1. Algoritmo e conceitos básicos

(referência básica: capítulo 1 do livro texto)

Referências adicionais:

. Jayme L. Szwarcfiter, Grafos e Algoritmos Computacionais, Ed. Campus, 1986. ISBN 85-7001-341-8

. Cláudio L. Lucchesi et all, Aspectos Teóricos da Computação, IMPA, CNPq, 1979

. Notes on Complexity and NP Completeness. Web endereço (03/2007): www.cs.umd.edu/class/spring2003/cmsc351/notes/complexity.html

.Wikipidia (03/2007) palavras chave: Reduction (complexity) e NP-Complete.

Algoritmo

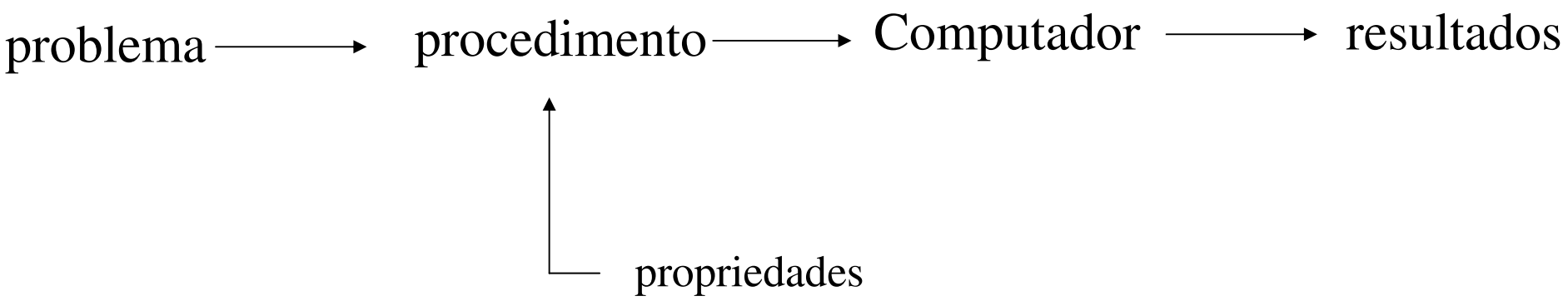
Realização de cálculos com grandezas → USO DO COMPUTADOR →

- Controle
- Processamento de textos
- Processamento de Imagens
- Geração de som e imagens

O que há de comum em todas as aplicações de computadores ?

