MAC 5747 – Geometria Computacional
Prof. José Coelho de Pina
Monitor: Fábio Henrique Viduani Martinez

Lista de Exercícios 7 – Respostas

1. (Exercício 3.4.1.1 de [1] — `Embrulho-para-Presente`: melhor caso) Encontre o melhor caso para o Algoritm `Embrulho-para-Presente`: conjuntos de n pontos no plano tal que a complexidade de tempo assintótica do algoritmo seja o menor possível em função de n. Qual é esta complexidade de tempo?

   O melhor caso para o algoritmo `Embrulho-para-Presente` é aquele em que o conjunto S de n pontos de entrada tem $\text{conv}(S) = (p_i, p_j, p_k)$, para $p_i, p_j, p_k \in S$, isto é, o fecho convexo de S é um triângulo. Neste caso o tempo de execução $T(n)$ do algoritmo pode ser descrito

$$T(n) = \underbrace{O(n)}_{(1)} + \underbrace{O(n) + O(n) + O(n)}_{(2)} = O(n)$$

   onde (1) é devido ao pré-processamento e (2) às três iterações realizadas no laço do algoritmo.

3. (Exercício 3.4.1.4 de [1] — `QuickHull`: pior caso) Construa um conjunto de pontos genérico que força o Algoritmo `QuickHull` a ter um comportamente quadrático. Por 'genérico' (como sempre) queremos dizer uma construção que seja o pior caso para o `QuickHull` para valores arbitrários de n.

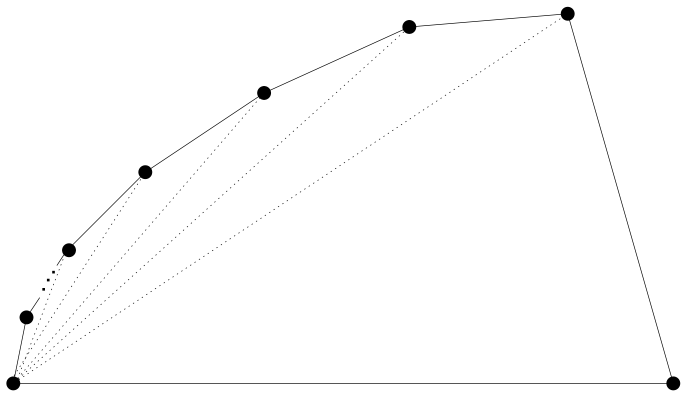
   Para que o algoritmo `QuickHull` tenha comportamento quadrático teqmos que tomar um conjunto do pontos que faz com que as chamadas recursivas ao `QuickHull` sejam feitas de forma que o pior caso sempre aconteça. Isto é, podemos, por exemplo, tomar um conjunto de pontos tal que para qualquer chamada recursiva temos

$$\alpha = |S_1| = 1, \beta = |S_2| = n - 1$$

   e então

$$T(n) = T(1) + T(n-1) + cn = T(n-1) + c'n = O(n^2).$$

   Um conjunto como este é apresentado na figura a seguir.



7. (Exercício 3.5.6.1 de [1] — `Graham-Scan`: pior caso). Construa um conjunto de pontos para o qual o laço **while** da Linha 6 do algoritmo de Graham faz o maior número de iterações.

   Mostramos na figura a seguir, duas possibilidades de conjuntos de pontos em que o pior caso para o algoritmo `Graham-Scan` acontece.

