ANÁLISE DE ALGORITMOS

Paulo Feofiloff

Instituto de Matemática e Estatística
Universidade de São Paulo

agosto 2009

Introdução

Problema

Encontrar a soma dos elementos positivos de um vetor $A[1\,..\,n]$

Uma instância do problema: Encontrar a soma dos elementos positivos do vetor $(20, -30, 15, -10, 30, -20, -30, 30)$

1							$n = 8$
20	−30	15	−10	30	−20	−30	30

Algoritmo

SomaPositivos $(A,\ n)$

$s \leftarrow 0$

para $i \leftarrow 1$ até n faça

se $A[i] > 0$

então $s \leftarrow s + A[i]$

devolva s

O algoritmo está correto?

- testes só podem mostrar que o algoritmo está errado
- análise pode provar que o algoritmo está correto

O algoritmo está correto

Invariante: no começo de cada iteração

- s é a soma dos positivos de $A[1\,..\,i{-}1]$

No fim, s é a soma dos positivos de $A[1\,..\,n]$

Quanto tempo consome a execução do algoritmo?

- depende da instância

Consumo de tempo do algoritmo

- proporcional ao número de iterações
- tempo de cada iteração não depende de n
- tempo total: proporcional a n
- se n dobra, o tempo dobra
- se n decuplica, o tempo decuplica

Observações sobre consumo de tempo:

- estimar consumo *do algoritmo*, independente do computador
- despreze constantes multiplicativas: $10\,n$ é o mesmo que n
- consumo de tempo é diferente para cada instância do problema
- agrupe instâncias por "tamanho"
- o conceito de tamanho de uma instância
- muitas instâncias têm o mesmo tamanho
- consumo de tempo no pior caso
- consumo de tempo no melhor caso

De volta ao problema da soma dos elementos positivos

Algoritmo recursivo

SomaPos $(A,\ n)$
se $n = 0$
    então devolva 0
    senão $s \leftarrow$ SomaPos $(A,\ n-1)$
        se $A[n] > 0$
            então devolva $s + A[n]$
            senão devolva s

Consumo de tempo de SomaPos

$T(n)$: consumo de tempo no pior caso

- recorrência: $T(n) = T(n-1) + \text{const}$
- $T(n) = ?$
- preciso aprender a resolver recorrências

Observações sobre algoritmos recursivos

Problemas com estrutura recursiva:

- cada instância do problema
  contém uma instância menor do mesmo problema

Algoritmo recursivo:

se a instância em questão é pequena
    resolva-a diretamente
senão
    reduza-a a uma instância menor do mesmo problema
    encontre solução S da instância menor
    use S para construir solução da instância original

"Para entender recursão,
é preciso primeiro entender recursão."

— folclore

"Ao tentar resolver o problema,
encontrei obstáculos dentro de obstáculos.
Por isso, adotei uma solução recursiva."

— um aluno

Comparação assintótica de funções

- funções de $\mathbb{N}$ em $\mathbb{R}^{\geq}$
- $G(n) \geq 0$ para $n = 0, 1, 2, 3, \ldots$

Definição da ordem $\mathrm{O}(G)$

Função F está em $\mathrm{O}(G)$ se existe c em $\mathbb{N}^{>}$ tal que
$F(n) \ \leq \ c \cdot G(n)$ para todo n suficientemente grande

- "F está em $\mathrm{O}(G)$" tem sabor de "$F \leq G$"
- ... tem sabor de "F não cresce mais que G"
- conceito sob medida para tratar de consumo de tempo de algoritmos

- Exemplo 1: $100n$ está em $\mathrm{O}(n^2)$
- Prova: Para todo $n \geq 100$

$$\begin{aligned} 100\,n &\leq n \cdot n \\ &= n^2 \\ &= 1 \cdot n^2 \end{aligned}$$

- Exemplo 2: $2n^3 + 100n$ está em $\mathrm{O}(n^3)$
- Prova: Para todo $n \geq 1$

$$\begin{aligned} 2n^3 + 100n &\leq 2n^3 + 100n^3 \\ &\leq 102\,n^3 \end{aligned}$$

- Outra prova: Para todo $n \geq 100$

$$\begin{aligned} 2n^3 + 100n &\leq 2n^3 + n \cdot n \cdot n \\ &= 3\,n^3 \end{aligned}$$

- Exemplo 3: n está em $\mathrm{O}(2^n)$
- Prova: prove por indução em n que

  $n \leq 2^n$ para todo $n \geq 1$

- Exemplo 4: $\lg n$ está em $\mathrm{O}(n)$
- Prova: para todo $n \geq 1$

$$n \leq 2^n$$

$$\lg n \leq n$$

Ordem Ômega

- "$F \in \Omega(G)$" tem sabor de "$F \geq G$"
- $F \in \Omega(G) \quad \Leftrightarrow \quad G \in \mathrm{O}(F)$

