Universidade Federal do Rio de Janeiro - PESC/COPPE
Disciplina: (COS-841, MAB-704) - Complexidade de Algoritmos - 2023/3
Professores: Celina Miraglia, Fábio Botler e Franklin Marquezino
Monitores: Diego Amaro, Guilherme Bridi e Oscar Martins

Gabarito Lista 1

---

1 (Relações de recorrência). Para cada uma das relações de recorrência abaixo, caso seja possível, aplique o teorema mestre; caso contrário, explique o porquê da impossibilidade. (2,5 pontos)

a) $T(n) = 3T(n/2) + n^2$

Resolução:

Inicialmente, vamos identificar as constantes e a função $f(n)$ para analisar em qual caso o Teorema Mestre se aplica. Note que:

$$T(n) = \underbrace{3}_{a} T(n/\underbrace{2}_{b}) + \underbrace{n^2}_{f(n)},$$

Note que $n^{\log_b a} = n^{\log_2 3}$.

Como $n^2 > n^{\log_2 3}$ para todo $n > 1$, $n^{\log_b a}$ é um limite assintótico inferior para $f(n)$, o que nos faz concluir que a complexidade da recorrência $T(n)$ tem estrutura encaixada no caso do 3 do Teorema Mestre.

Evidentemente, $n^2 \leq n^2$, logo, pode-se concluir que $n^2 = f(n) \in \Omega(n^2) = \Omega(n^{\log_2 4}) = \Omega(n^{\log_2 3+\epsilon})$ com $\epsilon = 1$.

Além disso, $\underbrace{3}_{a} . \underbrace{\left(\frac{n}{2}\right)^2}_{f(\frac{n}{b})} = \frac{3n^2}{4} = \frac{3}{4}n^2,$

portanto, a função $f(n) = n^2$ satisfaz a condição de regularidade para qualquer constante c tal que $0,75 \leq c < 1$.

Portanto, pelo caso 3 do Teorema Mestre, concluímos que $T(n) = \Theta(f(n)) = \Theta(n^2)$.

b) $T(n) = 2T(n/2) + n \log n$

Resolução:

Inicialmente, vamos identificar as constantes e a função $f(n)$ para analisar em qual caso o Teorema Mestre se aplica. Note que:

$$T(n) = \underbrace{2}_{a} T(n/\underbrace{2}_{b}) + \underbrace{n \log n}_{f(n)},$$

Note que $n^{\log_b a} = n^{\log_2 2} = n$. Como $n \log n > n$ para todo $n > e$, $n^{\log_b a}$ é um limite assintótico inferior para $f(n)$, o que nos faz concluir que a complexidade da recorrência $T(n)$ só pode ser avaliada no caso do 3 do Teorema Mestre.

Já concluímos que $\log n > 1$, $\forall n > e$, assim $f(n) = n \log n \in \Omega(n) = \Omega(n^{\log_2 2})$. Resta provar que $f(n) = n \log n \in \Omega(n^{\log_2 2+\epsilon})$ para algum $\epsilon > 0$ e em seguida mostrar a condição de regularidade.

Vamos assumir que existe um ϵ conforme descrito acima. Note que a condição de regularidade não seria satisfeita nesse caso. Repare:

$$a.f(n) = 2.\frac{n}{2} \log\left(\frac{n}{2}\right) = n \log\left(\frac{n}{2}\right) = n[\log(n) - \log(2)].$$

Para satisfazer a condição de regularidade, deveríamos encontrar um c, com $0 < c < 1$ tal que

$$n[\log(n) - \log(2)] \leq cn\log(n) \implies \frac{n[\log(n) - \log(2)]}{cn\log(n)} \leq 1,$$

para todo n suficientemente grande.