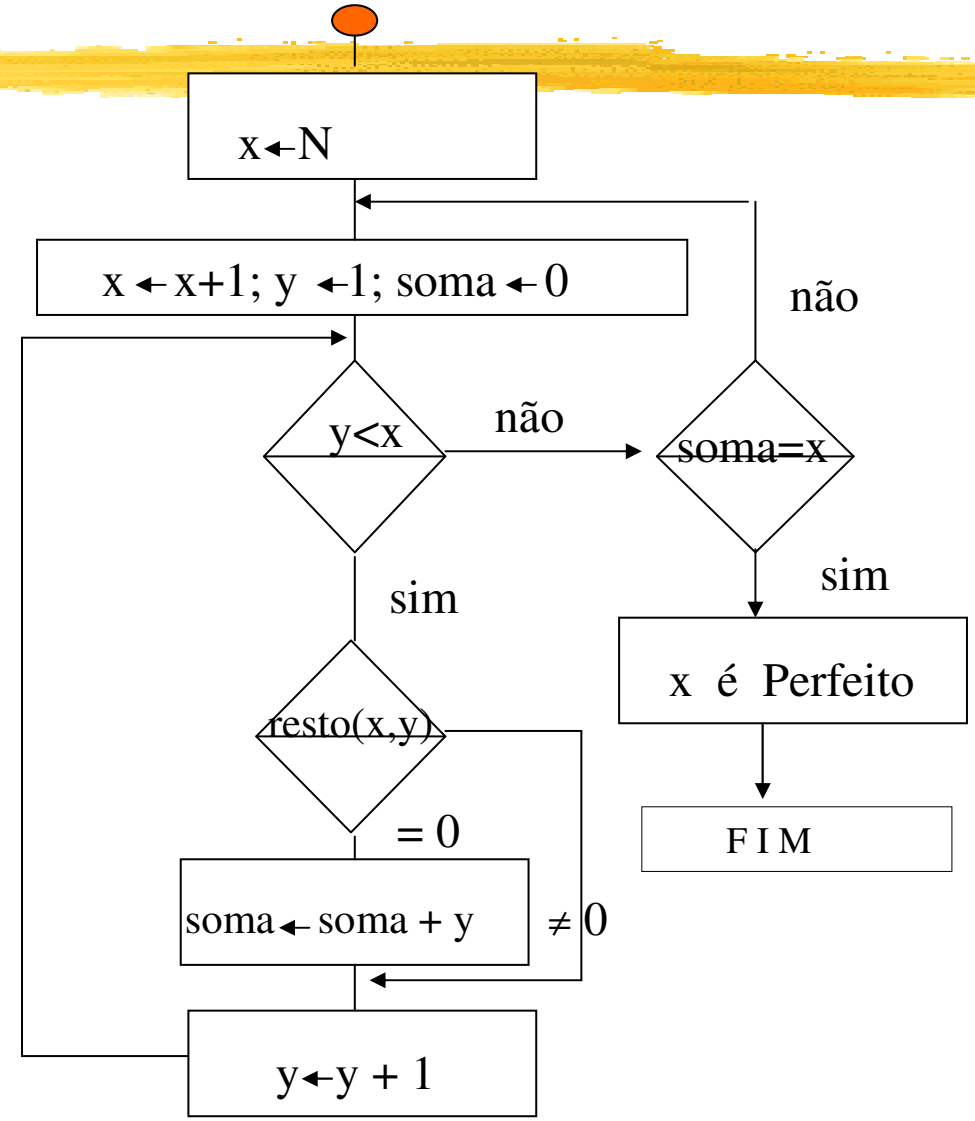
1.1 Procedimentos

- Procedimentos são executados para obter-se resultados



Propriedades de interesse em procedimentos em computadores

Cálculo do menor Número Perfeito maior do que N



Número Perfeito M: soma dos divisores de M igual a M

Número Perfeito

Seja dado o número 5. Verificar se 6, 7, 8, ... são Perfeitos.

- Divisores de 6: 1, 2, 3 (exceto 6)
- Soma dos divisores: 1 + 2 + 3 = 6
- Então 6 é um **Número Perfeito**.

Aplique o procedimento dado !

Propriedades essenciais de um procedimento

1. a descrição do procedimento deve ser finita, composta por uma seqüência finita de palavras e símbolos que descrevem o procedimento;
2. há sempre um conjunto de dados de entrada e pode ser previsto um resultado ou um conjunto de resultados, como o Número Perfeito;
3. algum agente pode executar o procedimento, tratando e armazenando resultados intermediários e finais;
4. toda instrução, passo atômico do procedimento, deve estar bem definida e ser passível de execução em tempo finito.

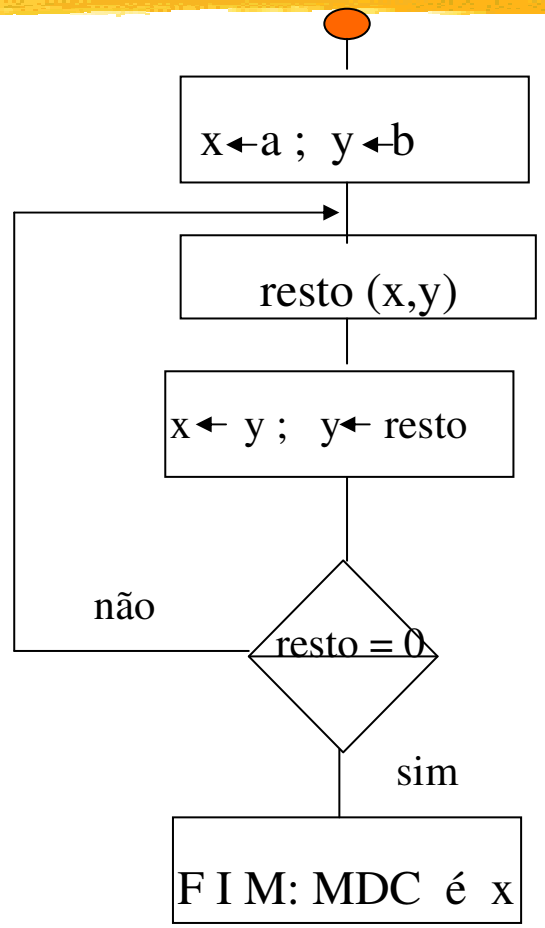
Algoritmo de Euclides para o cálculo de MDC de dois números

