Análise de Algoritmos

Terceira Lista de Exercícios

Algoritmos Algébricos

1. O modelo utilizado é o de algoritmos diretos, onde cada passo é executado aplicando alguma operação aritmética no valores de entrada ou que foram computados em passos anteriores. Assim, cada passo s_i é dado por $s_i = q \diamond r$, onde $\diamond \in \{+, -, *, /\}$ e q, r são valores já obtidos. No caso do método de Horner, $s_{2i+1} = s_{2i} * x$ e $s_{2(i+1)} = s_{2i+1} + a_{n-(i+1)}$, onde $0 \leq i \leq n$ e s_0 é definido como sendo a_0.

   A prova é feita mostrando a hipótese para uma soma de $n+1$ parcelas e o resultado usado em um polinômio de grau n. Então, para um procedimento direto onde divisões não são permitidas, queremos mostrar que para calcular $a_0 + a_1 + \cdots + a_n$ precisamos de pelo menos n passos de $\pm$'s. A prova é por indução no número de parcelas.

   BI: Para $n = 0$ precisa-se de pelo menos 0 passos de $\pm$'s

   PI: Assuma que para uma soma de n parcelas são necessários pelo menos $n - 1$ passos de $\pm$'s.

   Suponha que para $a_0 + \cdots + a_n$ consegue-se um algoritmo com menos de n passos de $\pm$'s. Como não se pode assumir nenhuma propriedade para as parcelas, seja $s_i = q \diamond r$ o primeiro passo de $\pm$ onde a_n é utilizado. Esse passo existe senão a soma seria um múltiplo de a_n e sua soma poderia ser absorvida em algum passo de $*$. Sendo s_i o primeiro passo onde a_n é utilizado, q ou r é igual a a_n ou a um múltiplo de a_n. Como a_n pode ter qualquer valor, para $a_n = 0$ tem-se os seguintes casos:

   1. $s_i = q \pm 0$
   2. $s_i = 0 + r$
   3. $s_i = 0 - r$

   Nos casos 1 e 2 basta substituir s_i por q ou r, respectivamente, e em cada utilização de s_i no restante do procedimento. No caso 3 deve-se substituir s_i por $-1 * r$. Em todos os casos um passo de $\pm$ foi eliminado, resultando então em um procedimento com menos de $n - 1$ passos para calcular $a_0 + \cdots + a_{n-1}$. Isso contradiz a hipótese de indução. Portanto, um procedimento para avaliar $a_0 + \cdots + a_n$ tem pelo menos n passos de $\pm$'s.

   Para um polinômio $p(x) = a_n x^n + \cdots + a_1 x + a_0$, como x pode ser qualquer, basta considerar o caso em que $x = 1$. Assim, pelo resultado anterior teríam-se pelo menos n passos de $\pm$'s para avaliar $p(x)$.

   Referência: [BvG99]

2. Seja $p(x) = x^7 + 2x^6 + 3x^5 + 4x^4 + 5x^3 + 6x^2 + 7x + 8$. Para $k = 3$ tem-se que $2^k - 1 = 7$, $j = 2^{k-1} = 4$ e $b = a_{2^{k-1}} = a_3 - 1 = 4$. Assim, dividindo $p(x)$ por $x^4 + 4$ obtem-se $q(x) = x^3 + 2x^2 + 3x + 4$ e $r(x) = x^3 - 2x^2 - 5x - 8$.

Aplicando o pré-processamento recursivamente em $q(x)$ e $r(x)$ obtem-se a expressão final $p(x) = (x^4 + 4)[(x^2 + 2)(x + 2) + x] + [(x^2 - 6)(x - 2) + (x - 20)]$.

Para cada aplicação do método de Horner tem-se 7 $\pm$'s e 6 $*/$'s, dando um total de 70 e 60 operações, respectivamente. No método com pré-processamento, teriam-se 9 $\pm$'s e 5 $*/$'s para cada ponto, com um total de 90 $\pm$'s e 50 $*/$'s. Portanto 10 $*/$'s são poupadas ao custo de 20 $\pm$'s extras.

3. a) Sejam $V = (v_1, \dots, v_n)$ e $W = (w_1, \dots, w_n)$ dois vetores de comprimento n. O cálculo do produto $V \cdot W = \sum_{i=1}^{n} v_i w_i$, para n par, pode ser obtido por:

$$V \cdot W = \sum_{i=1}^{n/2} (v_{2i-1} + w_{2i})(v_{2i} + w_{2i-1}) - \sum_{i=1}^{n/2} v_{2i-1} v_{2i} - \sum_{i=1}^{n/2} w_{2i-1} w_{2i}$$

Para n ímpar usa-se $\lfloor n/2 \rfloor$ como limite superior dos somatórios e adicona-se o termo $v_n w_n$ na equação acima.

