

AULA 21

Máximo divisor comum

CLRS 31.1 e 31.2

Divisibilidade

Suponha que a , b e d são números inteiros.

Dizemos que d **divide** a se $a = k d$ para algum número inteiro k .

$d \mid a$ é uma abreviação de “ d divide a ”

Se d divide a , então dizemos que a é um **múltiplo** de d .

Se d divide a e $d > 0$, então dizemos que d é um **divisor** de a

Se d divide a e d divide b , então d é um **divisor comum** de a e b .

Exemplo:

os divisores de 30 são: $1, 2, 3, 5, 6, 10, 15$ e 30

os divisores de 24 são: $1, 2, 3, 4, 6, 8, 12$ e 24

os divisores comuns de 30 e 24 são: $1, 2, 3$ e 6

Máximo divisor comum

O **máximo divisor comum** de dois números inteiros a e b , onde pelo menos um é não nulo, é o maior divisor comum de a e b .

O máximo divisor comum de a e b é denotado por $\text{mdc}(a, b)$.

Exemplo:

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

Problema: Dados dois números inteiros não-negativos a e b , determinar $\text{mdc}(a, b)$.

Café com leite

Recebe números inteiros não-negativos a e b e devolve $\text{mdc}(a, b)$.

Café-Com-Leite (a, b) \triangleright **supõe** $a \neq 0$ ou $b \neq 0$

```
1  se  $b = 0$  então devolva  $a$ 
2  se  $a = 0$  então devolva  $b$ 
3   $d \leftarrow b$ 
4  enquanto  $d \nmid a$  ou  $d \nmid b$  faça
5       $d \leftarrow d - 1$ 
6  devolva  $d$ 
```

Relações invariantes: na linha 4 vale que

- (i0) na linha 4 vale que $1 \leq d \leq b$;
- (i1) na linha 4 vale que $k \nmid a$ ou $k \nmid b$ para cada $k > d$; e
- (i2) na linha 5 vale que $d \nmid a$ ou $d \nmid b$.

Consumo de tempo

linha	consumo de todas as execuções da linha
-------	---

1-2	$\Theta(1)$
-----	-------------

3	$\Theta(1)$
---	-------------

4	$O(b)$
---	--------

5	$O(b)$
---	--------

6	$\Theta(1)$
---	-------------

total $\Theta(3) + O(b) = O(b)$

Quando a e b são **relativamente primos**, ou seja $\text{mdc}(a, b) = 1$, o consumo de tempo do algoritmo é $\Theta(b)$.

Conclusão

O consumo de tempo do algoritmo **Café-Com-Leite** é $O(b)$.

No pior caso, o consumo de tempo do algoritmo **Café-Com-Leite** é $\Theta(b)$.

Se o **valor** de b **dobra**, o consumo de tempo pode **dobrar**.

Seja $\beta := \langle b \rangle$ o número de bits ou **tamanho** de b .

O consumo de tempo do algoritmo **Café-Com-Leite** é $O(2^\beta)$.

Brincadeira

Usei uma implementação do algoritmo **Café-Com-Leite** para determinar $\text{mdc}(2147483647, 2147483646)$.

Vejam quanto tempo gastou:

```
meu_prompt> time mdc 2147483647 2147483646
mdc: mdc de 2147483647 e 2147483646 e' 1.
```

```
real      2m49.306s
user      1m33.412s
sys       0m0.099s
```


Algoritmo de Euclides

Recebe números inteiros não-negativos a e b e devolve $\text{mdc}(a, b)$.

EUCLIDES (a, b) \triangleright **supõe** $a \neq 0$ ou $b \neq 0$

1 **se** $b = 0$

2 **então devolva** a

3 **senão devolva** **EUCLIDES** ($b, a \bmod b$)

“ $a \bmod b$ ” é o resto da divisão de a por b .

Exemplo: $\text{mdc}(12, 18) = 6$, pois

$\text{mdc}(12, 18)$

$\text{mdc}(18, 12)$

$\text{mdc}(12, 6)$

$\text{mdc}(6, 0)$

Correção

A correção do algoritmo **EUCLIDES** é baseado no seguinte fato.

