foram colocados pelo pré-processamento. Sejam $p_1, p_2, \ldots, p_{n-1}$ os pontos de $S \setminus \{p_0\}$ ordenados em relação ao ângulo polar que esses pontos formam ao redor do

[1] O 'talvez' nesta frase é pelo fato de Toussaint [16] ter encontrado um artigo mais antigo que contém muitas das idéias que apareceram no artigo de Graham [7].

ponto p_0. Seja ainda $S_i := \{p_0, p_1, \ldots, p_i\}$ $(i = 1, \ldots, n-1)$. Durante esta fase cada novo ponto p_i é um ponto extremo de $\text{conv}(S_i)$; entretanto alguns pontos que eram pontos extremos de $\text{conv}(S_{i-1})$ podem não ser pontos extremos de $\text{conv}(S_i)$.

O algoritmo de Graham mantém em cada iteração i o fecho convexo dos ponto S_i. A Figura 8 mostra o algoritmo de Graham em ação.

Considere a iteração que processa o ponto p_7. Devido a ordenação ângular em volta de p_0 é claro que o ponto p_7 é um ponto extremo de $\text{conv}(S_7)$. Mas como saber quais os pontos que eram pontos extremos de $\text{conv}(S_6)$ e não são pontos extremos de $\text{conv}(S_7)$? Para responder a esta pergunta usaremos o fato de que ao percorrermos a fronteira do fecho convexo no sentido anti-horário devemos virar à esquerda em cada ponto extremo. Como indo de p_5 até p_7 passando por p_6 viramos à direita em p_6 sabemos que p_6 não pode ser um ponto extremo de $\text{conv}(S_7)$. Analogamente, o ponto p_5 não é um ponto extremo de $\text{conv}(S_7)$ pois indo de p_4 até p_5 e depois de p_5 até p_7 viramos à direita em p_5. Logo, na iteração correspondente ao ponto p_7 os pontos p_6, p_5 e p_4 serão removidos da lista de pontos extremos de $\text{conv}(S_7)$ e o ponto p_7 será incluído nesta lista. Com isto completamos mais uma iteração do algoritmo.

Note que na descrição da iteração do algoritmo nos percorremos a lista de vértices pertencentes ao fecho convexo corrente do final para o começo (considerando que p_0 é o começo da lista). Por isto, ao descrevermos o algoritmo usaremos uma pilha T para guardar os pontos do fecho convexo corrente. Depois da última iteração do algoritmo a pilha T contém os vértices extremos de $\text{conv}(S)$ em ordem anti-horária — a medida que eles foram encontrados ao percorrermos a fronteira de $\text{conv}(S)$. Abaixo encontra-se uma decrição de todo o algoritmo.

Algoritmo `Graham-Scan`(S)
`Entrada:` Um conjunto S com $n > 3$ pontos do plano.
`Saída:` O fecho convexo dos pontos em S.
1. Seja p_0 o ponto em S com a menor y-coordenada, em caso de empate tome o ponto mais a direita. [custo $O(n)$]
2. Seja $p_1, \ldots, p_{n-1}$ os pontos de $S \setminus p_0$ ordenados em relação ao ângulo polar que estes pontos formam ao redor do ponto p_0. Se mais de um ponto tem o mesmo ângulo, descarte todos menos o que esta mais distânte de p_0. [custo $O(n \log n)$]
3. $T \leftarrow \emptyset$.
4. Empilhe p_0, p_1 e p_2, nesta ordem, na pilha T.
5. $i \leftarrow 3$.
6. **while** $i < n$ **do**
   Seja p_t o ponto que está no topo de T e p_{t-1} o ponto que está embaixo de p_t em T.
7.   **if** p_i está à esquerda de (p_{t-1}, p_t)
8.   **then** Empilhe p_i em T e $i \leftarrow i + 1$
9.   **else** Desempilhe p_t de T.
10. **return** T.