Na multiplicação de duas matrizes A e B a equação acima pode ser usada para o cálculo de cada elemento de $C = A \times B$. Observe que, para poupar trabalho, os elementos que dependem apenas das linhas de A e apenas das colunas de B serão pré-computados e usados repetidas vezes. Assim, para as matrizes $A_{m \times n}$ e $B_{n \times q}$ tem-se o pré-processamento $\bar{a}_i = \sum_{j=1}^{n/2} a_{i,2j-1} a_{i,2j}$, $\forall 1 \leq i \leq m$ e $\bar{b}_j = \sum_{i=1}^{n/2} a_{2i-1,j} a_{2i,j}$,$\forall 1 \leq j \leq q$. Assim, $(A \times B)_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \bar{a}_i - \bar{b}_j$.

b) Tempo: No pré-processamento tem-se $(m + q)\frac{n}{2}$ $*/$'s e $(m + q)\left(\frac{n}{2} - 1\right)$ $\pm$'s. Assim, tem-se um total de $(m+q)\frac{n}{2} + mq\frac{n}{2}$ $*/$'s e $\frac{3}{2}mnq + \frac{n}{2}(m+q) + mq - m - q$ $\pm$'s. Para matrizes quadradas tem-se que o número de $*/$'s é $\frac{n^3}{2} + n^2$ e o de $\pm$'s é $\frac{3}{2}n^3 + 2n^2 - 2n$, ambos em $\Theta(n^3)$.

Espaço: Tem-se $mn + nq$ de entrada das matrizes A e B, mq da matriz C com o resultado da multiplicação e $m + q$ dos valores de pré-processamento. Portanto, para matrizes quadradas tem-se $2n^2 + n^2 + 2n = 3n^2 + 2n \in \Theta(n^2)$.

Referência: [BvG99]

4. a) Esse método explora um algoritmo que computa a multiplicação de matrizes 2×2 com 7 $*$'s ao invés de 8, como no método de Winograd descrito anteriormente. Dadas as matrizes $A_{2\times2}$ e $B_{2\times2}$, primeiro calcula-se os 7 valores seguintes:

$$\begin{array}{lll} x_1 &=& (a_{11} + a_{22}) * (b_{11} + b_{22}) \\ x_2 &=& (a_{21} + a_{22}) * b_{11} \\ x_3 &=& a_{11} * (b_{12} - b_{22}) \\ x_4 &=& a_{22} * (b_{21} - b_{11}) \end{array} \qquad \begin{array}{lll} x_5 &=& (a_{11} + a_{12}) * b_{22} \\ x_6 &=& (a_{21} - a_{11}) * (b_{11} + b_{12}) \\ x_7 &=& (a_{12} - a_{22}) * (b_{21} + b_{22}) \end{array}$$

A matriz $C = A \times B$ será computada por:

$$\begin{array}{lll} c_{11} &=& x_1 + x_4 - x_5 + x_7 \\ c_{21} &=& x_2 + x_4 \end{array} \qquad \begin{array}{lll} c_{12} &=& x_3 + x_5 \\ c_{22} &=& x_1 + x_3 - x_2 + x_6 \end{array}$$

Suponha que n é uma potência de 2. Dado duas matrizes quadradas $A_{n\times n}$ e $B_{n\times n}$, o algoritmo consiste em dividir cada uma delas em quatro matrizes

$\frac{n}{2}\times\frac{n}{2}$ e calcular a multiplicação dessas matrizes 2×2 de acordo com as equações acima. Como os componentes são matrizes, é fundamental que as equações não assumem a comutatividade da multiplicação. O procedimento então é aplicado recursivamente em cada multiplicação de matrizes $\frac{n}{2}\times\frac{n}{2}$ no cálculo dos x_i' s acima.