Suponha que $a \geq 0$ e $b > 0$. Para cada $d > 0$
vale que

$d \mid a$ e $d \mid b$ se e só se $d \mid b$ e $d \mid a \bmod b$.

Em outras palavras, os pares (a, b) e $(b, a \bmod b)$ têm os mesmos divisores.

Outro exemplo

`mdc (317811 , 514229)`

`mdc (514229 , 317811)`

`mdc (317811 , 196418)`

`mdc (196418 , 121393)`

`mdc (121393 , 75025)`

`mdc (75025 , 46368)`

`mdc (46368 , 28657)`

`mdc (28657 , 17711)`

`mdc (17711 , 10946)`

`mdc (10946 , 6765)`

`mdc (6765 , 4181)`

`mdc (4181 , 2584)`

`mdc (2584 , 1597)`

`mdc (1597 , 987)`

`mdc (987 , 610)`

`mdc (610 , 377)`

Outro exemplo (cont.)

$\text{mdc}(377, 233)$

$\text{mdc}(233, 144)$

$\text{mdc}(144, 89)$

$\text{mdc}(89, 55)$

$\text{mdc}(55, 34)$

$\text{mdc}(34, 21)$

$\text{mdc}(21, 13)$

$\text{mdc}(13, 8)$

$\text{mdc}(8, 5)$

$\text{mdc}(5, 3)$

$\text{mdc}(3, 2)$

$\text{mdc}(2, 1)$

$\text{mdc}(1, 0)$

$\text{mdc}(317811, 514229) = 1$

Mais brincadeirainha

Usei uma implementação do algoritmo **EUCLIDES** para determinar $\text{mdc}(2147483647, 2147483646)$.

Vejam quanto tempo gastou:

```
meu_prompt> time euclides 2147483647 2147483646
mdc(2147483647,2147483646)
    mdc(2147483646,1)
        mdc(1,0)
euclides: mdc de 2147483647 e 2147483646 e' 1

real        0m0.007s
user        0m0.002s
sys         0m0.004s
```

Consumo de tempo

O consumo de tempo do algoritmo **EUCLIDES** é proporcional ao **número de chamadas recursivas**.

Suponha que a função **EUCLIDES** faz k chamadas recursivas e que na 1a. chamada ao algoritmo tem-se que $a \geq b > 0$.

Sejam

$$(a, b) = (a_0, b_0), (a_1, b_1), \dots, (a_k, b_k) = (\text{mdc}(a, b), 0)$$

os valores dos parâmetros no início de cada chamada.

Portanto, que $a_{i+1} = b_i$ e $b_{i+1} = a_i \bmod b_i$ para $i = 1, 2, \dots, k$.

Número de chamadas recursivas

Para números inteiros p e q , $p \geq q > 0$ vale que

$$p \bmod q < p/2$$

Desta forma,

$$\begin{array}{ccccccc} b_2 & = & a_1 \bmod b_1 & = & b_0 \bmod b_1 & < & b_0/2 \leq b/2^1 \\ b_4 & = & a_3 \bmod b_3 & = & b_2 \bmod b_3 & < & b_2/2 \leq b/2^2 \\ b_6 & = & a_5 \bmod b_5 & = & b_4 \bmod b_5 & < & b_4/2 \leq b/2^3 \\ b_8 & = & a_7 \bmod b_7 & = & b_6 \bmod b_7 & < & b_6/2 \leq b/2^4 \\ \vdots & = & \vdots & = & \vdots & < & \vdots \leq \vdots \end{array}$$

O valor do 2o. parâmetro é reduzido a menos da sua metade a cada 2 chamadas recursivas.

Número de chamadas recursivas

Seja t o número inteiro tal que

$$2^t \leq b < 2^{t+1},$$

ou seja, $t = \lfloor \lg b \rfloor$.

Da desigualdade estrita concluimos que o número de chamadas recursivas é $\leq 2\lfloor \lg b \rfloor + 1$.