No entanto, para qualquer c, com $0 < c < 1$, $n[\log(n) - \log(2)]$ é assintoticamente maior do que $cn\log(n)$. Mostraremos esse fato, provando que para todo $c < 1$ positivo

$$\frac{n[\log(n) - \log(2)]}{cn\log(n)} > 1.$$

Com efeito,

$$\begin{aligned}
\lim_{n\to\infty} \frac{n[\log(n) - \log(2)]}{cn\log(n)} &= \lim_{n\to\infty} \frac{n\log(n) - n\log(2)}{cn\log(n)} \\
&= \lim_{n\to\infty} \frac{1\log(n) + \cancel{n}\overbrace{\frac{1}{\cancel{n}}}^{1} - \log(2)}{c(1\log(n) + \underbrace{\cancel{n}\frac{1}{\cancel{n}}}_{1})} && \text{(Regra de L'Hôpital)} \\
&= \lim_{n\to\infty} \frac{\log(n) + 1 - \log(2)}{c\log(n) + c} \\
&= \lim_{n\to\infty} \frac{\frac{1}{n}}{c\frac{1}{n}} && \text{(Regra de L'Hôpital)} \\
&= \lim_{n\to\infty} \frac{1}{c} > 1. && \text{(Pois } c \text{ é menor que 1)}
\end{aligned}$$

portanto, a função $f(n) = n\log n$ não satisfaz a condição de regularidade para nenhuma constante $c < 1$.

Outra maneira de argumentar seria justificar que $f(n) = n\log(n)$ não é polinomialmente maior que $n^{1+\varepsilon}$.

c) $T(n) = 4T(n/2) + \dfrac{n}{\log n}$

Resolução:

Inicialmente, vamos identificar as constantes e a função $f(n)$ para analisar em qual caso o Teorema Mestre se aplica. Note que:

$$T(n) = \underbrace{4}_{a} T(n/\underbrace{2}_{b}) + \underbrace{\frac{n}{\log n}}_{f(n)},$$

Note que $n^{\log_b a} = n^{\log_2 4} = n^2$. Além disso,

$$\log n = \log_e n > 1, \forall n > \underbrace{e}_{\text{base}} \implies \frac{n}{\log n} \leq n, \forall n > e.$$

Como $n^2 > n \geq \dfrac{n}{\log n}$ para todo $n > e$, $n^{\log_b a}$ é um limite assintótico superior para $f(n)$, o que nos faz concluir que a complexidade da recorrência $T(n)$ está estruturado conforme o caso do 1 do Teorema Mestre.
Já mostramos que $n \geq \dfrac{n}{\log n}$, portanto,

$$\frac{n}{\log n} = f(n) \in O(n) = O(n^{\log_2 2}) = O(n^{\log_2 4 - \epsilon}),$$

com $\epsilon = 2$.
Portanto, pelo caso 1 do Teorema Mestre, concluímos que $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

(2) Verdadeiro ou falso: para todo $k > 0$, $\log^k n = O(\sqrt{n})$, onde $\log^k n = (\log n)^k$? Justifique devidamente a tua resposta. (2,5 pontos)

Resolução: (Utilizando propriedades de log.) Verdadeiro.

$$\begin{aligned}
\log^k(n) &= (\log(n))^k \\
&= \left(2k\left(\tfrac{1}{2k}\right)\log(n)\right)^k \\
&= \left(2k\log(n)^{1/2k}\right)^k \\
&\leq \left(2kn^{1/2k}\right)^k \\
&= \underbrace{(2k)^k}_{C}(n^{1/2}) \\
&= C\sqrt{n}
\end{aligned}$$

Resolução: (Utilizando a regra de L'Hopital.) Verdadeiro.

$$\lim_{n\to\infty}\frac{\log^k(n)}{\sqrt{n}} = \lim_{n\to\infty}\frac{k\log^{k-1}(n)\frac{1}{n}}{\frac{1}{2\sqrt{n}}} \qquad \text{(Regra de L'Hôpital e da cadeia no numerador)}$$

$$= \lim_{n\to\infty}\frac{2\cancel{\sqrt{n}}k\log^{k-1}(n)}{\underbrace{\cancel{n}}_{\sqrt{n}}}$$

$$= \lim_{n\to\infty}\frac{2k\log^{k-1}(n)}{\sqrt{n}}$$

$$= \lim_{n\to\infty}\frac{2k(k-1)\log^{k-2}(n)\frac{1}{n}}{\frac{1}{2\sqrt{n}}} \qquad \text{(Regra de L'Hôpital e da cadeia)}$$

$$= \lim_{n\to\infty}\frac{2.2\cancel{\sqrt{n}}k(k-1)\log^{k-2}(n)}{\underbrace{\cancel{n}}_{\sqrt{n}}}$$