Sejam a=12 e b=8

- resto(12,8) : 4
- resto(8,4) : 0 >>>> FIM (MDC=4)

Sejam a=12 e b=7

- resto(12,7) : 5
- resto(7,5) : 2
- resto(5,2) : 1
- resto(2,1) : 0 >>>> FIM (MDC=1)



Propriedade da Terminação

- Para o cálculo do MDC é possível provar que o procedimento apresentado sempre termina (obtém um resultado).

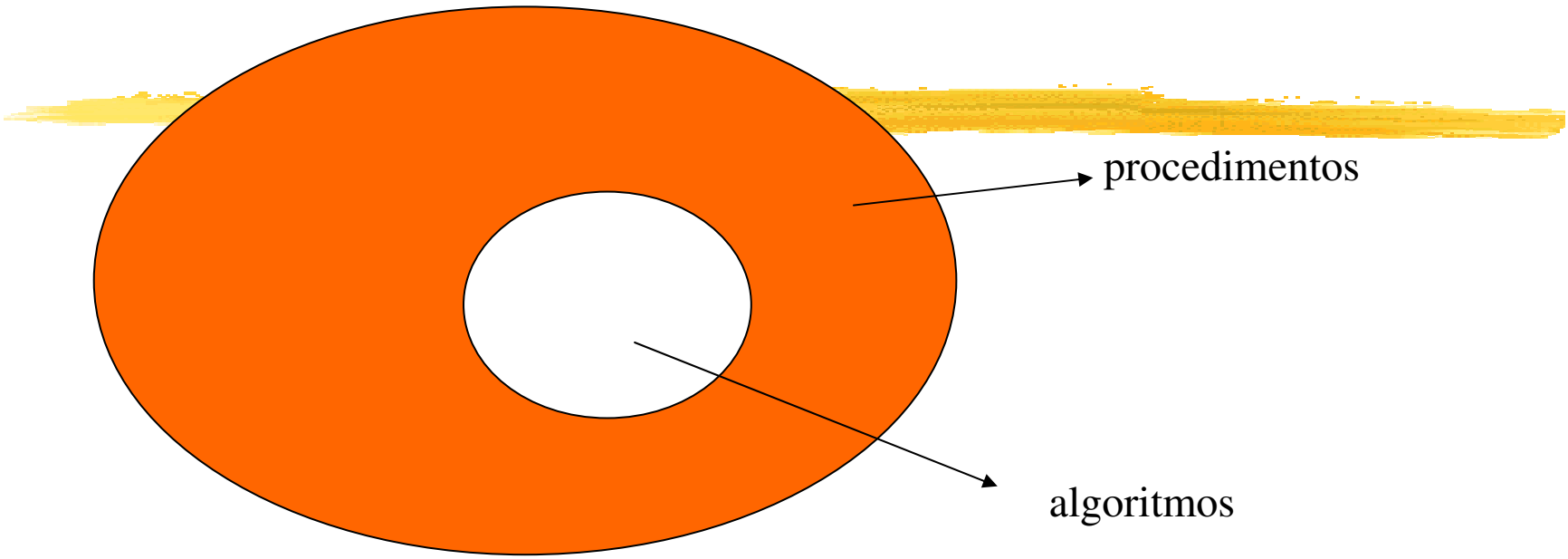
- O mesmo já não ocorre para o procedimento do Número Perfeito; a existência de uma quantidade finita / infinita de números perfeitos ainda é um problema em aberto.

5. Qualquer algoritmo deve sempre terminar após um número finito de passos

Procedimentos e Algoritmos

1. a descrição do procedimento deve ser finita, composta por uma seqüência finita de palavras e símbolos que descrevem o procedimento;
2. há sempre um conjunto de dados de entrada e pode ser previsto um resultado ou um conjunto de resultados, como o Número Perfeito;
3. algum agente pode executar o procedimento, tratando e armazenando resultados intermediários e finais;
4. toda instrução, passo atômico do procedimento, deve estar bem definida e ser passível de execução em tempo finito;
5. qualquer algoritmo deve sempre terminar após um número finito de passos.