Por exemplo, para $a = 514229$ e $b = 317511$ temos que

$$2 \lg(b) + 1 = 2 \lg(317511) + 1 < 2 \times 18.3 + 1 = 37.56$$

e o número de chamadas recursivas feitas por **EUCLIDES** ($514229, 317511$) é **27**.

Conclusões

O consumo de tempo do algoritmo EUCLIDES é $O(\lg b)$.

Seja $\beta := \langle b \rangle$ o número de bits ou tamanho de b .

O consumo de tempo do algoritmo EUCLIDES é $O(\beta)$.

Se o valor de β dobra, o consumo de tempo pode dobrar.

Se o tamanho de b dobra, o consumo de tempo pode dobrar.

$$b \times \lg b$$

b	$\lfloor \lg b \rfloor$
4	2
5	2
6	2
10	3
64	6
100	6
128	7
1000	9
1024	10
1000000	19
1000000000	29
\vdots	\vdots

Números de Fibonacci

$$F_0 = 0 \quad F_1 = 1 \quad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Algoritmo recursivo para F_n :

FIBO-REC (n)

```
1  se  $n \leq 1$ 
2      então devolva  $n$ 
3      senão  $a \leftarrow$  FIBO-REC ( $n - 1$ )
4               $b \leftarrow$  FIBO-REC ( $n - 2$ )
5      devolva  $a + b$ 
```

Euclides e Fibonacci

Os números de **Fibonacci** estão intimamente relacionados com o algoritmo de **Euclides**.

Verifique que

A chamada **EUCLIDES** (F_{k+2}, F_{k+1}) faz k chamadas recursivas.

Verifique ainda que

Se $a > b \geq 0$ e se **EUCLIDES** (a, b) faz $k \geq 1$ chamadas recursivas, então

$$a \geq F_{k+2} \quad \text{e} \quad b \geq F_{k+1}.$$

Euclides e Fibonacci

Uma consequência imediata do fato anterior é:

Para todo inteiro $k \geq 1$, se $a > b \geq 0$ e $b < F_{k+1}$, então **EUCLIDES** (a, b) faz **menos de** k chamadas recursivas.

Exemplo:

$F_{29} = 514229$ e $F_{28} = 317811$ e **Euclides**(**514229**,**317811**) faz **27** chamadas recursivas.

Razão Aurea

Suponha que um **passarinho** me contou que para todo $t \geq 2$ vale que

$$\phi^{t-2} \leq F_t < \phi^{t-1},$$

onde $\phi = (1 + \sqrt{5})/2$ que é um número entre 1.618 e 1.619.

Exemplo:

$$\phi^{26} < 1.619^{26} < 275689 < F_{28} = 317811 < 438954 < 1.618^{27} < \phi^{27}$$

Verifique que $1 + \phi = \phi^2$.

Número de chamadas (novamente)

Suponha $b \geq 4$. Se t é o número inteiro tal que

$$\phi^{t-2} \leq b < \phi^{t-1} < F_{t+1}$$

então o número k de chamadas recursivas de **EUCLIDES** (a, b) é no máximo

$$t - 1 \leq (2 + \log_{\phi} b) - 1 = 1 + \log_{\phi} b.$$

Exemplo: Segundo esta nova estimativa temos que **EUCLIDES**(514229, 317811) faz no máximo

$$1 + \log_{\phi}(317811) < 1 + \log_{1.619}(317811) < 1 + 26.28 = 27.45$$

e o número de chamadas é 27.

[Uauuuu. Isto foi muito perto! Talvez tenha alguma conta errada.]

Conclusão

Se k é número de chamadas recursivas feitas por
EUCLIDES (a, b) então

$$k \leq 1 + \log_{\phi} b, \text{ onde } \phi = (1 + \sqrt{5})/2.$$

Certificados

O algoritmo **EUCLIDES** pode ser facilmente modificado para nos fornecer números inteiros x e y tais que

$$ax + by = \text{mdc}(a, b).$$

Certificados

O algoritmo **EUCLIDES** pode ser facilmente modificado para nos fornecer números inteiros x e y tais que

$$ax + by = \text{mdc}(a, b).$$

E daí? Qual a graça nisto?