$$= \lim_{n\to\infty}\frac{2^2k(k-1)\log^{k-2}(n)}{\sqrt{n}}$$

$$= \ldots \qquad \text{(Sucessivas aplicações de L'Hôpital)}$$

$$= \lim_{n\to\infty}\frac{2^{(k-1)}k(k-1)...(3)(2)\log(n)}{\sqrt{n}}$$

$$= \lim_{n\to\infty}\frac{2^{(k-1)}k(k-1)...(3)(2)\frac{1}{n}}{\frac{1}{2\sqrt{n}}} \qquad \text{(Regra de L'Hôpital)}$$

$$= \lim_{n\to\infty}\frac{2^k k!}{\sqrt{n}} = 0$$

Como encontramos valor 0 para o limite, fica demonstrado que o crescimento do denominador é mais acelerado que o do numerador. Formalmente, Assim, fica provado que para qualquer k pré-fixado, a função $\sqrt{n}$ limita superiormente o crescimento do tempo de execução de $\log^k n$ para valores suficientemente grandes de n, assim $\log^k n = O(\sqrt{n})$, para todo $k > 0$.

Ao demonstrarmos esse resultado, assumimos que a função $\log(x)$ está definida na base neperiana, isto é, $\log(x) = \log_e(x) = \ln(x)$.

Para quem interpretou $\log(x)$ considerando 10 como base do logaritmo, a construção é análoga à acima. A única diferença está na derivada desta função. A saber, basta aplicarmos a propriedade de mudança de base de logaritmo:

$$\frac{d}{dx}(\log_{10}(x)) = \frac{d}{dx}\left(\frac{\ln(x)}{\ln(10)}\right) = \frac{1}{\ln(10)}\underbrace{\frac{d}{dx}(\ln(x))}_{\frac{1}{x}} = \frac{1}{x\ln(10)}.$$

Ou seja, para cada derivação de $\log(x)$, deveríamos acrescentar a constante multiplicativa $\frac{1}{\ln(10)}$. Ao final de todos os passos, obteríamos o mesmo resultado para o limite, uma vez que a constante multiplicativa não alteraria a convergência para zero:

$$\lim_{n\to\infty} \frac{\log^k(n)}{\sqrt{n}} = \lim_{n\to\infty} \frac{k\log^{k-1}(n)\frac{1}{n\ln(10)}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2k\log^{k-1}(n)}{\ln(10)\sqrt{n}} = \cdots = \lim_{n\to\infty} \frac{2^k k!}{\ln^k(10)\sqrt{n}} = 0.$$

3 (Dividir para conquistar). Resolva o problema de encontrar um par de pontos mais próximos no $\mathbb{R}^2$(2,5 pontos).

Dados n pontos em um espaço métrico, encontrar um par de pontos com a menor distância dentre as distâncias entre quaisquer dois destes pontos.

Resolução: Para resolver o problema faremos uso da técnica de dividir e conquistar.

A distância entre dois pontos do plano é usualmente definida a partir da métrica euclidiana, definida abaixo:

$$\begin{array}{rccl} d: & \mathbb{R}^2 \times \mathbb{R}^2 & \longrightarrow & \mathbb{R} \\ & (\underbrace{A}_{(x_1,y_1)}, \underbrace{B}_{(x_2,y_2)}) & \mapsto & d(A,B) = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} \end{array}$$

Passo 1) Para construir o algoritmo, considere $L = [p_1, p_2, ..., p_n]$ uma lista formada pelos n pontos do plano $\mathbb{R}^2$, onde cada ponto p_i é da forma $p_i = (p_{xi}, p_{yi})$, isto é, tem coordenadas p_{xi} e p_{yi}.

Passo 2) Em seguida, estabelecemos uma ordenação para os pontos de L. Como $\mathbb{R}^2$ não é um conjunto ordenado, determinamos que p_i precede p_j se, e somente se, $p_{xi} \leq p_{xj}$, isto é, ordenamos a lista a partir da primeira coordenada de cada ponto.

Passo 3) No próximo passo, encontramos p_M, o ponto médio de L (com base na ordenação estabelecida acima), e "repartirmos"a lista em duas sub-listas, que chamaremos de L_{esq} e L_{dir}. Elas representam a lista de pontos que precedem p_M e a lista de pontos que são precedidos por p_M.

Passo 4) Em cada sub-lista, obtemos a menor distância e salvamos o par de pontos mais próximo em L_{esq} e L_{dir}. Definimos d_{min} o minimo entre a menor distância em L_{esq} e a menor distância em L_{dir} e salvamos o par de pontos que determina essa distância.