Uma adaptação para matrizes que não são potência de 2 é a de aplicar o método de Strassen até atingir matrizes de dimensão ímpar e então um método, e.g. de Horner, é aplicado. Para matrizes que não são quadradas, um método $n$aive é o de completar as matrizes com 0's para que se tornem quadradas.

b) Tempo: Suponha que $n = 2^k$. O número de multiplicações de números reais é dado pela relação de recurrência

$$\begin{cases} M(0) & = & 1 \\ M(k) & = & 7M(k-1) \end{cases}$$

Logo, $M(k) = 7^k = 7^{\lg n} = n^{\lg 7} \approx n^{2.81} \in o(n^3)$.
O número de $\pm$'s é dado por

$$\begin{cases} P(0) & = & 0 \\ P(k) & = & 18(2^{k-1})^2 + 7P(k-1) \end{cases}$$

Portanto,

$$\begin{aligned} P(k) &= \sum_{i=0}^{k-1} 7^i 18(2^{k-i-1})^2 \; + \; 7^k P(0) \\ &= \frac{18}{4}(2^k)^2 \sum_{i=0}^{k-1} \left(\frac{7}{4}\right)^i \\ &= \frac{9}{2} 2^{2k} \left(\frac{(\frac{7}{4})^k - 1}{3/4}\right) \\ &= 6{\cdot}4^k \left(\frac{7^k}{4^k} - 1\right) \\ &= 6{\cdot}7^k - 6{\cdot}4^k \\ &\approx 6n^{2.81} - 6n^2 \in o(n^3) \end{aligned}$$

Espaço: Suponha que $n = 2^k$. Tem-se duas matrizes de entrada A e B, $n \times n$, e o resultado da multiplicação vai ser escrito em uma terceira matriz C, $n \times n$. Pode-se determinar recursivamente qual é o coeficiente x_1 e o resultado gravado nas submatrizes C_{11} e C_{22}. Em seguida pode-se determinar qual é o coeficiente x_2 e gravar o resultado em C_{21} e gravar $C_{22} - x_2$ em C_{22}. Assim, pode-se usar apenas duas matrizes $\frac{n}{2}\times\frac{n}{2}$ para gravar as duas matrizes a serem multiplicadas no cálculo de cada coeficiente e o resultado gravado na área correspondente, de acordo com a expressão em função dos coeficientes x_j. Logo, observando que a relação é a mesma para cada nível de recursão e que $S(1) = 2$, tem-se

$$\begin{aligned}
S(n) &= \sum_{i=0}^{k-1} 3\Big(\frac{n}{2^i}\Big)^2 \;+\; S(1) \\
&= 3\sum_{i=1}^{k} (2^i)^2 \;+\; 2 \\
&= 3\sum_{i=1}^{k} 4^i \;+\; 2 \\
&= 3\Big(\frac{4^{k+1}-1}{3} - 1\Big) + 2 \\
&= 4^{k+1} - 2 \\
&= 4(2^k)^2 - 2 \;=\; 4n^2 - 2 \;\in\; \Theta(n^2)
\end{aligned}$$

Referência: [BvG99]

5. Sejam $a+bx$ e $c+dx$ dois polinômios de grau 1. A multiplicação $(a+bx)*(c+dx)$ é dada por $ac + (ad+bc)x + bdx^2$. A idéia da multiplicação de Karatsuba é, ao invés das 4 multiplicações usuais (de coeficientes), a multiplicação dos polinômios é calculada por $ac + \big((a+b)(c+d) - ac - bd\big)x + bdx^2$. Observe que o resultado é obtido com apenas 3 multiplicações.

   Sejam $p(x)$ e $t(x)$ dois polinômios de grau $n = 2^k$. Assim, $p(x) = p_0(x) + p_1(x)x^{n/2}$ e $t(x) = t_0(x) + t_1(x)x^{n/2}$, onde $p_1(x)$ e $t_1(x)$ tem graus $\frac{n}{2}$ e os graus de $p_0(x)$ e de $t_0(x)$ são menores ou iguais a $\frac{n}{2}$. Logo, a multiplicação de Karatsuba pode ser aplicada para $p(x)*t(x)$, obtendo a seguinte expressão:

$$p_0(x)*t_0(x)+\Big((p_0(x)+p_1(x))*(t_0(x)+t_1(x))-p_0(x)*t_0(x)-p_1(x)*t_1(x)\Big)x^{n/2}+p_1(x)*t_1(x)x^n$$