Definição da ordem $\Omega(G)$

Função F está em $\Omega(G)$ se existe c em $\mathbb{N}^{>}$ tal que
$F(n) \ \geq \ \frac{1}{c} \cdot G(n)$ para todo n suficientemente grande

Exemplos:

- $n^2 - 2n$ está em $\Omega(n^2)$
- $n \lg n$ está em $\Omega(n)$
- $100\,n$ não está em $\Omega(n^2)$

Ordem Teta

- tem sabor de "$F = G$"

Definição da ordem $\Theta(G)$

Função F está em $\Theta(G)$ se $F \in \mathrm{O}(G)$ e $F \in \Omega(G)$

Algumas funções em $\Theta(n^2)$:

- $11n^2 - 22n + 33$
- $2n^2 + 3n \lg n - 4n + 5$

Consumo de tempo de algoritmos

Algoritmo linear:

- consome $\Theta(n)$ unidades de tempo no pior caso
- n multiplicado por $10 \;\Rightarrow\;$ tempo multiplicado por 10
- algoritmos lineares são considerados muito rápidos

Algoritmo quadrático:

- tempo $\Theta(n^2)$ no pior caso
- n multiplicado por $10 \;\Rightarrow\;$ tempo multiplicado por 100

Algoritmo polinomial:

- consome tempo $\mathrm{O}(n^k)$ para algum k
- exemplos: $\mathrm{O}(n)$, $\mathrm{O}(n \lg n)$, $\mathrm{O}(n^2)$, $\mathrm{O}(n^{100})$
- não-exemplos: $\Omega(2^n)$, $\Omega(1.1^n)$

Algoritmo exponencial:

- consome tempo $\Omega(a^n)$ para algum $a > 1$
- exemplos: $\Omega(2^n)$, $\Omega(1.1^n)$
- n multiplicado por $10 \;\Rightarrow$ tempo elevado a 10

Solução de recorrências

Exemplo 1

- função F de $\mathbb{N}$ em $\mathbb{R}^{>}$
- sabe-se que $F(1) = 1$ e
$$F(n) = 2\,F(n-1) + 1 \quad \text{para } n \geq 2$$
-

n	0	1	2	3	4	$\cdots$
$F(n)$	?	1	3	7	15	$\cdots$

- queremos "fórmula" para $F(n)$

Fato

$F(n) = 2^n - 1$ para todo $n \geq 1$

Prova:

$$
\begin{aligned}
F(n) &= 2\,F(n-1)+1 \\
&= 2\,\big(2F(n-2)+1\big)+1 \\
&= 4F(n-2)+3 \\
&\ \vdots \\
&= 2^j F(n-j) + 2^j - 1 \\
&= 2^{n-1}F(1) + 2^{n-1} - 1 \\
&= 2^{n-1} + 2^{n-1} - 1 \\
&= 2^n - 1
\end{aligned}
$$

Conseqüência

$F(n)$ está em $\Theta(2^n)$

Exemplo 2

- função F de $\mathbb{N}$ em $\mathbb{R}^{>}$
- sabe-se que $F(1) = 1$ e
$$F(n) = 2\,F(\lfloor n/2 \rfloor) + n \quad \text{para todo } n \geq 2$$
n	0	1	2	3	4	$\cdots$
$F(n)$	?	1	4	5	12	$\cdots$
- queremos "fórmula" para $F(n)$
- trate primeiro das potências de 2

Fato A (potências de 2)

$F(n) = n \lg n + n$ para $n = 1, 2, 4, 8, 16, 32, 64, \ldots$

Prova, com $n = 2^j$

$$\begin{aligned}
F(2^j) &= 2\,F(2^{j-1}) + 2^j \\
&= 2\left(2F(2^{j-2}) + 2^{j-1}\right) + 2^j \\
&= 2^2F(2^{j-2}) + 2^1 2^{j-1} + 2^j \\
&= 2^3F(2^{j-3}) + 2^2 2^{j-2} + 2^1 2^{j-1} + 2^j \\
&= 2^jF(2^0) + 2^{j-1}2^1 + \cdots + 2^2 2^{j-2} + 2^1 2^{j-1} + 2^0 2^j \\
&= 2^j 2^0 + 2^{j-1}2^1 + \cdots + 2^2 2^{j-2} + 2^1 2^{j-1} + 2^0 2^j \\
&= (j+1)2^j \\
&= j\,2^j + 2^j
\end{aligned}$$

Se n não é potência de 2 . . .

