1. Algoritmo e conceitos básicos

(referência básica: capítulo 1 do livro texto)

Referências adicionais:

.**David Harel**, Computers LTD – what they really can't do. Oxford University Press, Ed. 2004, ISBN 978-0-19-860442-6

. **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

53

. 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

. **Moshe Y.Vardi**, On P, NP, and Computational Complexity. Comm. Of the ACM, Nov. 2010, vol.53, pág. 5.

. **Paulo Ferfiloff**, IME-USP, www.ime.usp.br/~pf/analise_de_algoritmos/aulas/NPcompleto2.html

Algoritmo

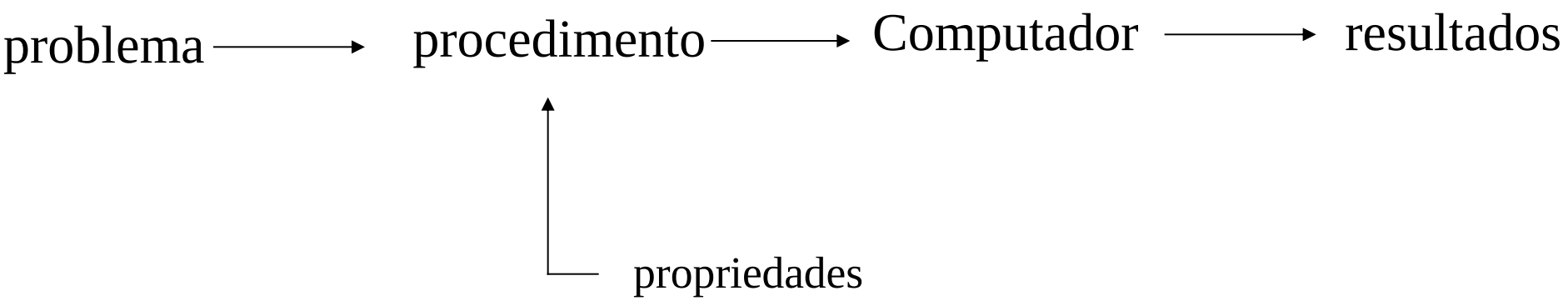
Realização de cálculos com grandezas → USO DO COMPUTADOR →