Procedimentos: 1- 4 | Algoritmos : 1 - 5



Conclusão: computadores **processam** algoritmos.

1.2 Computabilidade e Complexidade

Passos na resolução de um problema

- **Definição** do algoritmo
- Verificação da **factibilidade** da solução
- Programação do algoritmo
- Execução do programa

Computabilidade e Complexidade

Passos na resolução de um problema

- **Definição** do algoritmo → Computabilidade ou Decidibilidade
- Verificação da **factibilidade** da solução
- Programação do algoritmo
- Execução do programa

Computabilidade e Complexidade

Passos na resolução de um problema

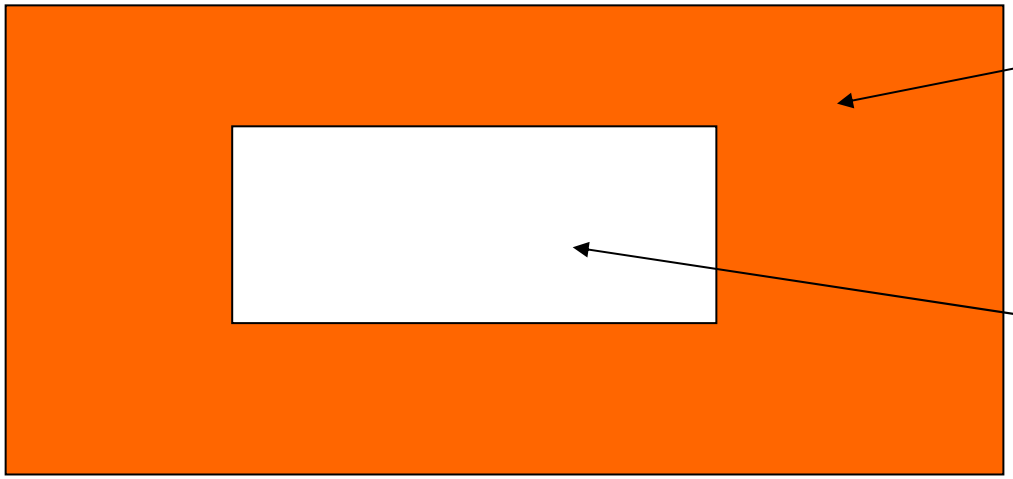
- **Definição** do algoritmo
- Verificação da **factibilidade** da solução
- Programação do algoritmo
- Execução do programa

Computabilidade
ou
Decidibilidade

Complexidade do algoritmo

Computabilidade

Refere-se **à existência ou não** de um algoritmo para a resolução de um determinado problema.



Não existe algoritmo para decidir sobre a **computabilidade** de qualquer problema bem formulado. ← Entscheidungsproblem

Exemplos clássicos de problemas não-computáveis

- Último teorema de Fermat :

para $k > 2$ e x, y, z inteiros positivos, não existem x, y, z, k tais que:

$$x^k + y^k = z^k$$

Provado em 1995 → **é um procedimento**

Exemplos clássicos de problemas não-computáveis (continuação)

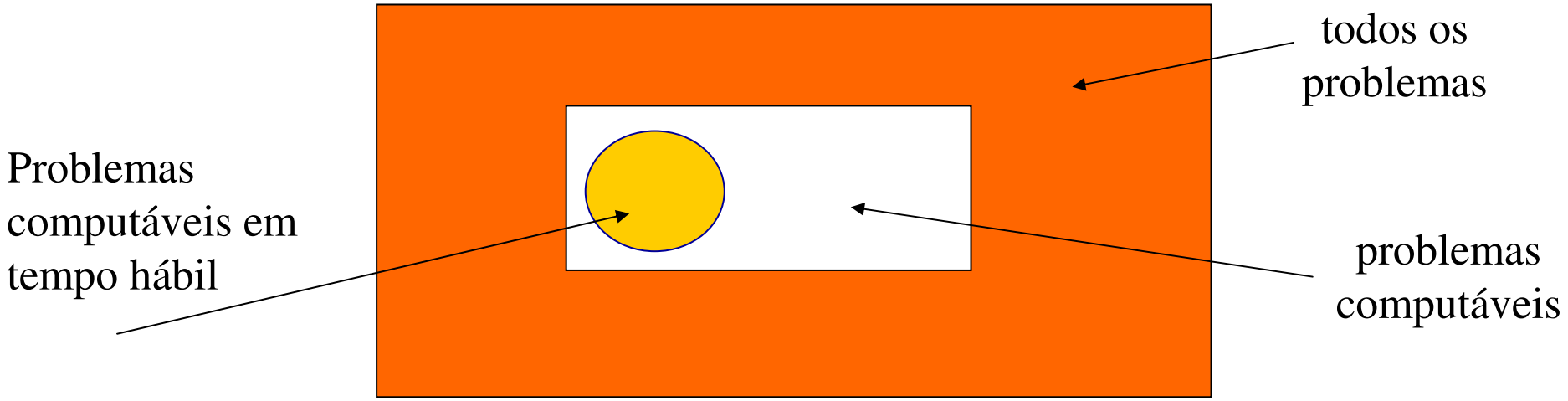
- Decidir se a resolução de um problema é livre de laços infinitos para qualquer conjunto de dados de entrada.