Passo 5) Para atualizar o valor de distância mínima em L devemos analisar as distâncias formadas por pontos que pertencem a diferentes partições de L, ou seja, as distâncias entre pares formados por um ponto em L_{esq} e um ponto em L_{dir}. Para isso, é suficiente utilizar o valor de d_{min} como uma cota superior para distância mínima. Assim, formamos a lista Y, definida pelos pontos que se encontram em uma faixa horizontal de amplitude $2d_{min}$ ao ponto médio de L, p_M.

Passo 6) Estabelecemos uma ordenação para Y em função da ordem crescente das segundas coordenadas de seus pontos. Para maior organização, renomeamos esses pontos como q_i, $i \in \{1, ..., |Y|\}$.

Passo 7) Por fim, compare as distâncias em Y com a menor distância encontrada até o momento. Retornamos o par de pontos com a menor distância em L,

Segue abaixo o algoritmo:

Algoritmo 1: Par de pontos mais próximos no plano.

Dados: $L = [p_1, p_2, ..., p_n] <$ uma lista L com n pontos do plano $\mathbb{R}^2$, $p_i = (p_{x_i}, p_{y_i}) >$

Procedimento:;

def $d(p_1, p_2) : <$ definindo distância $>$;

retorna $\sqrt{(p_{x_1} - p_{x_2})^2 + (p_{y_1} - p_{y_2})^2}$;

Ordene a lista L de modo que $p_i \prec p_j \iff p_{x_i} \leq p_{x_j} <$ ordem crescente das primeiras coordenadas $>$;

se n *é par* **então**

  $p_M \leftarrow p_{\frac{n}{2}}$

senão

  $p_M \leftarrow p_{\frac{n-1}{2}}$

fim

$L_{esq} \leftarrow [p_1, ..., p_M]$;

$L_{dir} \leftarrow [p_{M+1}, ..., p_N]$;

$d_1 \leftarrow \infty$

$p_{esq} \leftarrow [\ ]$

para $i = 1, ..., (|L_{esq}| - 1)$ **faça**

  para $j = i + 1, ..., |L_{esq}|$ **faça**

    se $d(p_i, p_j) < d_1$ **então**

      $d_1 \leftarrow d(p_i, p_j) <$ obtendo a menor distância em $L_{esq}>$;

      $p_{esq} \leftarrow [p_i, p_j] <$ obtendo o par mais próximo em $L_{esq}>$;

    fim

  fim

fim

$d_2 \leftarrow \infty$

$p_{dir} \leftarrow [\ ]$

para $i = M + 1, ..., n - 1$ **faça**

  para $j = M + 2, ..., n$ **faça**

    se $d(p_i, p_j) < d_2$ **então**

      $d_2 \leftarrow d(p_i.p_j) <$ obtendo a menor distância em $L_{dir}>$;

      $p_{dir} \leftarrow [p_i, p_j] <$ obtendo o par mais próximo em $L_{dir}>$;

    fim

  fim

fim

$d_{min} \leftarrow 0$

$p_{min} \leftarrow [\ ]$

se $d_1 < d_2$ **então**

  $d_{min} \leftarrow d_1$;

  $p_{min} \leftarrow p_{esq}$

senão

  $d_{min} \leftarrow d_2$;

  $p_{min} \leftarrow p_{dir}$

fim

$Y \leftarrow \emptyset$

para $p_i \in L$ *tal que* $|p_{x_i} - px_m| < d_{min}$ **faça**

  $Y \leftarrow Y \cup \{p_i\} <$ adiciona pontos em que a primeira coordenada dista d_{min} de $p_M>$

fim

Ordene $M = Y$ tal que $p_i \prec p_j \iff p_{y_i} < p_{y_j}$

$Y = \{q_1, q_2, ..., q_{|Y|}\} <$ Reescreva os elementos de Y como q_i adequados à ordenação $>$

para $i = 1, ..., (|Y| - 1)$ **faça**

  para $j = i + 1, ..., |Y|$ **faça**

    enquanto $|q_{y_i} - q_{y_j}| < d_{min}$ **faça**

      $d_{min} \leftarrow d(q_i, q_j)$;

      $p_{min} \leftarrow [q_i, q_j]$

    fim

  fim

fim

retorna *o par de pontos cuja distância é* p_{min} *e sua distância é* d_{min}.