Certificados

O algoritmo **EUCLIDES** pode ser facilmente modificado para nos fornecer números inteiros x e y tais que

$$ax + by = \text{mdc}(a, b).$$

E daí? Qual a graça nisto?

Os números x e y são uma **prova** ou **certificado** da correção da resposta!

Certificados

Suponha que **EUCLIDES** (a, b) devolva d junto com x, y tais que $ax + by = d$

- podemos verificar se $d \mid a$ e $d \mid b$
- podemos verificar se $ax + by = d$

Se $d' \mid a$ e $d' \mid b$ então

$$d' \mid (ax + by) = d$$

e portanto $d' \leq d$

Conclusão: $d = \text{mdc}(a, b)$

Extended-Euclides

Recebe números inteiros não-negativos a e b e devolve números inteiros d , x e y tais que $d \mid a$, $d \mid b$ e $ax + by = d$.

EXTENDED- EUCLIDES (a, b) \triangleright **supõe** $a \neq 0$ **ou** $b \neq 0$

1 **se** $b = 0$

2 **então devolva** ($a, 1, 0$)

3 $(d', x', y') \leftarrow$ **EXTENDED- EUCLIDES** ($b, a \bmod b$)

4 $(d, x, y) \leftarrow (d', y', x' - \lfloor a/b \rfloor y')$

5 **devolva** (d, x, y)

Self-certifying algorithms

Algoritmos que devolvem certificados da sua correção são chamados *self-certifying*.

Exemplos:

- EXTENDED- EUCLIDES
- Algoritmo de Dijkstra (caminhos mínimos)
- Algoritmo de Ford e Fulkerson (fluxos em redes)
- Algoritmos para programação linear devolvem soluções do problema primal e dual

Na página pessoal de Kurt Mehlhorn há um link para uma palestra sobre *Certifying Algorithms*.

Cópia local:

<http://www.ime.usp.br/~coelho/mac5711-2006/aulas/CertifyingAlgs.pdf>

Complex. computacional: **P** versus NP

CLR 36 ou CLRS 34

Complexidade computacional

Classifica os problemas em relação à dificuldade de resolvê-los algoritmicamente.

Disciplina:

MAC5722 Complexidade Computacional

Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma **seqüência de símbolos** retirados de algum **alfabeto**.

Este alfabeto pode ser, por exemplo, o conjunto de símbolos **ASCII** ou o conjunto $\{0, 1\}$.

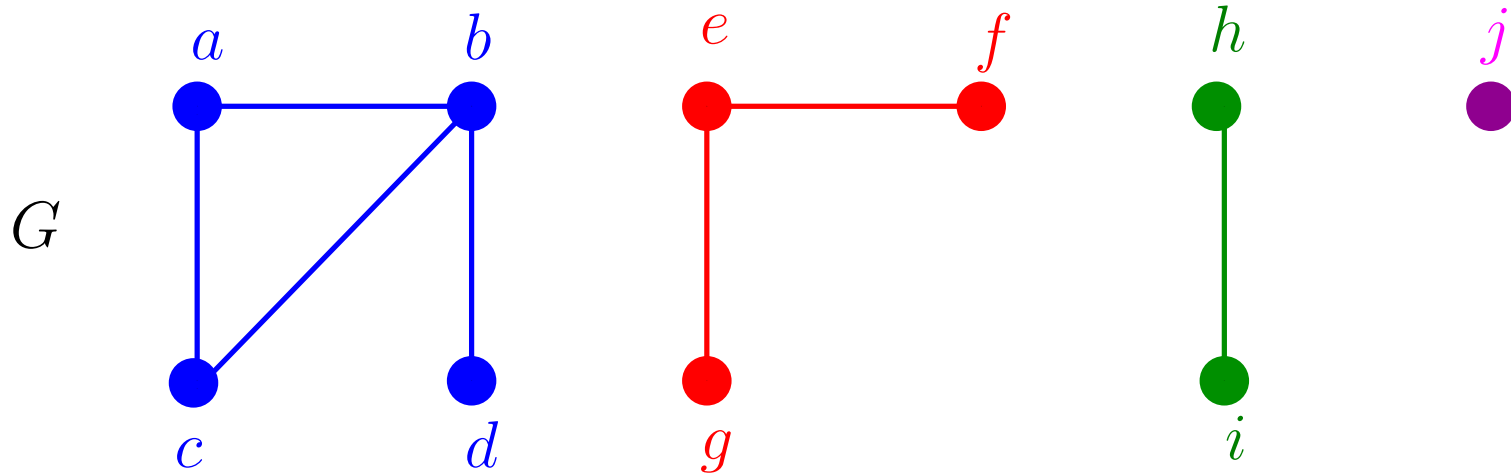
Qualquer seqüência dos elementos de um alfabeto é chamada de uma **palavra**.