   Observe que as 3 multiplicações de polinômios na expressão acima tem fatores de grau $\frac{n}{2}$. Portanto, a multiplicação de Karatsuba pode ser aplicada recursivamente, dividindo o problema original de tamanho n em 3 problemas de tamanho $\frac{n}{2}$. Observe que, em cada nível da recursão as somas e subtrações dos polinômios dão um total de $4n$ operações e que para um polinômio de grau 1 o número de operações, incluindo $*$'s, é 7. Assim, o custo do método (número de $*$'s e $\pm$'s) pode ser expresso pela seguinte relação de recurrência:

$$\begin{cases} M(1) &= 7 \\ M(n) &= 3M(n/2) + 4n \end{cases}$$

   Resolvendo a relação tem-se que

$$
\begin{aligned}
M(n) &= \sum_{i=0}^{k-1} 3^i\Big(4\frac{n}{2^i}\Big) \;+\; 3^k M(1)\\
&= 4n\sum_{i=0}^{k-1}\Big(\frac{3}{2}\Big)^i \;+\; 7\cdot 3^k\\
&= 4n\Big(\frac{(3/2)^k-1}{3/2-1}\Big) \;+\; 7\cdot 3^k\\
&= 8n\Big(\frac{3^k}{2^k}-1\Big)+7\cdot 3^k\\
&= 8\cdot 3^k-8n+7\cdot 3^k\\
&= 15\cdot 3^k-8n\\
&= 15\cdot 3^{\lg(n)}-8n \;=\; 15\cdot n^{\lg 3}-8n \;\in\; \Theta(n^{\lg 3})
\end{aligned}
$$

Um número de n dígitos $d_{n-1}\cdots d_1 d_0$ pode ser visto como $d_o+d_1 10+d_2 10^2+\cdots+d_{n-1}10^{n-1}$. Assim, tomando o polinômio $d_o+d_1x+d_2x^2+\cdots+d_{n-1}x^{n-1}$ é fácil ver como o método da multiplicação de Karatsuba é aplicado para multiplicação de números de n dígitos.

6. Seja $P(x)=1+2x+3x^2+4x^3$. Então $P_P(x)=1+3x$ e $P_I(x)=2+4x$. Pelo método recursivo para FFT, tem-se que $P(x)=P_P(x^2)+xP_I(x^2)$ e $P(-x)=P_P(x^2)-xP_I(x^2)$. Então:

$$
\begin{aligned}
P(1) &= P_P(1)+P_I(1) \qquad & P(i) &= P_P(-1)+iP_I(-1)\\
P(-1) &= P_P(1)-P_I(1) \qquad & P(-i) &= P_P(-1)-iP_I(-1)
\end{aligned}
$$

Para calcular $P_P(x)$ e $P_I(x)$, onde $x\in\{-1,1\}$, aplica-se o método recursivamente aos polinômios. Tem-se que $P_{PP}(x)=1$, $P_{PI}(x)=3$, $P_{IP}(x)=2$ e $P_{II}(x)=4$. Portanto, $P_P(1)=4$, $P_P(-1)=-2$, $P_I(1)=6$ e $P_I(-1)=-2$. Substituindo os valores nas equações da Transformada de Fourier (TF) de $P(x)$ descritas acima tem-se que $P(1)=10$, $P(-1)=-2$, $P(i)=-(2+2i)$ e $P(-i)=-2+2i$. Logo, o vetor $(10\,,\,-(2+2i)\,,\,-2\,,\,-2+2i)$ é a TF do vetor $(1\,,\,2\,,\,3\,,\,4)$.

7. O método da FFT não recursivo será aplicado no exercício. O procedimento é ilustrado abaixo:

	t	$\pi_k(t)$
p_0	000	0
p_1	001	4
p_2	010	2
p_3	011	6
p_4	100	1
p_5	101	5
p_6	110	3
p_7	111	7

p_0	p_4	p_2	p_6	p_1	p_5	p_3	p_7
$P_{PP}(1)$	$P_{PP}(-1)$	$P_{PI}(1)$	$P_{PI}(-1)$	$P_{IP}(1)$	$P_{IP}(-1)$	$P_{II}(1)$	$P_{II}(-1)$
$P_P(1)$	$P_P(i)$	$P_P(-1)$	$P_P(-i)$	$P_I(1)$	$P_I(i)$	$P_I(-1)$	$P_I(-i)$
$P(1)$	$P(\omega)$	$P(\omega^2)$	$P(\omega^3)$	$P(\omega^4)$	$P(\omega^5)$	$P(\omega^6)$	$P(\omega^7)$

O procedimento começa achando a permutação apropriada dos coeficientes através da funcão π_k, que leva a representação binária de um número em seu reverso.

Assim, a computação *b*ottom-up inicia-se nas folhas da árvore representada no procedimento recursivo.