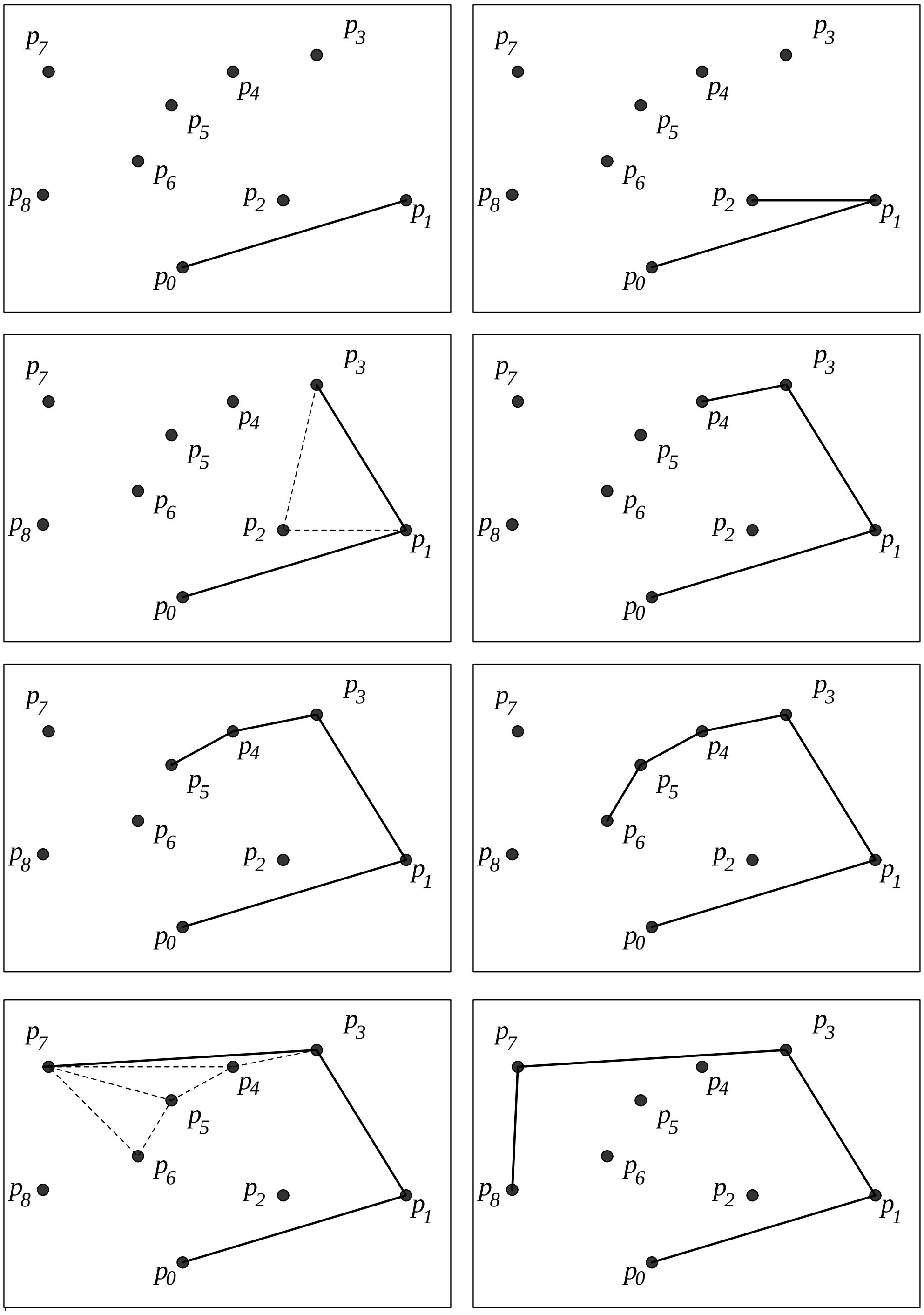

FIGURA 8. O Algoritmo de Graham em ação.

7.3. **Análise do algoritmo.** A complexidade de tempo do Algoritmo de Graham é $O(n \log n)$, onde n é o número de pontos do conjunto S.