Não é difícil codificar objetos tais como **racionais, vetores, matrizes, grafos e funções** como palavras.

O **tamanho** de uma palavra w , denotado por $\langle w \rangle$ é o número de símbolos usados em w , contando multiplicidades. O tamanho do racional '123/567' é **7**.

Exemplo 1

Grafo



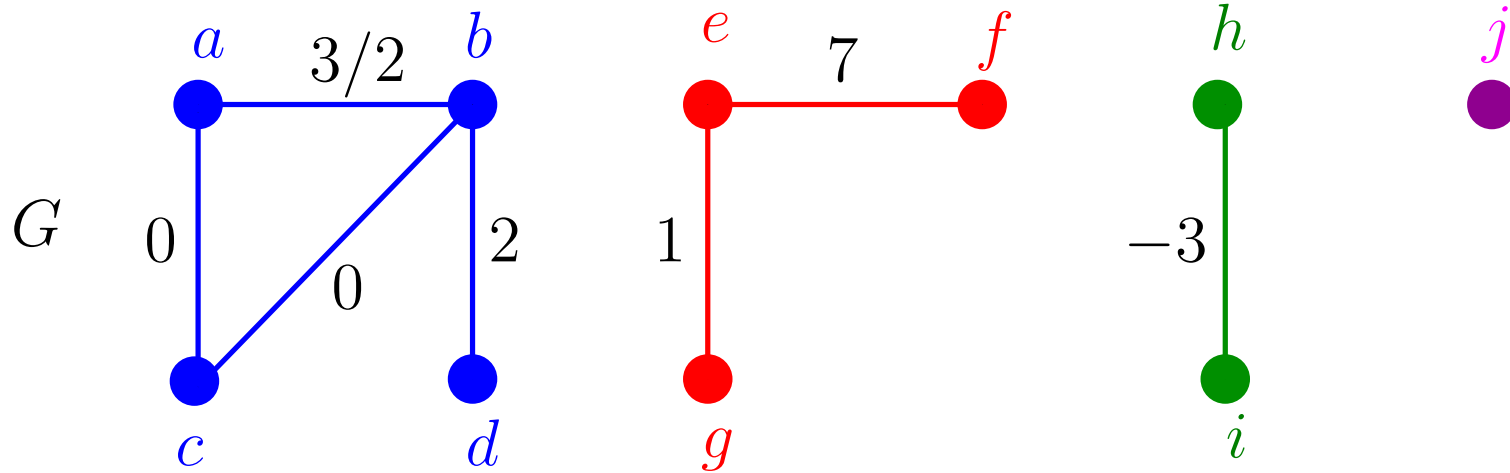
Palavra:

$(\{a, b, c, d, e, f, g, h, i, j\}, \{\{bd\}, \{eg\}, \{ac\}, \{hi\}, \{ab\}, \{ef\}, \{bc\}\})$

Tamanho da palavra: 59

Exemplo 2

Função



Palavra:

$((\{bd\}, 2), (\{eg\}, 1), (\{ac\}, 0), (\{hi\}, -3), (\{ab\}, 3/2), (\{ef\}, 7), (\{bc, 0\}))$

Tamanho da palavra: 67

Tamanho de uma palavra

Para os nossos propósitos, não há mal em subestimar o tamanho de um objeto.