Para $P(x) = 1 + 2x + 3x^2 + 4x^3 + 5x^4 + 6x^5 + 7x^6 + 8x^7$, a sequência de computação será dada por

1	5	3	7	2	6	4	8

6	-4	10	-4	8	-4	12	-4

16	$-4(1+i)$	-4	$4(-1+i)$	20	$-4(1+i)$	-4	$4(-1+i)$

36	$-4-4(1+\sqrt{2})i$	$-4(1+i)$	$-4+4(1-\sqrt{2})i$	-4	$-4+4(-1+\sqrt{2})i$	$4(-1+i)$	$-4+4(1+\sqrt{2})i$

8. Seja $A(x) = a_0 + a_1x + \cdots + a_7x^7$. Então, para o vetor $A = (a_0, a_1, \ldots, a_7)$ tem-se o seguinte circuito na Figura 1. Observe que as saídas y_i para cada $0 \leq i \leq 7$ representam $A(\omega^i)$, onde $\omega = \frac{(1+i)}{\sqrt{2}}$.

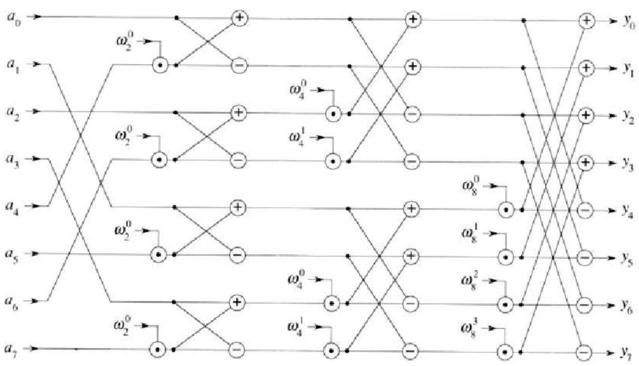

Figure 1: Circuito para FFT8

Fonte: `http://www.cs.fsu.edu/~cop4531/slideshow/chapter32/32-3.html`

9. Sejam $p(x)$ e $q(x)$, dois polinômios de grau $n-1$ há serem multiplicados. Os 3 passos que compõem a multiplicação de polinômios baseado no FFT são descritos a seguir:

   1. Avaliação de $p(x)$ e $q(x)$ em $2n$ pontos. O método de FFT é usado para avaliar os polinômios nas $2n$-ésimas raízes de 1. Assumindo que n é potência de 2, o tempo de avaliação está em $\Theta(2n \lg(2n))$.

      Para o espaço, temos um vetor omega de comprimento $2n$ (onde ficam gravadas a n-raízes primitivas de 1), e dois vetores de comprimento $2n$ onde o resultado do primeiro passo está gravado (observe que os coeficientes de $p(x)$ e $q(x)$, que são entradas, podem estar gravados nas n primeiras posições dos respectivos vetores). Portanto, o espaço utilizado é de $2n + 2 \cdot 2n = 3 \cdot 2n \in \Theta(2n)$.

   2. Achar os valores de $r(x) = p(x) * q(x)$ nesses $2n$ pontos. Para isso, basta multiplicar os vetores calculados anteriormente, ponto a ponto. Para isso, tem-se $2n$ multiplicações, ou seja, o tempo de execução em $\Theta(2n)$. O resultado será gravado em um vetor de comprimento $2n$, que pode ser um dos vetores utilizados anteriormente

3. Achar o único polinômio de grau $2n-2$ que passa pelos pontos $(\omega^i, r(\omega^i))$, para $0 \leq i \leq 2n-1$. Observe que $F_{2n}R = W$ é o valor calculado anteriormente, onde R é o vetor composto pelos coeficientes de $r(x)$. Logo, $R = F_{2n}^{-1}W$. Tem-se que nF_{2n}^{-1} é composta pelas colunas de F_{2n} com as colunas de 2 a $2n$ em ordem reversa (devido a $(F_n^{-1})_{ij} = \frac{1}{n}\omega^{-ij}$, n qualquer, e que $\omega^{-i} = \omega^{n-i}$, para $1 \leq i \leq n-1$). Assim, para o vetor $W' = (r(1), r(\omega^{2n-1}), \ldots, r(\omega))$, tem-se que $\frac{1}{n}F_{2n}^{-1}W' = R$. Logo, o tempo da terceira parte inclui mais uma aplicação de FFT, em $\Theta(2n\lg(2n))$, e $2n$ divisões. O espaço utilizado é de um vetor de comprimento $2n$ para W' e onde o FFT e as divisões serão aplicados.