Fato B

$F(n) \;\leq\; 6\, n \lg n$ para todo $n \geq 2$

Prova:

- $F(n) = n \lg n + n$ quando n é potência de 2
- F é crescente
- $\ldots$

Fato C

$F(n) \;\geq\; \frac{1}{2}\, n \lg n$ para todo $n \geq 2$

Prova:

- $F(n) = n \lg n + n$ quando n é potência de 2
- F é crescente
- $\ldots$

Conseqüência

$F(n)$ está em $\Theta(n \lg n)$

Detalhes não alteram a ordem de F:

- se $F(n) = F(\lfloor n/2 \rfloor) + F(\lceil n/2 \rceil) + 10n$
- então F continua em $\Theta(n \lg n)$

Exemplo 3

- função F de $\mathbb{N}$ em $\mathbb{R}^>$
- $F(1) = 1$ e
  $F(n) = 3\,F(\lfloor n/2 \rfloor) + n$ para todo $n \geq 2$
- "fórmula" para $F(n)$?

Fato A (para $n = 2^j$)

$F(2^j) = 3 \cdot 3^j - 2 \cdot 2^j$ para todo $j \geq 0$

Prova, por indução em j

- se $j = 0$ então $F(2^j) = 1 = 3 \cdot 3^j - 2 \cdot 2^j$
- agora tome $j \geq 1$
- hipótese de indução: $F(2^{j-1}) = 3^j - 2^j$
- $$\begin{aligned} F(2^j) &= 3\,F(2^{j-1}) + 2^j \\ &= 3\left(3^j - 2^j\right) + 2^j \\ &= 3 \cdot 3^j - 2 \cdot 2^j \end{aligned}$$

- $3 = 2^{\lg 3}$
- $3^j = (2^{\lg 3})^j = (2^j)^{\lg 3} = n^{\lg 3}$
- $1.5 < \lg 3 < 1.6$
- $n\sqrt{n} < n^{\lg 3} < n^2$

Conseqüência

$F(n) = 3\,n^{\lg 3} - 2n$ para $n = 1, 2, 4, 8, 16, 32, 64, \ldots$

Se n não é potência de 2 ...

Fato B (n arbitrário)

$F(n) \ \leq \ 9\, n^{\lg 3}$ para todo $n \geq 1$

Fato C (n arbitrário)

$F(n) \ \geq \ \frac{1}{3}\, n^{\lg 3}$ para todo $n \geq 1$

Conseqüência

F está em $\Theta(n^{\lg 3})$

Detalhes não alteram a ordem de F:

- se $F(n) = 2\,F(\lfloor n/2 \rfloor) + F(\lceil n/2 \rceil) + 10n$
- então F continua em $\Theta(n^{\lg 3})$

Teorema mestre

Se

- $F : \mathbb{N} \to \mathbb{R}^{>}$
- $F(n) = a\,F\left(\frac{n}{2}\right) + cn^{k}$ para $n = 2^1, 2^2, 2^3, \ldots$
- F é crescente

então

- se $\lg a > k$ então F está em $\Theta(n^{\lg a})$
- se $\lg a = k$ então F está em $\Theta(n^{k} \lg n)$
- se $\lg a < k$ então F está em $\Theta(n^{k})$

Ordenação de vetor

Problema da ordenação

Rearranjar um vetor $A[p \,.\,.\, r]$ em ordem crescente

- vetor é crescente se $A[p] \leq A[p+1] \leq \cdots \leq A[r]$

Algoritmo Mergesort

MergeSort $(A,\ p,\ r)$
se $p < r$
    então $q \leftarrow \lfloor (p+r)/2 \rfloor$
        MergeSort $(A,\ p,\ q)$
        MergeSort $(A,\ q+1,\ r)$
        Intercala $(A,\ p,\ q,\ r)$

p					q	$q{+}1$				r
111	333	555	555	777	888	222	444	777	999	999

Intercala rearranja $A[p\mathinner{.\,.}r]$ em ordem crescente
supondo $A[p\mathinner{.\,.}q]$ e $A[q{+}1\mathinner{.\,.}r]$ crescentes

O algoritmo está correto

Tamanho de uma instância: $n = r - p + 1$

- se $n \leq 1$ então $A[p\,.\,.\,r]$ já é crescente
- agora suponha $n \geq 2$
- por hipótese de indução, $A[p\,.\,.\,q]$ é crescente
- por hipótese de indução, $A[q{+}1\,.\,.\,r]$ é crescente
- INTERCALA coloca $A[p\,.\,.\,r]$ em ordem crescente

Consumo de tempo

$T(n)$: consumo de tempo no pior caso

- recorrência: $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$
- parcela n representa o consumo de INTERCALA
- semelhante a $T(n) = 2\,T(\lfloor n/2 \rfloor) + n$
- solução da recorrência: T está em $\Theta(n \lg n)$

Método de divisão e conquista:

- instância original do problema é dividida em duas menores
- instâncias menores resolvidas recursivamente
- as duas soluções são combinadas

Segredo do sucesso: divisão e combinação devem ser rápidas!