- Verificar a equivalência de dois algoritmos para qualquer conjunto de dados de entrada.

Complexidade

Para efeito prático não basta assegurar a existência de um algoritmo que solucione o problema de interesse.

É necessário também que a execução do mesmo se dê **em tempo razoável**.



Complexidade (continuação)

A **complexidade** de um algoritmo, incluídos seus dados de entrada, está relacionada a seu desempenho com respeito à utilização de recursos computacionais

. geralmente tempo de máquina (execução do algoritmo)

. mas também pode ser espaço de memória

em nosso caso nos interessa **tempo de execução**.

Complexidade: exemplos

- Multiplicação de 2 números de n dígitos

$m_1 \; m_2 \; m_3 \; m_4 \ldots m_n$

$p_1 \; p_2 \; p_3 \; p_4 \ldots p_n$

x

Teremos cada elemento de p multiplicando n elementos

e assim, n multiplicações. Isto para n elementos, assim

$n \cdot n \rightarrow n^2$ multiplicações.

E então a soma das parcelas: $n \cdot (n - 1)$ somas

Assim a execução da multiplicação é assintoticamente proporcional a n^2.

Dizemos então que a complexidade deste algoritmo de multiplicação é **polinomial quadrática**.

Complexidade: exemplos

- Outro algoritmo de multiplicação de 2 números de n dígitos

$$
\begin{array}{ll}
\begin{array}{cc} A & B \\ C & D \end{array} & \times \\
\hline
A \,.\, C & = \; X\,X \\
(A+B)\,.\,(C+D) - A.C - B.D & = \; X\;\;X\,X \\
B \,.\, D & = \qquad X\,X \\
\hline
 & \quad X\;\;X\,X\,X
\end{array}
$$

$$
\begin{array}{r}
23 \\
\underline{62} \\
1\,2 \\
0\,2\,2 \\
\underline{0\,6} \\
1\,4\,2\,6
\end{array}
$$

Prova-se que este algoritmo tem complexidade $n^{1,59}$

Complexidade: novo exemplo

- Consideremos que um problema admite soluções através de 4 algoritmos diferentes que têm diferentes complexidades para entradas de dados análogas:
  - Logarítmica
  - Linear
  - Polinomial quadrática
  - Exponencial

Comparação entre ordens de complexidade

(tempo para 1 operação é 1 ms)

tamanho do número (n-dígitos)	$\log_2 n$ (segundos)	n (segundos)	n^2 (segundos)	2^n (segundos)
10	0,003	0,01	0,1	1
100	0,006	0,1	10	10^{17} séculos
1.000	0,009	1	17 minutos	...
10.000	0,013	10	28 horas	...
100.000	0,016	100	116 dias	...

(exemplo – continuação)

- Notemos que:
  - as 4 soluções apresentadas são algoritmos e assim definem soluções computáveis;
  - algoritmos com complexidade de ordem superior à polinomial só são **exeqüíveis** para n (tamanho do problema) pequeno.

É igualmente importante ter presente que estamos tratando da chamada complexidade assintótica ($n \rightarrow \infty$). Vamos introduzir a notação O, para exprimir esta complexidade.

No exemplo teríamos: $O(\log_2 n)$, $O(n)$, $O(n^2)$, $O(2^n)$

Obs.: algoritmos exponenciais - $O(c^n)$ – são computacionalmente práticos somente para n pequeno.