Não é necessário contar rigorosamente os caracteres ‘{’, ‘}’, ‘(’, ‘)’ e ‘,’ dos exemplos anteriores.

Tamanho de um inteiro p é essencialmente $\lg |p|$.

Tamanho do racional p/q é, essencialmente, $\lg |p| + \lg |q|$.

Tamanho de um vetor $A[1..n]$ é a soma dos tamanhos de seus componentes

$$\langle A \rangle = \langle A[1] \rangle + \langle A[2] \rangle + \cdots + \langle A[n] \rangle.$$

Problemas e instâncias

Cada conjunto específico de dados de um problema define uma **instância**.

Tamanho de uma instância é o tamanho de uma palavra que representa a instância.

Problema que pede uma resposta do tipo **SIM** ou **NÃO** é chamado de **problema de decisão**.

Problema que procura um elemento em um conjunto é um **problema de busca**.

Problema que procura um elemento de um conjunto de soluções viáveis que seja **melhor possível** em relação a algum critério é um **problema de otimização**.

Máximo divisor comum

Problema: Dados dois números inteiros não-negativos a e b , determinar $\text{mdc}(a, b)$.

Exemplo:

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

Problema de busca

Instância: a e b

Tamanho da instância: $\langle a \rangle + \langle b \rangle$, essencialmente

$$\lg a + \lg b$$

Consumo de tempo do algoritmo **Café-Com-Leite** é $O(b)$.

Consumo de tempo do algoritmo **EUCLIDES** é $O(\lg b)$.

Máximo divisor comum (decisão)

Problema: Dados dois números inteiros não-negativos a , b e k , $\text{mdc}(a, b) = k$?

Exemplo:

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

Problema de decisão: resposta SIM ou NÃO

Instância: a , b , k

Tamanho da instância: $\langle a \rangle + \langle b \rangle + \langle k \rangle$, essencialmente

$$\lg a + \lg b + \lg k$$

Subseqüência comum máxima

Problema: Encontrar uma **ssco máxima** de $X[1..m]$ e $Y[1..n]$.

Exemplos: $X = A \text{ B C B D A B}$

$Y = \text{B D C A B A}$

ssco máxima = B C A B

Problema de otimização

Instância: $X[1..m]$ e $Y[1..n]$

Tamanho da instância: $\langle X \rangle + \langle Y \rangle$, essencialmente

$$n + m$$

Consumo de tempo **REC-LCS-LENGTH** é $\Omega(2^{\min\{m,n\}})$.

Consumo de tempo **LCS-LENGTH** é $\Theta(mn)$.

Subseqüência comum máxima (decisão)

Problema: $X[1 \dots m]$ e $Y[1 \dots n]$ possuem uma sscó máxima $\geq k$?

Exemplo: $X = A \text{ B C B D A B}$

$Y = \text{B D C A B A}$

ssco máxima = B C A B

Problema de decisão: resposta SIM ou NÃO

Instância: $X[1 \dots m]$, $Y[1 \dots n]$, k

Tamanho da instância: $\langle X \rangle + \langle Y \rangle + \langle k \rangle$, essencialmente

$$n + m + \lg k$$

Problema booleano da mochila

Problema (Knapsack Problem): Dados n , $w[1..n]$ $v[1..n]$ e W , encontrar uma **mochila booleana ótima**.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	0	1	1	0

valor = 1000

Problema de otimização

Instância: n , $w[1..n]$ $v[1..n]$ e W

Tamanho da instância: $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle$,
essencialmente $\lg n + n \lg W + n \lg V + \lg W$

Consumo de tempo MOCHILA-BOOLEANA é $\Theta(nW)$.

Problema booleano da mochila (decisão)

Problema (Knapsack Problem): Dados n , $w[1..n]$ $v[1..n]$ e W e k , existe uma **mochila booleana** de valor $\geq k$.