Multiplicação de inteiros

$$35871227428009 \times 11234908764388$$

Problema

Dados números naturais u e v com n dígitos cada
calcular o produto $u \cdot v$

8	7	6	5	4	3	2	1
7	7	7	6	2	2	2	3

- cada número tratado como um vetor de dígitos
- $u \cdot v$ terá $2n$ dígitos

Algoritmo usual de multiplicação

$$
\begin{array}{r}
9\,9\,9\,9 \\
7\,7\,7\,7 \\
\hline
6\,9\,9\,9\,3 \\
6\,9\,9\,9\,3 \\
6\,9\,9\,9\,3 \\
6\,9\,9\,9\,3 \\
\hline
7\,7\,7\,6\,2\,2\,2\,3
\end{array}
$$

Consumo: $\Theta(n^2)$ unidades de tempo

Preparando algoritmo mais eficiente

u	9	9	9	9	8	8	8	8
		a				b		

Truque (supondo n par)

- $u = a \cdot 10^{n/2} + b$
- $v = c \cdot 10^{n/2} + d$
- $u \cdot v \ = \ a \cdot c \cdot 10^n + (a \cdot d + b \cdot c) \cdot 10^{n/2} + b \cdot d$
- 4 multiplicações de tamanho $n/2$
- bom algoritmo para calculadora pequena!

$$
\begin{array}{rl}
99998888 & u \\
77776666 & v \\
\\
9999 & a \\
8888 & b \\
7777 & c \\
6666 & d \\
\\
77762223 & ac \\
135775310 & ad + bc \\
59247408 & bd \\
\\
7777580112347408 & x
\end{array}
$$

Algoritmo (supondo que n é potência de 2)

$\textsc{Multiplica}\ (u,\ v,\ n)$

se $n = 1$
    então devolva $u \cdot v$
    senão $k \leftarrow n/2$
            $a \leftarrow \lfloor u/10^k \rfloor$
            $b \leftarrow u \bmod 10^k$
            $c \leftarrow \lfloor v/10^k \rfloor$
            $d \leftarrow v \bmod 10^k$
            $ac \leftarrow \textsc{Multiplica}\ (a,\ c,\ k)$
            $bd \leftarrow \textsc{Multiplica}\ (b,\ d,\ k)$
            $ad \leftarrow \textsc{Multiplica}\ (a,\ d,\ k)$
            $bc \leftarrow \textsc{Multiplica}\ (b,\ c,\ k)$
            $x \leftarrow ac \cdot 10^{2k} + (ad + bc) \cdot 10^k + bd$
            devolva x

Consumo de tempo (n é potência de 2)

Tamanho de uma instância: n

- $T(n)$: consumo de tempo do algoritmo no pior caso
- recorrência: $T(n) = 4\,T(n/2) + n$
- n é o consumo das linhas 3–7 e 12
- solução: $T(n)$ está em $\Theta(n^2)$

Algoritmo MULTIPLICA não é mais rápido que algoritmo usual...

Algoritmo mais eficiente

Antes: 4 multiplicações de tamanho $n/2$:

$$u \cdot v \;=\; a \cdot c \cdot 10^n + (a \cdot d + b \cdot c) \cdot 10^{n/2} + b \cdot d$$

Agora: 3 multiplicações de tamanho $n/2$:

$$u \cdot v \;=\; a \cdot c \cdot 10^n + (y - a \cdot c - b \cdot d) \cdot 10^{n/2} + b \cdot d$$

$$y = (a + b) \cdot (c + d)$$

999988888	u
077766666	v
07769223	ac
5925807408	bd
98887	$a+b$
67443	$c+d$
6669235941	y
735659310	$y-ac-bd$
077765801856807408	x

Algoritmo de Karatsuba–Ofman: um rascunho

$\textsc{Karatsuba}\ (u,\ v,\ n)$

se $n \leq 1$

  então devolva $u \cdot v$

  senão $k \leftarrow \lceil n/2 \rceil$

    $a \leftarrow \lfloor u/10^k \rfloor$

    $b \leftarrow u \bmod 10^k$

    $c \leftarrow \lfloor v/10^k \rfloor$

    $d \leftarrow v \bmod 10^k$

    $ac \leftarrow \textsc{Karatsuba}\ (a,\ c,\ k)$

    $bd \leftarrow \textsc{Karatsuba}\ (b,\ d,\ k)$

    $y \leftarrow \textsc{Karatsuba}\ (a+b,\ c+d,\ k+1)$

    $x \leftarrow ac \cdot 10^{2k} + (y - ac - bd) \cdot 10^k + bd$

    devolva x

Idéia básica correta, mas tem erros técnicos

Consumo de tempo

$T(n)$: consumo de tempo do algoritmo no pior caso

- recorrência: $T(n) = 3\,T(\lfloor n/2 \rfloor) + n$
- solução: $T(n)$ está em $\Theta(n^{\lg 3})$