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

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

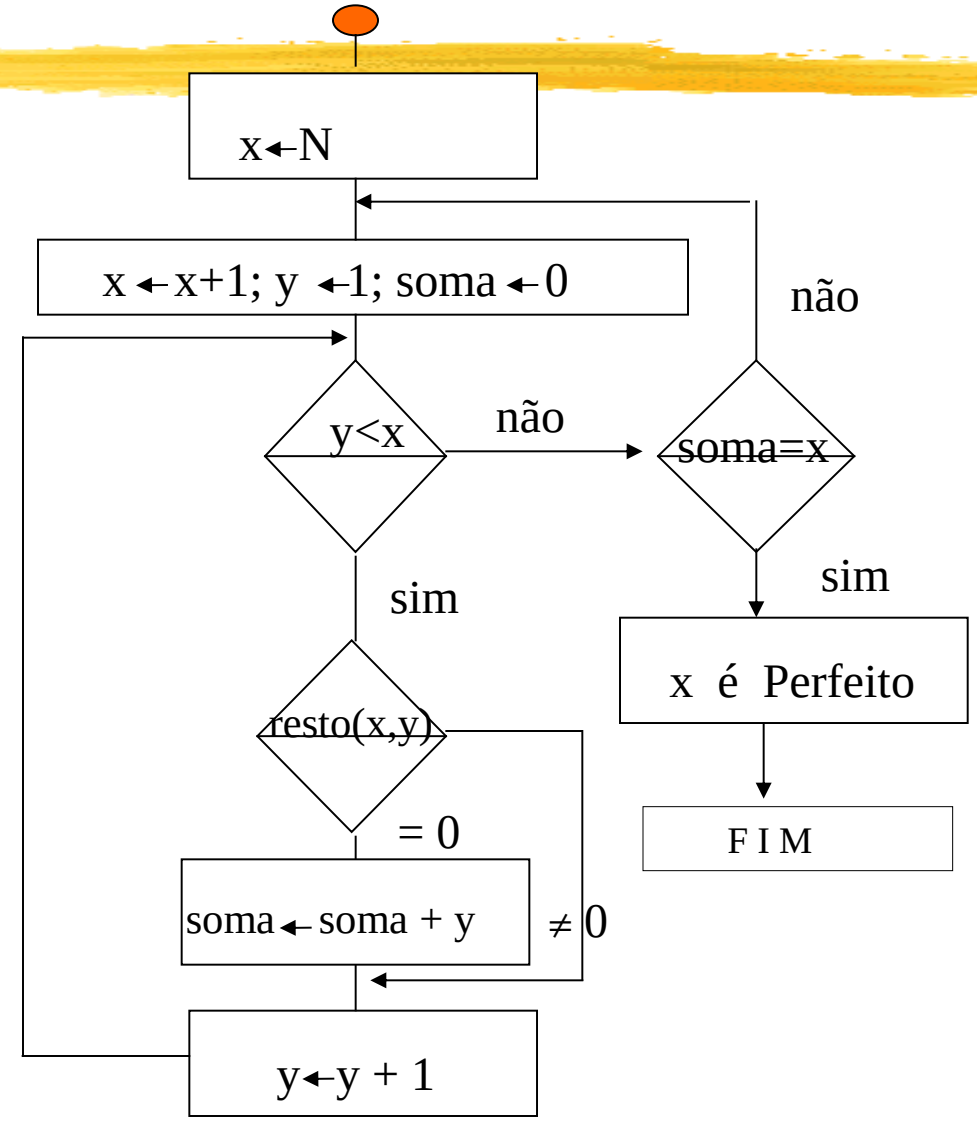
1.1 Procedimentos

• 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.**

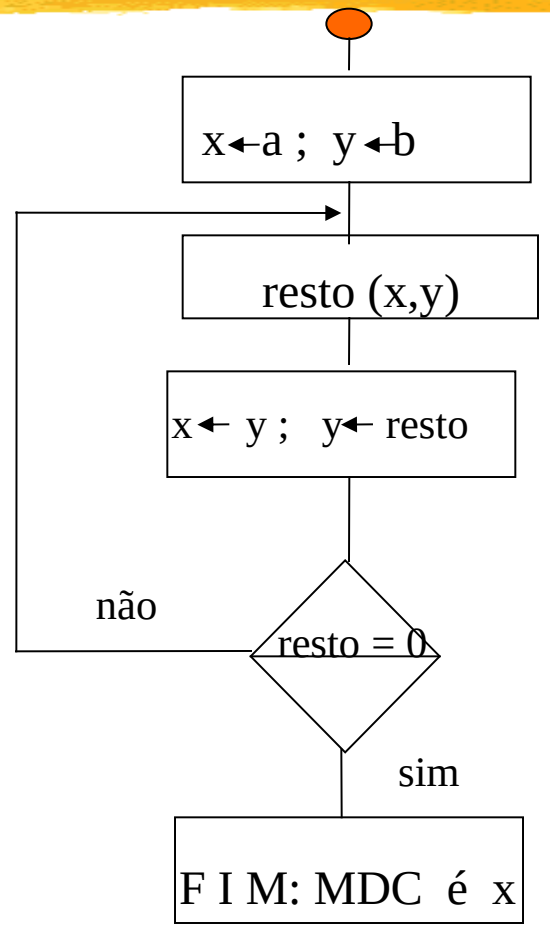
Algoritmo de Euclides para o cálculo de MDC de dois números

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).

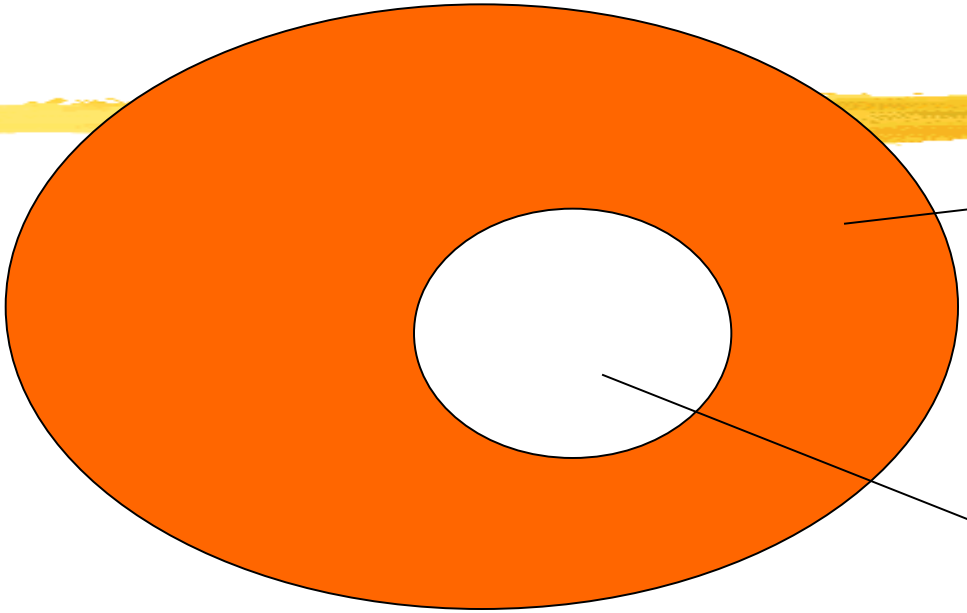
- O mesmo já não ocorre para o procedimento do Número Perfeito; a existência de uma quantidade finita / infinita de números perfeitos ainda é um problema em aberto.