Notação O

- Seja T(n) o número de operações em função do tamanho da entrada n

Assim se a multiplicação de 2 números tem complexidade (em função do número de operações) definida por $T(n) = n^2 + 8n$ operações

então $T(n) = O(n^2)$

Ou seja:

$T(n) = O(f(n))$

se existirem inteiro **m** e constante **d**, tais que $T(n) \leq d \cdot f(n)$ para $n > m$.

Para valores de n suficientemente grandes,

T(n) não cresce mais rapidamente do que f(n)

O( ) é o pior caso (limite superior)

Notação O(cont.)

Consideremos à título de ilustração:

(ver transparência anterior)

$T(n) = n^2 + 8.n$

$f(n) = n^2$

$d = 3$

$T(n) \leq d \,.\, f(n)$ para $n > m$

e então: $O(f(n)) = O(n^2)$

Então:

n	$n^2 + 8.n$	$3\ n^2$
1	9	3
2	20	12
3	33	27
4	48	48
5	65	75

$m = 5$

1.3 Problemas P, NP e NPC

Definição 1

Problemas Polinomiais – P : aqueles cuja melhor solução demanda algoritmos com complexidade (assintótica) de ordem até polinomial.

$O(n^c)$

Problemas Tratáveis

Problemas Não deterministicamente Polinomiais - NP: aqueles cuja melhor solução <u>conhecida</u> demanda algoritmos com complexidade (assintótica) de ordem exponencial.

$O(c^n)$

Problemas não Tratáveis

<u>Se</u> já provado que melhor solução é $O(c^n)$

Relembrando

Problemas NP são aqueles $O(c^n)$, porém não se provou que não existem algoritmos $O(n^c)$ para estes problemas.

Se já foi provado então >> **não tratáveis**.

Pergunta de grande interesse teórico:

P = NP ?

Não há resposta !

Outra Classe de Problemas

Dentre os problemas NP existe uma classe de problemas denominada **Problemas NP-Completos** introduzida em 1971 pelo pesquisador Cook.

classe de especial interesse teórico

Prova-se que se um problema está em NP e todos os outros problemas NP são reduzíveis em tempo polinomial para ele, então este problema é NP-completo (NPC).

continuação

Desta forma obter um algoritmo eficiente para qualquer problema NPC implicará que um algoritmo eficiente terá sido obtido para toda esta classe de problemas (NPC). E consequentemente para NP.

Com isto terá sido provado que P=NPC=NP

COM ISTO:

Os problemas NPC são alvo de estudo contínuo no tratamento da questão se NP = P ou NP $\neq$ P.

outra forma de apresentar a mesma questão

Definição 2:

P é a classe de problemas de decisão resolvidos por algoritmos que executam em tempo polinomial.

ou seja:

P = {q, onde q é um problema de decisão e existe um algoritmo A_q e constantes d , k tal que para todas as entradas i de tamanho n, A_q(i) termina em tempo inferior a $d.n^c$ e gera uma resposta correta (sim, não, falso, ...) }

Exemplo: decidir se um valor pertence à uma sequência.

Definição 3: (Verificação de Problemas)

q é um problema verificável por um algoritmo V, se V utilizando duas entradas i (qualquer entrada de q) e y (entrada qualquer) e para todo i há um y tal que V(i,y)=yes <u>se e só se</u> q(i) for yes.

(obs.: as duas entradas seriam seriam, por exemplo, no caso do C.Viajante – tr. 86 – o percurso e um valor desejado para o percurso.)

Definição 4: (note que esta definição não contraria aquela da Transp. 77)

NP = {q, onde q é um problema de decisão e existe um algoritmo de verificação V para q em tempo polinomial}

Exemplo:

Observemos que P **C** NP, pois qualquer problema em P pode ser verificado por V em tempo polinomial.

Observação sobre: redução de problemas

Redução é uma transformação de um problema em outro. Se um problema A pode ser reduzido a B, uma solução para B oferece uma solução para A. Assim resolver A não pode ser mais difícil (harder) do que resolver B.

Exemplo: redução de multiplicação para quadrado.

$a \cdot b = ((a + b)^2 - a^2 - b^2) / 2)$

A reduzido para B

Definição 5: Classe NPC

Esta classe contém os problemas da classe NP que são ainda mais difíceis de resolver.