- $1.5 < \lg 3 < 1.6$
- $n\sqrt{n} < n^{\lg 3} < n^2$
- para n grande, Karatsuba é bem mais rápido que algoritmo usual

Algoritmo de Karatsuba–Ofman: versão final, n arbitrário

$\textsc{Karatsuba}\ (u,\ v,\ n)$

se $n \leq 3$
    então devolva $u \cdot v$
    senão $k \leftarrow \lceil n/2 \rceil$
        $a \leftarrow \lfloor u/10^k \rfloor$
        $b \leftarrow u \bmod 10^k$
        $c \leftarrow \lfloor v/10^k \rfloor$
        $d \leftarrow v \bmod 10^k$
        $ac \leftarrow \textsc{Karatsuba}\ (a,\ c,\ k)$
        $bd \leftarrow \textsc{Karatsuba}\ (b,\ d,\ k)$
        $y \leftarrow \textsc{Karatsuba}\ (a+b,\ c+d,\ k{+}1)$
        $x \leftarrow ac \cdot 10^{2k} + (y - ac - bd) \cdot 10^k + bd$
        devolva x

Consumo de tempo

- recorrência: $T(n) = 2\,T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + n$
- solução: $T(n)$ está em $\Theta(n^{\lg 3})$

Observações finais:

- é natural escrever o algoritmo em estilo recursivo
- é fácil ver que algoritmo está correto
- estimar consumo de tempo: difícil
- importante saber resolver recorrências

Segmento de soma máxima

Definições:

- segmento de um vetor: índices consecutivos
- soma de um segmento
- solidez de vetor: soma de um segmento não-vazio de soma máxima

p							r
20	−30	15	−10	30	−20	−30	30

solidez 35

Problema

Calcular a solidez de um vetor $A[p\,..\,r]$ de números inteiros.

Algoritmo trivial

Idéia:

- aplicação cega da definição
- examina todos os segmentos de $A[p\,.\,.\,r]$

Algoritmo 1: trivial

SolidezI (A, p, r)
1 $x \leftarrow A[r]$
2 para $q \leftarrow r-1$ decrescendo até p faça
3 $\quad s \leftarrow 0$
4 $\quad$ para $j \leftarrow q$ até r faça
5 $\quad\quad s \leftarrow s + A[j]$
6 $\quad\quad$ se $s > x$ então $x \leftarrow s$
7 devolva x

p			q	$q+1$			r		x
20	-30	15	-10	30	-20	-30	30		30

início de uma iteração (linha 2)

Algoritmo 1 está correto

Invariante: no início de cada iteração

- x é a solidez do vetor $A[q{+}1\,..\,r]$

Última iteração: x é a solidez de $A[p\,..\,r]$

Consumo de tempo

Tamanho do vetor: $n = r - p + 1$

- para cada q fixo, o bloco de linhas 5–6 é repetido $r - q + 1$ vezes
- número total de repetições do bloco 5–6:

  $\sum_{q=p}^{r-1}(r - q + 1) \;=\; \sum_{j=2}^{n} j \;=\; \frac{1}{2}\,(n^2 + n - 2)$
- consumo de tempo do algoritmo: $\Theta(n^2)$

Algoritmo 2: divisão e conquista

Idéia:

- semelhante a MERGESORT
- mas a fase de "conquista" é mais complexa

p			q	$q{+}1$			r
20	-30	15	-10	30	-20	-30	30

Algoritmo 2: divisão e conquista

SOLIDEZII $(A,\ p,\ r)$

se $p = r$
  então devolva $A[p]$
  senão $q \leftarrow \lfloor (p+r)/2 \rfloor$
    $x' \leftarrow$ SOLIDEZII $(A,\ p,\ q)$
    $x'' \leftarrow$ SOLIDEZII $(A,\ q+1,\ r)$
    calcula máximo $y'+y''$ de $\cdots + A[q] + A[q+1] + \cdots$
    $x \leftarrow \max\,(x',\ y'+y'', x'')$
    devolva x

p			q	$q+1$			r
20	−30	15	−10	30	−20	−30	30

calcula máximo $y'+y''$ *de* $\cdots+A[q]+A[q+1]+\cdots$:

$y' \leftarrow s \leftarrow A[q]$
para $i \leftarrow q-1$ decrescendo até p faça
    $s \leftarrow A[i] + s$
    se $s > y'$ então $y' \leftarrow s$
$y'' \leftarrow s \leftarrow A[q+1]$
para $j \leftarrow q+2$ até r faça
    $s \leftarrow s + A[j]$
    se $s > y''$ então $y'' \leftarrow s$

Algoritmo 2 está correto

Tamanho de uma instância: $n = r - p + 1$

- se $n = 1$ então $A[p]$ é solução
- agora suponha $n \geq 2$
- hipótese de indução: x' é solidez de $A[p \mathinner{.\,.} q]$
- etc.

Consumo de tempo

$T(n)$: consumo de tempo no pior caso

- recorrência: $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$
- solução: T está em $\Theta(n \lg n)$

Conclusão: algoritmo 2 é mais rápido que algoritmo 1

Algoritmo 3: programação dinâmica

- firmeza de $A[p\,..\,r]$: a maior soma da forma $A[i] + \cdots + A[r]$
- a solidez de $A[p\,..\,r]$ é o máximo das firmezas de todos os segmentos iniciais

p							r
20	−30	15	−10	30	−20	−30	30

firmeza $= 30$

Problema auxiliar

Calcular a firmeza de todos os segmentos iniciais de $A[p\,..\,r]$

- $F[q]$ é a firmeza do segmento inicial $A[p\,..\,q]$

	p							r
A	20	−30	15	−10	30	−20	−30	30
F	20	−10	15	5	35	15	−15	30

Propriedade recursiva:

- a firmeza de $A[p\,..\,q]$ "contém" a firmeza de $A[p\,..\,q{-}1]$
- $F[q] = \max\big(\, F[q{-}1] + A[q] \;,\; A[q] \,\big)$

Algoritmo 3: programação dinâmica

SolidezIII (A, p, r)

$F[p] \leftarrow A[p]$
para $q \leftarrow p+1$ até r faça
    $s \leftarrow F[q-1] + A[q]$
    se $s > A[q]$
        então $F[q] \leftarrow s$
        senão $F[q] \leftarrow A[q]$
$x \leftarrow F[p]$
para $q \leftarrow p+1$ até r faça
    se $F[q] > x$ então $x \leftarrow F[q]$
devolva x

Algoritmo 3 está correto

Invariante: a cada passagem pela linha 2

- $F[q{-}1]$ é a firmeza de $A[p\,..\,q{-}1]$
- $F[q{-}2]$ é a firmeza de $A[p\,..\,q{-}2]$
- etc.

Bloco 6–8 escolhe a maior das firmezas

Consumo de tempo

$T(n)$: consumo de tempo no pior caso

- T está em $\Theta(n)$

Conclusão: algoritmo 3 é mais rápido que algoritmo 2

Observações:

- algoritmo SOLIDEZIII é um exemplo de programação dinâmica
- (nada a ver com programação de computadores)
- uma tabela armazena soluções de subinstâncias
- o problema precisa ter *estrutura recursiva*:
  solução de uma instância contém soluções de subinstâncias
- consumo de tempo: proporcional ao tamanho da tabela

Mochila de valor máximo

Motivação:

- dado um conjunto de objetos e uma mochila
- cada objeto tem um peso e um valor
- problema: escolher um conjunto de objetos
  que tenha o maior valor possível
  mas não ultrapasse a capacidade da mochila

Exemplos:

- $p_i = v_i$: problema dos cheques
- $v_i = 1$: problema do pen drive

Definições:

- objetos $1, \ldots, n$
- números naturais $p_1, \ldots, p_n$ e $v_1, \ldots, v_n$
- p_i é o peso de i
- v_i é o valor de i
- o peso de um conjunto S de objetos é $\sum_{i \in S} p_i$
- o valor de S é $\sum_{i \in S} v_i$
- um conjunto S é viável se $\sum_{i \in S} p_i \leq M$

Problema da mochila

Dados números *naturais* $p_1, \ldots, p_n$, M, $v_1, \ldots, v_n$
encontrar um subconjunto viável de $\{1, \ldots, n\}$ que tenham valor máximo

- algoritmo trivial: tentar todos os subconjuntos $\{1, \ldots, n\}$
- consome tempo $\Omega(2^n)$
- inaceitável...