Assim, o tempo total do procedimento está em $\Theta(2n\lg(2n))$ e o espaço em $\Theta(2n)$. Como a entrada são dois polinômios de grau $n-1$, ou seja dois vetores de comprimento n cada (com os respectivos coeficientes) tem-se que, para $m = 2n$, o tempo está em $\Theta(m\lg(m))$ e o espaço em $\Theta(m)$.

Exemplo: Sejam $P(x) = 1 + 4x^2 + 2x^3$ e $Q(x) = x + 2x^2 + x^3 + 2x^4$. Sejam $P = (1,0,4,2,0,0,0,0)$ e $Q = (1,0,2,1,2,0,0,0)$.

1^o **passo**: Para F_8P tem-se que

1	0	4	0	0	0	2	0

1	1	4	4	0	0	2	2

5	$1+4i$	-3	$1-4i$	2	$2i$	-2	$-2i$

7	$1-\sqrt{2}+(4+\sqrt{2})i$	$-3-2i$	$1+\sqrt{2}+(\sqrt{2}-4)i$	3	$1+\sqrt{2}+(4-\sqrt{2})i$	$-3+2i$	$1-\sqrt{2}-(4+\sqrt{2})i$

Para F_8Q tem-se que

0	2	2	0	1	0	1	0

2	-2	2	2	1	1	1	1

4	$-2+2i$	0	$-2-2i$	2	$1+i$	0	$1-i$

6	$-2+(2+\sqrt{2})i$	0	$-2-(2-\sqrt{2})i$	2	$-2+(2-\sqrt{2})i$	0	$-2-(2+\sqrt{2})i$

2^o **passo**: Obtem-se $W = (R(1), R(\omega), \ldots, R(\omega^7))$, onde $R(x) = P(x) * Q(x)$, multiplicando os vetores calculados anteriormente coordenada a coordenada, obtendo

42	$-12-4\sqrt{2}-(3\sqrt{2}+8)i$	0	$-12+4\sqrt{2}+(8-3\sqrt{2})i$	6	$-12+4\sqrt{2}+(3\sqrt{2}-8)i$	0	$-12-4\sqrt{2}+(3\sqrt{2}+8)i$

3^o **passo**: Para W' obtido de W mantendo-se a primeira posição e invertendo o resto da lista tem-se, aplicando FFT, que:

42	6	0	0	$-12-4\sqrt{2}+(3\sqrt{2}+8)i$	$-12+4\sqrt{2}+(8-3\sqrt{2})i$	$-12+4\sqrt{2}+(3\sqrt{2}-8)i$	$-12-4\sqrt{2}-(3\sqrt{2}+8)i$

48	36	0	0	$-24+16i$	$-\sqrt{2}(8-6i)$	$-24-16i$	$\sqrt{2}(8+6i)$

48	36	48	36	-48	$-14\sqrt{2}(1-i)$	$32i$	$-2\sqrt{2}(1+i)$

0	8	16	40	96	64	80	32

Logo, $\frac{1}{8}F_8W' = (0, 1, 2, 5, 12, 8, 10, 4)$. Portanto,

$$R(x) = x + 2x^2 + 5x^3 + 12x^4 + 8^5 + 10x^6 + 4x^7$$

Noções de Teoria de Complexidade

10. (a) 3-coloração de grafos está em $\mathcal{NP}$.

    Algoritmo: Seja $G = (V, E)$, um grafo tal que $|V| = n$ e $|E| = m$. Assuma que tem-se fixada uma ordem qualquer dos vértices em V.

    1. Gerar uma string de comprimento n e verificar se é sintaticamente bem formada como a entrada de um problema de 3-coloração de grafos. Em outras palavras, se a string é composta por exatamente 3 símbolos diferentes.
    2. Verificar, $\forall(u, v) \in E$, se $C(u) \neq C(v)$, onde se $x \in V$, $C(x)$ é a cor associada a x através da posição de x na ordem dos vértices e da string gerada no passo 1.

    Tem-se uma entrada de tamanho $n + m$. O tempo $T_1(n + m)$ do primeiro passo está em $\Theta(n)$ e $T_2(n + m)$ do segundo em $\Theta(m)$. Assim, $T(n + m) = T_1(n + m) + T_2(n + m)$ está em $\Theta(n + m)$.

    (b) O problema de soma de subconjuntos está em $\mathcal{NP}$.

    Algoritmo: Seja $S = \{s_1, \ldots s_n\}$ um conjunto de numeros naturais e C também um natural.