A Linha 1 do algoritmo tem um custo de $O(n)$. A ordenação da Linha 2 pode ser feita em $O(n \log n)$ se usarmos o Algoritmo `Heapsort` ou `Mergesort`. Logo a fase de pré-procesamento tem um custo total de $O(n \log n)$.

As Linhas 3, 4 e 5 são executadas em tempo constante ($O(1)$).

O número de operações Desempilha é menor do que o número de operações Empilha (já que p_0, p_1 e p_{n-1} nunca serão desempilhados de T). Como o número total de operações Empilha executadas durante todo o algoritmo será n então o número de operações Desempilha realizadas será menor que n. Logo, o tempo total gasto com as operações Empilha e Desempilha ao longo da execução do algoritmo é $O(n)$ (já que cada uma destas operações pode ser feita em tempo constante). Isto também implica que o número total de execuções do comando **while** da Linha 6 será $O(n)$ (mais precisamente, no máximo $2n-3$).

Assim, concluímos que o tempo total gasto pelo algoritmo para encontrar o fecho convexo de um conjunto com n pontos é $O(n \log n)$. Como veremos mais adiante este algoritmo é ótimo.

7.4. **Corretude do algoritmo.** Notemos primeiro que p_0, como foi escolhido, é um ponto extremo de $\mathrm{conv}(S)$. Observemos ainda que se q é um ponto de S então qualquer ponto em $S \setminus \{p_0, q\}$ que pertença ao segmento p_0q não é um ponto extremo de $\mathrm{conv}(S)$ (pois este ponto seria combinação convexa dos pontos q e p_0). Estas observações justificam a Linha 1 e 2 do algoritmo.

Seja $S_i := \{p_0, p_1, \dots, p_i\}$ ($i = 2, \dots, n-1$). Ao final da execução da Linha 4 teremos que a pilha T contém os pontos extremos de $\mathrm{conv}(S_2)$. A corretude do algoritmo é baseada no seguinte teorema.

Teorema 1 (Invariante 1)**.** *Imediatamente após qualquer atualização da variável i (Linha 5 e Linha 6) temos que os pontos que estão na pilha T são pontos extremos de conv(S_{i-1}) em ordem anti-horária.*

Note que o teorema acima implica imediatamente a corretude do algoritmo. De fato, quando a execução do algoritmo chega na Linha 10 temos que $i = n$ e teremos portanto que S contém, em ordem anti-horária, os vértices extremos de $\mathrm{conv}(S_{n-1}) = \mathrm{conv}(S)$. Para provarmos este teorema faremos uso do seguinte lema.

Lema 2 (Invariante 2)**.** *O algoritmo de Graham mantém o invariante de que os pontos na pilha T são os vértices, em ordem anti-horária, de um polígono convexo.*

Prova. O lema é verdadeiro logo após a execução da Linha 4 do algoritmo, já que p_0, p_1 e p_2 são vértices de um polígono convexo. Se os pontos em T são os vértices em ordem anti-horária de um polígono convexo então após desempinharmos um ponto do topo de T teremos que os pontos restantes satisfazem a mesma propriedade. Logo, desempilhar um vértice preserva o invariante. (Isto corresponde a cortar uma orelha do polígono.)

Antes de examinarmos o que ocorre quando o algoritmo empilha um novo ponto em T examinaremos uma propriedade geometrica. (Veja a Figura 9.) Seja P um polígono convexo e escolha uma aresta qualquer p_rp_t de P. Considere a região limitada por p_rp_t e pelas extensões das duas arestas adjacentes.

Se nós adicionarmos um ponto p_s à esta região e trocarmos a aresta $p_r p_t$ de P pelas arestas $p_r p_s$ e $p_s p_t$ o polígono resultante é convexo.