Um problema p é **NPC** se:

a) p está em NP

b) todos os outros problemas q em NP podem ser "reduzidos" em tempo polinomial para p: há uma função f computável em tempo polinomial tal que para todas as entradas i de p, a resposta correta de q(i) é yes <u>se e somente se</u> a resposta correta de p(f(i)) for yes.

Ou seja, o tempo para resolver o problema q é o tempo para resolver o problema p adicionado ao tempo de f(i). O problema p é "mais difícil" ou "harder".

Obs.: a definição acima exige que se prove a redução de todos os problemas para o problema candidato (a NPC) p. Isto seria extremamente custoso. O lema a seguir trata isto.

Lema:

Para provar que um problema p, pertencente à classe NP, encontra-se em NPC (lembrar que NPC **C** NP) é suficiente provar que um problema q já conhecido e pertencente a NPC pode ser reduzido em tempo polinomial para p.

Ou seja:

q_{NPC} — reduzido para $\longrightarrow$ p_{NP} então p é NPC.

Pergunta: como se fez para identificar o 1° problema NPC ?

Cook, 1971, apontou o primeiro problema depois denominado NPC, definindo e provando o teorema:

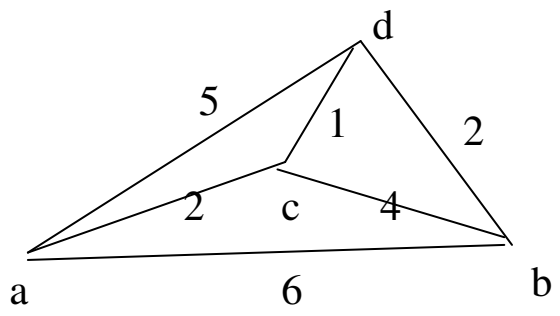
O problema da Satisfabilidade é NPC.

Problema da satisfabilidade: dada uma expressão booleana na Forma Normal Conjuntiva, verificar se ela é satisfatível.

Exemplo de problema NPC

Problema do caixeiro viajante

Dado um mapa com n cidades e a distância entre cada par de cidades, é possível obter para um vendedor que ele complete uma viagem, dentro de uma dada quilometragem, visitando cada cidade uma única vez ?



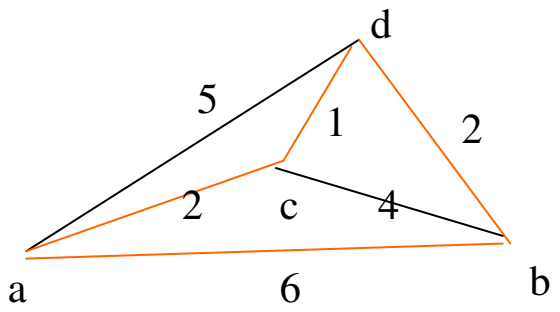
Na verdade podemos pensar em 3 problemas:

- Decisão: verificar se há viagem $\leq$ K km
- Localização: localizar viagem $\leq$ K km
- Otimização: obter viagem ótima ($K_{mínimo}$)

Continuação:

Solução força-bruta para viagem ótima: (nó inicial: d)

d	d	d	d	d	d
a	a	c	c	b	b
c	b	a	b	c	a
b	c	b	a	a	c
d	d	d	d	d	d
13	16	11	16	13	11



A solução é obtida analisando-se as (n-1)! possibilidades - todas as permutações: (4-1)!=6.

Por outro lado para verificar se um caminho satisfaz uma km temos custo n (=4), pois um caminho tem 4 nós.

→

nós	**n**	**(n-1)!**
4	4	6
10	10	362.880

1.4 Conclusões sobre algoritmos

- Computadores processam algoritmos
- Algoritmos
  - Computabilidade
  - Complexidade

1.5 Exercícios sobre algoritmos

1. Dê um exemplo de um procedimento, adicionalmente ao apresentado no texto (número perfeito).

2. Resolva os exercícios 2 a 6 do livro texto.

3. Dê 3 exemplos adicionais de problemas NPC. Para cada um deles analise a afirmação da transparência:

   "Adicionalmente problemas NPC têm a propriedade de que é possível mostrar em tempo polinomial se uma candidata à solução é ou não solução do problema."