    1. Gerar uma string t de comprimento no máximo n e verificar se esta forma um subconjunto de S (e.g. verificando se é composta apenas de números, menores ou iguais a n e se não possui repetição de chaves).
    2. Para $t = t_1 \cdots t_m$, string gerada no passo 1, verificar se $\sum_{i=1}^{m} s_{t_i} = C$.

    A entrada do problema é $n + 1$. Observe que neste caso o tamanho de C (de sua representação) não influi na complexidade como na versão em programação dinâmica, sendo tratado como um objeto assim como os elementos de S. O primeiro passo está em $\Theta(n)$. O segundo passo tem a soma, que é linear em n, mais uma comparação do total com C, estando portanto em $\Theta(n + 1)$.

    (c) O problema de execução de tarefas com penalidades está em $\mathcal{NP}$.

Algoritmo: Sejam $J_1, \ldots, J_n$ um conjunto de tarefas, onde $t_1, \ldots, t_n$ são seus respectivos tempos de execução, $d_1, \ldots, d_n$ os respectivos limites de tempo, contando do início da primeira execução, e $p_i, \ldots, p_n$ as respectivas penalidades. Dado $k \in \mathbb{N}$ tem-se:

1. Pegar aleatoriamente π, uma permutação de J_i's.
2. Calcular $P_\pi = \sum_{j=1}^{n} [\textbf{if } t_{\pi(1)} + \cdots + t_{\pi(j)} > d_{\pi(j)} \textbf{ then } p_{\pi(j)} \textbf{ else } 0]$.
   Basta então fazer uma comparação entre o P_π calculado e o k dado.

O primeiro passo está em $\Theta(n)$. No segundo passo são feitas n comparações no cálculo de P_π e mais uma comparação do resultado com k. Assim, o tempo do passo 2 está em $\Theta(n + 1)$.

(d) O problema de caminhos Eulerianos em grafos(conexos) está em $\mathcal{P}$.

Algoritmo: Seja $G = (V, E)$ um grafo tal que $|V| = n$ e $|E| = m$. Para verificar se existe um caminho Euleriano, basta verificar se cada vértice tem grau par (essa propriedade garante que ao percorrer o grafo, cada vez que se "entra" em um vértice é possível "sair", usando arestas distintas). A verificação pode ser feita determinando o grau de cada vértice, com tempo em $\Theta(m)$, seguido da checagem de paridade de cada grau, com tempo em $\Theta(n)$. Portanto, o algoritmo tem um custo de tempo em $\Theta(n + m)$.

(e) O problema de caminhos Hamiltonianos em grafos está em $\mathcal{NP}$.

Algoritmo: Seja $G = (V, E)$ um grafo tal que $|V| = n$ e $|E| = m$. Assuma que existe uma ordem para os elementos de V.

1. Pegar aleatoriamente uma permutação π dos elementos em V.
2. Verificar na sequência de vértices, dada pela permutação π, se vértices consecutivos tem arestas em E e se existe uma aresta ligando o último elemento da sequência ao primeiro. Ou seja, se $(v_{\pi(n)}, v_{\pi(1)}) \in E$ e se $(v_{\pi(i)}, v_{\pi(i+1)}) \in E$, $\forall 1 \leq i \leq n - 1$.

O número de arestas m está relacionado com o número de vértices n. No pior caso tem-se $m = n^2$, onde cada vértice tem aresta para todos os vértices. Assim, o tamanho da entrada pode ser expresso por $s = n^2 + n$. O passo 1 está em $\Theta(n)$. O passo 2 é limitado superiormente por n^3 comparações (comparação de $(v_{\pi(i)}, v_{\pi(i+1)})$ com os n^2 elementos de E), logo em $O(n^3)$. Assim, o custo do procedimento em número de comparações está em $O(n^3)$, logo em $o(n^4)$. Portanto, existe um polinômio $p(x)$ de grau 2 tal que $p(n + n^2)$ é limite superior para o custo total.

11. Se um problema de decisão Π está em $\mathcal{P}$, então existe um algoritmo A_Π que decide Π, limitado por um polinômio $p(x)$. Ou seja, o pior caso $W(n)$ de A_Π é limitado por $p(n)$. Se um problema Γ reduz-se polinomialmente a um problema Π, denotado por $\Gamma \leq_{\mathcal{P}} \Pi$, então existe uma transformação $T(x)$, com o tempo de computação limitado por um polinômio $q(y)$, tal que se x é uma instância de Γ então $T(x)$ é uma instância de Π onde a resposta para x em Γ coincide com a resposta para $T(x)$ em Π.