A estrutura recursiva do problema

Estrutura recursiva:

- seja S solução da instância (n, p, M, v)
- se $n \notin S$ então S é um solução da instância $(n-1,\ p,\ M,\ v)$
- se $n \in S$ então $S - \{n\}$ é solução de $(n-1,\ p,\ M - p_n,\ v)$

Recorrência:

- notação: $X(n, M)$ é valor de um solução
- $X(n, M) \;=\; \max \left\{ X(n-1, M)\,,\ X(n-1, M-p_n) + v_n \right\}$
- se $p_n > M$ então $X(n, M) \;=\; X(n-1, M)$

Algoritmo recursivo

Recorrência transformada em algoritmo:

- inaceitável: consumo de tempo $\Omega(2^n)$
- por que?
- refaz muitas vezes a solução das mesmas subinstâncias

Algoritmo de programação dinâmica

A recorrência abre as portas para a programação dinâmica:

- M e p_i são números *naturais*
- X é tabela indexada por $0\,..\,n \times 0\,..\,M$
- tabela deve ser preenchida na ordem "certa"

p	v
4	500
2	400
1	300
3	450

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	700	700	800
4	0	300	400	700	750	?

Mochila com $n = 4$ e $M = 5$

p	v
4	500
2	400
1	300
3	450

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	700	700	800
4	0	300	400	700	750	850

Mochila com $n = 4$ e $M = 5$

Algoritmo

MOCHILA $(n,\ p,\ M,\ v)$

1 para $L \leftarrow 0$ até M faça
2 $\quad X[0, L] \leftarrow 0$
3 $\quad$ para $m \leftarrow 1$ até n faça
4 $\quad\quad a \leftarrow X[m-1, L]$
5 $\quad\quad$ se $L - p_m \geq 0$
6 $\quad\quad\quad$ então $b \leftarrow X[m-1, L-p_m] + v_m$
7 $\quad\quad\quad\quad$ se $a < b$ então $a \leftarrow b$
8 $\quad\quad X[m, L] \leftarrow a$
9 devolva $X[n, M]$

O algoritmo está correto

Invariante: no começo de cada iteração (linha 1)

- as L primeiras colunas da tabela estão corretas

Consumo de tempo

- tempo para preencher uma casa da tabela: não depende de n nem M
- tempo total: proporcional ao número de casas da tabela
- tempo total: $\Theta(n\,M)$

- algoritmo lento...
- muito sensível às variações de M
- implicitamente adotamos (n, M) como tamanho de uma instância
- é mais razoável dizer que tamnaho é $(n, \lceil \lg M \rceil)$

Consumo de tempo: apresentação melhorada

Tamanho de uma instância: $(n, \lceil \lg M \rceil)$

- consumo de tempo: $\Theta(n\, 2^{\lceil \lg M \rceil})$
- algoritmo não é polinomial

Comentários:

- definição ideal de tamanho da instância: $2n + 1$
- eu gostaria de algoritmo $\mathrm{O}(n^2)$ ou até $\mathrm{O}(n^{100})$
- mas isso é pedir demais. . .

Mochila de valor quase máximo

Problema da mochila (de novo)

Dados naturais $p_1, \ldots, p_n$, M, $v_1, \ldots, v_n$
encontrar um subconjunto viável de $\{1, \ldots, n\}$ que tenham valor máximo

- não conheço algoritmo rápido
- que tal algoritmo que dá solução "aproximada"?
- algoritmo dá conjunto viável de valor $> 50\%$ do ótimo

Idéia:

- suponha $1 \leq p_i \leq M$
- suponha $\dfrac{v_1}{p_1} \geq \dfrac{v_2}{p_2} \geq \cdots \geq \dfrac{v_n}{p_n}$
- escolha o maior segmento inicial viável X de $\{1, \ldots, n\}$
- X é a resposta a menos que algum $\{v_i\}$ seja melhor

Algoritmo: 50% do máximo

$\textsc{MochilaAprox}\ (n,\ p,\ M,\ v)$

$s \leftarrow x \leftarrow 0$

$m \leftarrow 1$

enquanto $m \leq n$ e $s + p_m \leq M$ faça

  $s \leftarrow s + p_m$

  $x \leftarrow x + v_m$

  $m \leftarrow m + 1$

se $m > n$

  então devolva x

  senão devolva $\max\,(x, v_m)$

O algoritmo está correto

- seja $X = \{1, \ldots, m-1\}$
- seja $v(S) := \sum_{i \in S} v_i$ para qualquer S
- $\max\left(v(X), v_m\right) \;\geq\; \dfrac{v(X) + v_m}{2} \;=\; \frac{1}{2}\, v(X \cup \{m\})$
- $X \cup \{m\}$ é mais valioso que qualquer conjunto viável (veja abaixo)

Fato

$v(X \cup \{m\}) > v(S)$ para qualquer conjunto viável S

Prova:

- seja $Y = X \cup \{m\}$
- $$\begin{aligned} v(Y) - v(S) &= v(Y-S) - v(S-Y) \\ &= \textstyle\sum_{i \in Y-S} v_i - \sum_{i \in S-Y} v_i \\ &= \textstyle\sum_{i \in Y-S} \frac{v_i}{p_i}\, p_i - \sum_{i \in S-Y} \frac{v_i}{p_i}\, p_i \\ &\geq \tfrac{v_m}{p_m}\, p(Y-S) \;-\; \tfrac{v_m}{p_m}\, p(S-Y) \\ &= \tfrac{v_m}{p_m} \big(p(Y) - p(S)\big) \\ &> \tfrac{v_m}{p_m} \big(M - M\big) \\ &= 0 \end{aligned}$$

Consumo de tempo

Tamanho de uma instância: n

- o algoritmo consome $\Theta(n)$ unidades de tempo
- pré-processamento consome $\Theta(n \lg n)$
- consumo total: $\Theta(n \lg n)$

Observações finais:

- algoritmo de aproximação: idéia esperta mas natural
- análise da correção do algoritmo: não é obvia
- estimativa do consumo de tempo: fácil

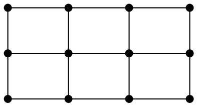
A cobertura de um grafo

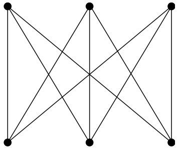
Motivação:

- dada rede de corredores de uma galeria de arte
- problema: encontrar o menor conjunto de sentinelas capaz de vigiar a rede

Grafos:

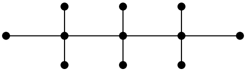
- um grafo é um par (V, A) de conjuntos
- V é um conjunto de vértices
- A é um conjunto de arestas
- cada aresta é um par não-ordenado ij de vértices
- i e j são as pontas da aresta ij
- o tamanho de um grafo é o par (n, m) sendo $n = |V|$ e $m = |A|$





Definições:

- uma cobertura de um grafo é um conjunto X de vértices que contém pelo menos uma das pontas de cada aresta
- se cada vértice i tem um custo c_i então o custo de uma cobertura X é $c(X) = \sum_{i \in X} c_i$



Problema da cobertura mínima

Encontrar uma cobertura de custo mínimo em um grafo cujos vértices têm custos em $\mathbb{N}$

Complexidade de problema

- tudo indica que não existe algoritmo polinomial
- mas existe um algoritmo polinomial para cobertura quase mínima
- cobertura X cujo custo é o dobro do ótimo
- ninguém descobriu ainda algoritmo polinomial com fator 1.9

Algoritmo: dobro da cobertura mínima

CoberturaBarata $(V,\ A,\ c)$

para cada i em V faça
    $x_i \leftarrow 0$
para cada ij em A faça
    $y_{ij} \leftarrow 0$
para cada pq em A faça
    $e \leftarrow \min\left(c_p - x_p\ ,\ c_q - x_q\right)$
    $y_{pq} \leftarrow y_{pq} + e$
    $x_p \leftarrow x_p + e$
    $x_q \leftarrow x_q + e$
$X \leftarrow \emptyset$
para cada i em V faça
    se $x_i = c_i$ então $X \leftarrow X \cup \{i\}$
devolva X

Invariantes

No início de cada iteração do bloco de linhas 6-9

i. $x_i = \sum_j y_{ij}$ para todo i em V

ii. $x_i \leq c_i$ para todo i em V

iii. para toda aresta ij já examinada tem-se $x_i = c_i$ ou $x_j = c_j$

O algoritmo está correto

X é uma cobertura e

$$c(X) \ \leq \ 2 \sum_{ij \in A} y_{ij} \ \leq \ 2\, c(Z)$$

para qualquer cobertura Z

Prova do segundo "$\leq$":

$$\begin{array}{rcll} \sum_{ij \in A} y_{ij} & \leq & \sum_{i \in Z} \sum_j y_{ij} & (i \in Z \text{ ou } j \in Z) \\ & = & \sum_{i \in Z} x_i & \text{(invariante i)} \\ & \leq & \sum_{i \in Z} c_i & \text{(invariante ii)} \end{array}$$

Prova do primeiro "$\leq$":

$$\begin{array}{rcll} \sum_{i \in X} c_i & = & \sum_{i \in X} x_i & (c_i = x_i) \\ & = & \sum_{i \in X} \sum_j y_{ij} & \text{(invariante i)} \\ & \leq & 2 \sum_{ij \in A} y_{ij} & \text{(aresta só tem 2 pontas)} \end{array}$$

Consumo de tempo

Tamanho de instância: (n, m)

- consumo total: $\Theta(n + m)$

Fim

Obrigado pela atenção!