4 O problema da mochila tem a seguinte formulação: Dado um número real W e um conjunto C_n de n itens, representados por $C_n = \{1, 2, \ldots, n\}$, em que cada $i \in C_n$ tem um peso p_i e um valor v_i ($p_i > 0$ e $v_i > 0$), determine um subconjunto $S \subseteq C_n$ tal que a soma dos pesos dos elementos de S seja menor ou igual a W e a soma dos valores seja máxima (2,5 pontos).

- Proponha um algoritmo guloso para resolver esse problema. Indique um contraexemplo para o qual o seu algoritmo não funciona e explique o porquê.
  Resolução:
  Propomos o algoritmo guloso abaixo para resolvermos o problema da mochila, onde:
  v: valor dos itens;
  p: peso de cada item;
  n: número de objetos;
  W: capacidade máxima (peso máximo) da mochila.
  A ordenação considerada é tal que os elementos estejam ordenados em termo da maior "eficiência" ou benefício/custo, para a menor, isto é:

$$\frac{v[j]}{p[j]} \geq \frac{v[j+1]}{p[j+1]}, \text{ para } j = 1, ..., n-1.$$

Algoritmo 2: Mochila

```
   Dados: Mochila (v, p, n, W)
1  Procedimento:;
2  para i = 1 até n faça
3      Calcular custo [i] = v[i]
4      para j = 1, ...., n − 1 faça
5          Ordenar custo [i] em ordem não crescente
6          enquanto A mochila não estiver cheia faça
7              se p[j] ≤ W então
8                  X[i] ⟵ 1
9                  W ⟵ W − p[j]
10             senão
11                 X[i] ⟵ 0
12             fim
13         fim
14     fim
15 fim
16 retorna X
```

  - $X[i]$ é uma mochila $X[1, ..., n]$.

  Note que este algoritmo é guloso, pois em cada iteração, escolhe o objeto de maior valor, sem se preocupar com a capacidade da mochila. Observe que a solução gerada por este algoritmo nem sempre nos retorna uma mochila ótima. Por exemplo, consideremos no exemplo abaixo $W = 60$, e os itens da Tabela abaixo.

Item	A	B	C
Valor	100	280	120
Peso	10	40	20
v/p	10	7	6

Note que, utilizando o algoritmo guloso proposto, o primeiro item a ser escolhido é o item A, e em seguida o item B, donde o benefício total é $100 + 280 = 380$. Contudo, observe que a solução ótima para este problema se dá pela escolha dos itens B e C, cujo benefício é $218 + 120 = 400$. Assim, concluímos que o algoritmo guloso não nos retorna uma solução ótima.

- Descreva um algoritmo para resolver esse problema utilizando a técnica da programação dinâmica. O seu algoritmo deve determinar quais objetos pertencem ao subconjunto viável de valor máximo.
  Resolução:
  Seja $k(w, i)$ o maior valor alcançado por uma mochila de capacidade w e itens $1, ..., i$. Adicionamos este subproblema como parâmetro para a resposta que procuramos, que é $k(W, n)$, onde $k(w, i), 0 \leq i \leq n$. Este valor tem a finalidade de carregar informação acerca dos itens já utilizados. Dessa forma, podemos chamar a recursividade do problema transformando $k(w, i)$ em termos de subproblemas menores, de modo que ou o item i é necessário para alcançarmos o valor ótimo, ou não é. Assim, definimos o seguinte algoritmo utilizando programação dinâmica:

Algoritmo 3: Mochila Prog Dinâmica

Dados: $k(0, i) = 0$, $\forall i \in \{1, ..., n\}$ e $k(w, 0) = 0$, $\forall w = \{1, ..., W\}$
Procedimento:;
para $i = 1$ *até* n **faça**
  para $w = 1$ *até* W **faça**
    se $w_i > w$ **então**
      $k(w, i) = k(w, i - 1)$
    senão
      $k(w, i) = \max\{k(w, i - 1), k(w - w_i, i - 1) + v_i\}$
    fim
  fim
fim
retorna *k(W,n)*

Este algoritmo preenche uma tabela de duas dimensões, com $W + 1$ linhas e $n + 1$ colunas. Como cada chamada leva um tempo fixo de $O(n)$, e temos que fazer W chamadas, o tempo total de execução deste algoritmo será de $O(nW)$.