    Um algoritmo A_Γ para Γ pode ser composto da seguinte maneira: (1) Para uma instância x de Γ de tamanho n, compute $T(x)$. (2) Aplique A_Π em $T(x)$ e retorne a resposta obtida. O primeiro passo tem um tempo limitado por $q(n)$. O tamanho de $T(x)$ será, no pior caso, de $q(n)$ (quando um programa escreve um símbolo em cada passo), logo o tempo do segundo passo é limitado por $p(q(n))$. Portanto, o tempo do algoritmo A_Γ tem o pior caso limitado por $q(n) + p(q(n))$. Logo, pelo fechamento de polinômios para composição e soma, $\Gamma \in \mathcal{P}$.

12. Para provar que $CLIQUE \in \mathcal{NP}$, tem-se o algoritmo descrito abaixo:

    Algoritmo: Seja $G = (V, E)$ um grafo tal que $|V| = n$, $|E| = m$.

    1. Pegar aleatoriamente um subconjunto V' de V.
    2. Verificar se para cada $u, v \in V'$, $(u, v) \in E$.

    O passo 1 está em $\Theta(n)$. O passo 2 tem no pior caso $V' = V$, onde teriam-se n^2 comparações há serem feitas com os elementos de E, estando portanto em $O(n^4)$. Logo, existe um polinômio $p(x)$ tal que $p(n^2 + n)$ é um limite superior ao custo em número de comparações. Portanto, $CLIQUE \in \mathcal{NP}$.

    Seja $C \equiv C_1 \wedge \cdots \wedge C_n$, uma expressão lógica em forma normal conjuntiva. Suponha, sem perda de generalidade, que o número de literais (variáveis lógicas e suas negações) em cada cláusula C_i é o mesmo. Assim, $C_i \equiv l_1^i \vee l_2^i \cdots \vee l_m^i$. Seja $G = (V, E)$ um grafo construído da seguinte forma

    1. Seja v_j^i o vértice correspondente a l_j^i, para cada $1 \leq i \leq n$ e $1 \leq j \leq m$.
    2. Para cada $v_j^i, v_l^k \in V$, $(v_j^i, v_l^k) \in E$ se, e somente se, $i \neq k$ e os literais l_j^i, l_l^k, correspondentes aos vértices, não são negação um do outro.

    No processo de construção do grafo, tem-se uma entrada de tamanho nm (número de literais em C). O passo 1 é feito em $\Theta(nm)$. No passo 2, para cada vértice tem-se $((nm)^2 - 1)$ comparações, portanto tem-se um limite superior de $O((nm)^2)$ Assim, o tempo total de construção de G à partir de C é limitado por $O((nm)^2)$.

    Agora, basta verificar se o problema de n-clique para o grafo construído corresponde ao problema de satisfatibilidade para C. Suponha que C é satisfatível. Logo, para cada C_i em C, existe um literal l_j^i tal que o seu valor é *verdadeiro*. Seja $\{l_{j_1}^1, \ldots, l_{j_n}^n\}$ o conjunto desses literais. Por estarem em cláusulas diferentes e por nenhum destes literais ser a negação dos outros, pois tem associado o mesmo valor na designação de valores para as variáveis de C, tem-se por construção que o vértice correspondente a cada $l_{j_i}^i$ tem arestas para todos os outros vértices correspondente aos literais no conjunto. Logo, tem-se um n-clique em G.

    Suponha que exista um n-clique em G. Seja $V' = \{v_{j_1}^{i_1}, \ldots, v_{j_n}^{i_n}\}$ o conjunto dos vértices que compõem o n-clique. Tem-se que, tomando quaisquer dois vértices em V', existe uma aresta em E ligando os dois. Logo, por construção, os literais correspondentes $l_{j_1}^{i_1}, \ldots, l_{j_n}^{i_n}$ estão em cláusulas diferentes e nenhum é a negação do outro. Portanto, tomando uma valoração tal que esses literais sejam *verdadeiro* tem-se uma designação para variáveis de C tal que esta seja *verdadeiro*.

    Portanto, *CNF-SAT* $\leq_{\mathcal{P}}$ *CLIQUE*. Pelo Teorema de Cook *CNF-SAT* é $\mathcal{NP}$-completo, logo $CLIQUE$ é $\mathcal{NP}$-completo.