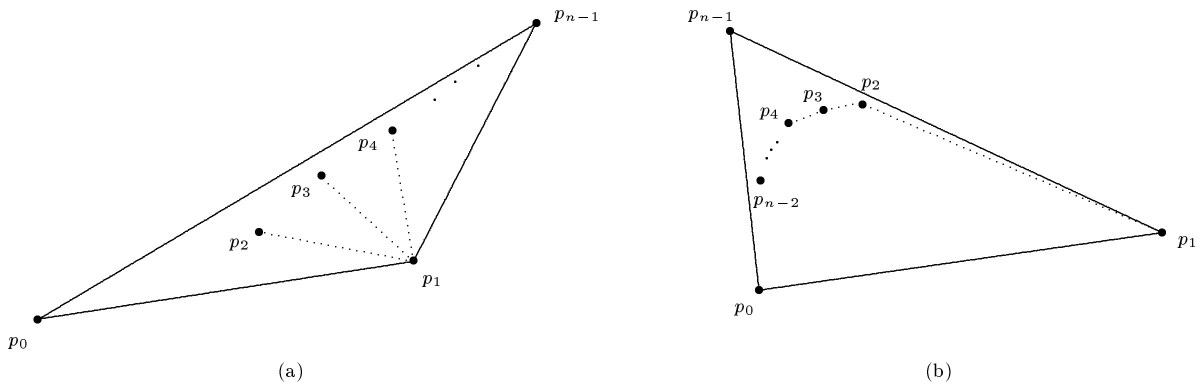


Figura 1: (a) Neste caso, a pilha T vai conter em sua base os vértices p_0 e p_1 e sofrerá operações Empilha e Desempilha para os vértices subseqüentes $p_2, p_3, \ldots, p_{n-2}$; no final T conterá apenas p_0, p_1 e p_{n-1}. (b) A pilha T, neste exemplo, conterá em um dado momento os elementos $p_0, p_1, p_2, \ldots, p_{n-2}$ quando deverá desempilhar todos os elementos $p_{n-2}, p_{n-3}, \ldots, p_3, p_2$; no final, T também conterá apenas p_0, p_1, p_{n-1}.

Note que o total de operações realizadas nestes dois casos é $2n - 3$.

8. (Polígono passando por conjunto de pontos) Descreva um algoritmo de complexidade de tempo $O(n \log n)$ que dado um conjunto S de n pontos no plano contrói um polígono que tem os pontos em S como vértices.

   O algoritmo é baseado na fase de pré-processamento do algoritmo `Graham-Scan`.

   Algoritmo `Polígono`(S)
   `Entrada:` Um conjunto S com $n > 3$ pontos do plano.
   `Saída:` Um polígono passando pelo conjunto de pontos S.

   1. Seja p_0 o ponto em S com a menor y-coordenada; em caso de empate, tome o ponto mais a direita.
   2. Sejam $p_1, \ldots, p_{n-1}$ os pontos de $S \setminus \{p_0\}$ ordenados em relação ao ângulo polar que estes formam ao redor do ponto p_0. Se mais de um ponto tem o mesmo ângulo, digamos $p_i, p_{i+1}, \ldots, p_j$, então $\text{dist}(p_0, p_i) > \text{dist}(p_0, p_{i+1}) > \ldots > \text{dist}(p_0, p_j)$.
   3. Devolva $\langle p_0, p_1, \ldots, p_{n-1} \rangle$.

   O algoritmo `Polígono` tem tempo de execução $O(n \log n)$.

9. (Algoritmo `Incremental` ótimo) Modifique o Algoritmo `Incremental` visto em sala de aula de tal forma que a sua complexidade de tempo seja reduzida para $O(n \log n)$.

   [**Sugestão.** Em um pré-processamento ordene os pontos de S em relação a x-coordenada de tal forma que $p \notin Q$ em cada passo. Examine os pontos nesta ordem.]

   Seguindo a sugestão dada, na fase de pré-processamento onde ordenamos os pontos de S em relação à x-coordenada, o algoritmo tem tempo de execução $O(n \log n)$.

   Para que esta modificação tenha o efeito desejado, isto é, que o algoritmo `Incrmental` tenha tempo de execução $O(n \log n)$ o restante do algoritmo deve ter tempo no máximo $O(n \log n)$. Usaremos um argumento de análise amortizada para mostrar que esta fase do algoritmo tem tempo de execução $O(n)$.

Observe que duas operações são realizadas, durante a execução do algoritmo, sobre cada ponto p de $\{p_0, p_1, \ldots, p_{n-1}\}$, a saber:

(a) Consulta: verifica se p está à esquerda de $p_{i-1}p_i$ e à direita de p_ip_{i+1} ou, ao contrário, se p está à direita de $p_{i-1}p_i$ e à esquerda de p_ip_{i+1};

(b) Aresta: traça uma 'aresta' de P_k entre um ponto marcado p_i de P_k e p.

Faremos agora uma atribuição de custos amortizados a estas operações:

Operação	Custo Real	Custo Amortizado
Consulta	1	0
Aresta	1	2

O custo real para a execução de qualquer uma destas operações é constante. Já o custo amortizado para a operação de traçar uma aresta é sempre 2: uma unidade é utilizada para "pagar" por esta operação e outra unidade é "deixada sobre o ponto", para pagamento de uma operação futura. No início, os pontos p_0, p_1 e p_2 têm 1 crétdito cada um. A medida que outros pontos vão sendo computados incrementalmente, outras operações vão sendo realizadas, mas sempre há crédito suficiente em cada ponto para que operações possam ser realizadas e o custo amortizado seja sempre ≥ 0.

Dessa forma, como temos n operações Aresta no pior caso, o custo total amortizado é $O(n)$. Finalmente, como o custo total amortizado é um limitante superior do custo total real, o custo total real desta fase do algoritmo é $O(n)$.

Portanto, o tempo de execução do algoritmo `Incremental` com esta modificação é $O(n \log n)$.