8. Um algoritmo incremental

Depois de termos visto um algoritmo que constrói o fecho convexo de um conjunto de n pontos em tempo $O(n \log n)$ pode parecer sem sentido continuarmos explorando este problema. Entretanto, existem pelo menos duas motivações para continuarmos: extensão do problema para o espaço tri-dimensional (ou dimensão maior) e o estudo de outros métodos em geometria computacional. O Algoritmo de Graham não tem nenhuma extensão natural para o problema tri-dimensional: o algoritmo depende fortemente da ordenação angular, que não tem (ou não se conhece!?) nenhuma contraparte tri-dimemsional. O algoritmo que veremos nesta seção tem uma extensão natural para o problema tri-dimensional.

Um *algoritmo incremental* examina os dados do problema, um item por vez, enquanto uma solução corrente dos itens que já foram vistos até o momento é mantida. Em cada iteração, o próximo item é examinado e processado e a solução corrente para o problema é atualizada para satisfazer este novo item. (Lembre-se dos algoritmo de ordenação por inserção: inserção direta e inserção binária.)

Observação. O material desta seção foi extraído do Capítulo 35 de [3], Capítulo 6 de [10], Capítulo 3 de [11] Capítulo 3 de [14], Capítulo 4 de [4], e Capítulo 3 de [12].

8.1. **O algoritmo.** A idéia básica do algoritmo é: examinar um ponto por vez, a cada passo construir o fecho convexo dos pontos já examinados, e usar esse fecho convexo no exame do próximo ponto. Dividir o problema desta maneira simplifica bastante o trabalho: a cada passo precisamos incluir um único ponto no fecho convexo do pontos já examinados.

Seja $S := \{p_0, p_1, \dots, p_{n-1}\}$ um conjunto de pontos que não contém três pontos colineares, i.e. o conjunto de pontos está em posição geral (para facilitar a exposição).

Algoritmo `Incremental`
`Entrada:` Um conjunto de pontos $\{p_0, p_1, \dots, p_{n-1}\}$.
`Saída:` O fecho convexo do conjunto S.
1. $P_2 \leftarrow \text{conv}(\{p_0, p_1, p_2\})$.
2. **for** $k = 3, \dots, n-1$ **do**
3. $\quad P_k \leftarrow \text{conv}(P_{k-1} \cup \{p_k\})$.
4. **return** P_{n-1}.

Como mostra o algoritmo acima, o primeiro fecho convexo é o triângulo $\text{conv}(\{p_0, p_1, p_2\})$. Seja $Q := P_{k-1}$ e $p = p_k$. O problema de construir $\text{conv}(Q \cup \{p\})$ (Linha 3 do algoritmo) cai em um dos dois casos a seguir:

1. $p \in Q$. Uma vez que foi determinado que $p \in Q$, o ponto p pode ser descartado e sabemos que $\text{conv}(Q \cup \{p\}) = Q$. Para decidir se $p \in Q$ podemos usar a função (primitiva) `Lefton`. Este teste pode ser feito em tempo $O(n)$ (como vocês fizeram no Exercício 4 da Lista 3): $p \in Q$ se e somente se está à esquerda ou sobre cada aresta (orientada) do polígono Q.

2. $p \notin Q$. Se em algum momento p está à direita de uma aresta de Q então nós temos que construir $\text{conv}(Q \cup \{p\})$. O que faz esta tarefa mais fácil é que nós temos que encontrar somente as duas retas que passam pelo ponto p e que são tangentes ao polígono Q (como é ilustrado na Figura 11) e modificar o fecho convexo de maneira conveniente. Pela nossa hipótese de posição geral cada reta que passa por p e tangência Q tem um único ponto de intersecção com Q: um vértice p_i do polígono Q.

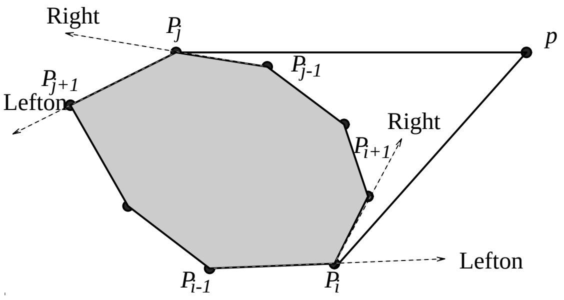

FIGURA 11. Retas tangentes a Q que passam por p; 'Lefton' significa que o ponto p está a esquerda ou é colinear ao vetor indicado, e 'Right' que p está à direita.

Examinando a Figura 11 nós vemos que é possível usarmos o resultado da primitiva `Lefton` para encontrar o ponto de tangência mais baixo p_i. Note que p esta à esquerda de (p_{i-1}, p_i) e à direita de (p_i, p_{i+1}). Para o ponto de tangência mais alto o raciocinio é análogo. Logo os dois vértices de tangência podem ser encontrados através da mesma série de testes `Lefton` usados para decidir se $p \in Q$. Logo o caso $p \notin Q$ também pode ser tratado em tempo $O(n)$.

Algoritmo `Vértices-Tangentes`(Q, p)
`Entrada:` Um polígono convexo Q com m vértices e um ponto p tal que $p \notin Q$.
`Saída:` Os dois vértices de tangência de Q em relação a p.
1. **for** $i = 0, 1, \ldots, m-1$ **do**
2. **if** `Lefton`$(p_{i-1}, p_i, p) \neq$ `Lefton`(p_i, p_{i+1}, p)
3. **then** p_i é vértice de tangência.

Precisamos agora só especificar como construir o novo polígono a partir do polígono Q, do ponto p e dos pontos de tangência (digamos) p_i e p_j. Através da Figura 11 nós vemos que o fecho (polígono) convexo corrente será

$$(p_0, p_1, \ldots, p_{i-1}, p_i, p, p_j, p_{j+1}, \ldots, p_{n-1}).$$

Se os vértices extremos do fecho convexo (em ordem anti-horária) estiverem representados por uma lista circular duplamente ligada então a atualização do fecho poderá ser feita através de uma simples seqüência de remoções e uma inserção.

8.2. **Análise do algoritmo.** Podemos decidir em tempo $O(n)$ se $p \in Q$. No caso de p não ser um ponto em Q os vértices de tangância podem ser encontrados em tempo $O(n)$. O número de remoções da lista ligada circular que armazena os vértices extremos de Q pode ser atualizada através de operações de remoções e inserções. Como o número total de remoções durante a execução de todo o algoritmo é menor que o número total de inserções que por sua vez é igual a n, temos que todas as inserções e remoções podem ser feitas em tempo $O(n)$. Portanto a complexidade de tempo do Algoritmo `Incremental` é $O(n^2)$.

Com um pouco de mais de esforço a complexidade de tempo do Algoritmo `Incremental` pode ser reduzida para $O(n \log n)$.