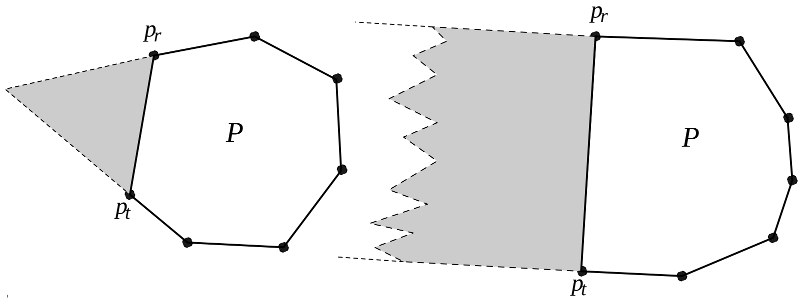

FIGURA 9. Colocar um vértice na região sombreada faz com que o novo polígono resultante continue a ser convexo.

Agora consideremos o ponto p_i que é empilhado em T. Seja p_t o vértice no topo de T antes de empilharmos p_i e seja p_{t-1} o predecessor de p_t em T. Afirmamos que p_i deve estar na região sombreada ilustrada na Figura 9. De fato, como estamos examinando os pontos em ordem crescente do ângulo polar que eles formam com p_0 temos que p_i está à esquerda de (p_0, p_t) e pela escolha de p_0 sabemos que p_i também está à esquerda de (p_0, p_1). Além disso, como p_i foi empilhado em S temos, pela comparação da Linha 6 do algoritmo, que p_i está à direita de (p_{t-1}, p_t). Logo, p_i está na região sombreada da figura. Com isto completamos a prova do lema. ∎

Prova do Teorema 1. Seja k o valor que acaba de ser recebido pela variável i. Quando $k = 3$ (atualização feita pela Linha 5) temos que T contém p_0, p_1 e p_2 que são os pontos extremos de $\text{conv}(S_2)$. (Observe que o fato destes pontos serem pontos extremos seguem da execução das Linhas 1 e 2 do algoritmo.)

Suponha agora que $k > 3$ e que para valores de i menores do que k o teorema vale. E considere a execução da Linhas 6, 7, 8 e 9 do algoritmo enquanto a variável i tem o valor $k-1$. Mostraremos primeiro que os pontos desempilhados de S na Linha 9 não são pontos extremos de $\text{conv}(S_{k-1})$.

Sejam p_t e p_{t-1} os pontos no topo da pilha T imediatamente antes do ponto p_t ser desempilhado durante a execução da Linha 9. Como p_t foi desempilhado temos então que p_i está à esquerda de (p_{t-1}, p_t); ou seja, viramos à direita no ponto p_t quando estavamos indo de p_{t-1} até p_i passando por p_t. Como estamos processando os pontos em ordem crescente do ângulo polar que eles formam com p_0 temos que o ponto p_t pertence ao triângulo $\triangle p_0 p_{t-1} p_i$ e portanto não pode ser um ponto extremo de $\text{conv}(S_{k-1})$. Veja Figura 10.

Agora, pelo Lema 2, temos que quando p_i é empilhado em T (imediatamente antes de atualizarmos i) os pontos em T são vértices de um polígono convexo. Como T só contém pontos de S_{k-1} e esses pontos são vértices de um polígono convexo temos que quanto i recebe o valor k os pontos em T são todos os pontos extremos de S_{k-1}. ∎

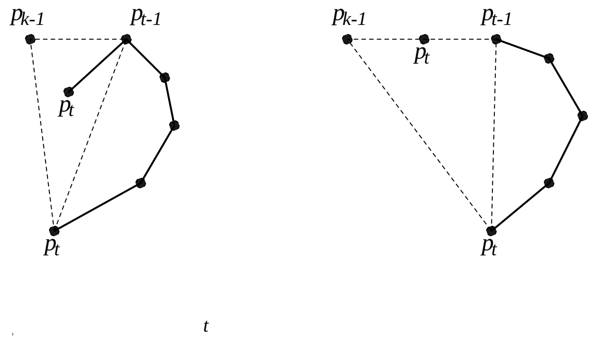

FIGURA 10. Em ambas as situações ilustradas acima o ponto p_t não é ponto extremo de $\text{conv}(S_{k-1})$.