Exemplo: $W = 50$, $n = 4$, $k = 1010$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	0	1	1	0

valor = 1000

Problema de decisão: resposta **SIM** ou **NÃO**

Instância: n , $w[1..n]$ $v[1..n]$, W e k

Tamanho da instância: $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle + \lg k$,
essencialmente $\lg n + n \lg W + n \lg V + \lg W + \lg k$

Problema fracionário da mochila

Problema: Dados n , $w[1..n]$, $v[1..n]$ e W , encontrar uma mochila ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	1	1/3	0	0

valor = 1040

Problema de otimização

Instância: n , $w[1..n]$, $v[1..n]$ e W

Tamanho da instância: $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle$,
essencialmente $\lg n + n \lg W + n \lg V + \lg W$

Consumo de tempo MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$.

Problema fracionário da mochila (decisão)

Problema: Dados n , $w[1..n]$, $v[1..n]$, W e k , existe uma mochila de valor $\geq k$?

Exemplo: $W = 50$, $n = 4$, $k = 1010$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	1	1/3	0	0

valor = 1040

Problema de decisão: resposta SIM ou NÃO

Instância: n , $w[1..n]$, $v[1..n]$ e W

Tamanho da instância: $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle$,
essencialmente $\lg n + n \lg W + n \lg V + \lg W$

Modelo de computação

É uma descrição abstrata e conceitual de um computador que será usado para executar um algoritmo.

Um modelo de computação especifica as **operações elementares** um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome.

Operações elementares típicas são operações aritméticas entre números e comparações.

No **critério uniforme** supõe-se que cada operação elementar consome uma **quantidade de tempo constante**.

Problemas polinomiais

Análise de um algoritmo em um determinado modelo de computação estima o seu **consumo de tempo** e **quantidade de espaço** como uma função do **tamanho da instância do problema**.

Exemplo: o consumo de tempo do algoritmo **EUCLIDES** (a, b) é expresso como uma função de $\langle a \rangle + \langle b \rangle$.

Um problema é **solúvel em tempo polinomial** se existe um algoritmo que consome tempo $O(\langle I \rangle^c)$ para resolver o problema, onde c é uma constante e I é instância do problema.

Exemplos

- Máximo divisor comum

Tamanho da instância: $\lg a + \lg b$

Consumo de tempo Café-Com-Leite é $O(b)$
(não-polinomial)

Consumo de tempo EUCLIDES é $O(\lg b)$ (polinomial)

- Subseqüência comum máxima

Tamanho da instância: $n + m$

Consumo de tempo REC-LEC-LENGTH é $\Omega(2^{\min\{m,n\}})$
(não-polinomial)

Consumo de tempo LEC-LENGTH é $\Theta(mn)$
(polinomial).

Mais exemplos

- Problema booleano da mochila

Tamanho da instância: $\lg n + n \lg W + n \lg V + \lg W$

Consumo de tempo MOCHILA-BOOLEANA é $\Theta(nW)$
(não-polinomial)

- Problema fracionário da mochila

Tamanho da instância: $\lg n + n \lg W + n \lg V + \lg W$

Consumo de tempo MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$
(polinomial).

- Ordenação de inteiros $A[1..n]$

Tamanho da instância: $n \lg M$,

$M := \max\{|A[1]|, |A[2]|, \dots, |A[n]|\} + 1$

Consumo de tempo MERGE-SORT é $\Theta(n \lg n)$
(polinomial).

Classe P

Por um **algoritmo eficiente** entendemos um **algoritmo polinomial**.

A classe de todos os problemas de **decisão** que podem ser resolvidos por **algoritmos polinomiais** é denotada por **P** (classe de complexidade).

Exemplo: As versões de decisão dos problemas:
máximo divisor comum, subsequência comum
máxima e mochila fracionária
estão em **P**.

Para muitos problemas, **não se conhece** algoritmo essencialmente melhor que “testar todas as possibilidades”. Em geral, isso **não** está em **P**.