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

Computabilidade
ou
Decidibilidade

Computabilidade e Complexidade

Passos na resolução de um problema

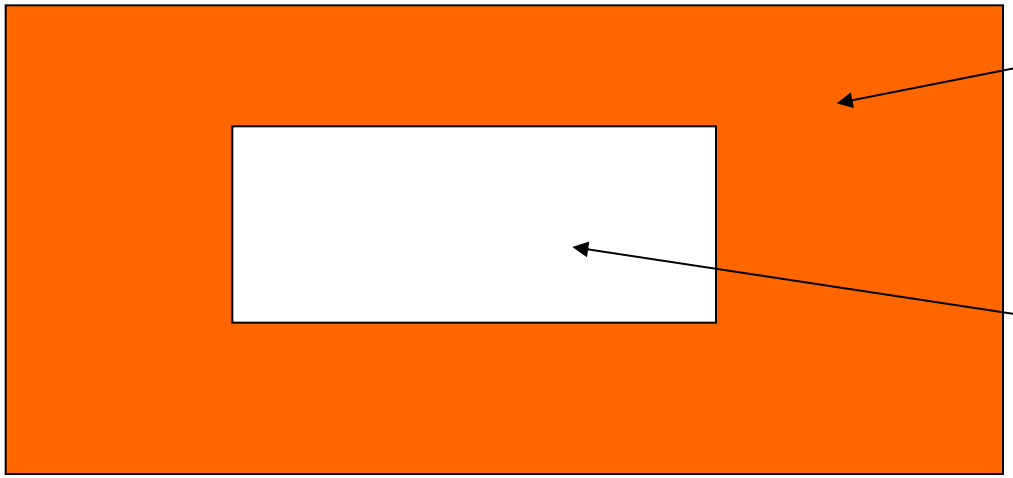
- **Definição do algoritmo**
- **Verificação da factibilidade da solução**
- **Programação do algoritmo**
- **Execução do programa**

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 **computa-bilidade** de qualquer problema bem formulado.

Entscheidungsproblem ou decidibilidade

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)

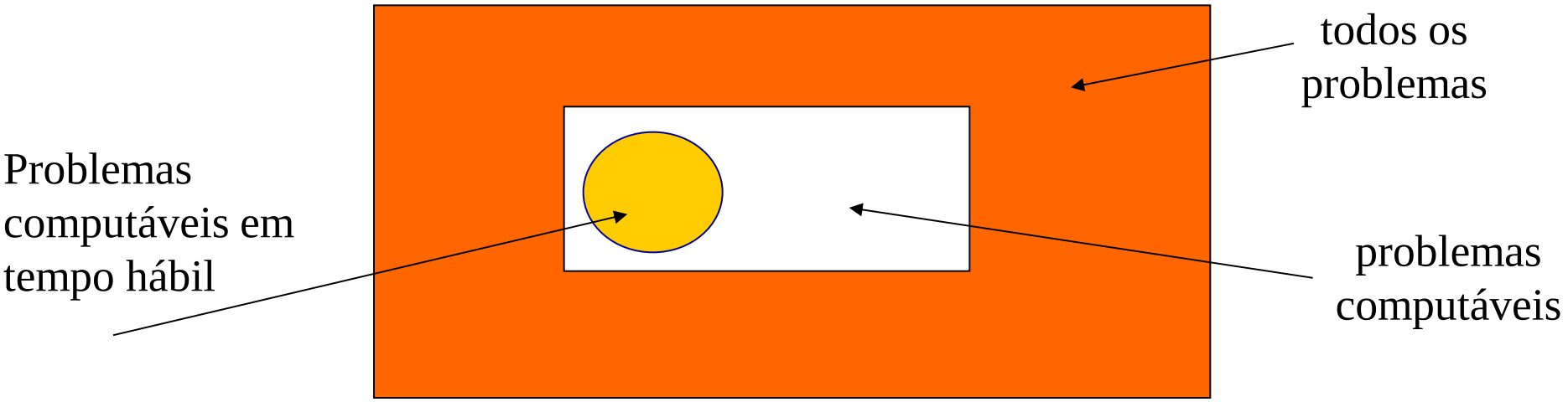
- **Decidir se a resolução de um problema é livre de laços infinitos para qualquer conjunto de dados de entrada.**

- **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**

$$\begin{array}{l} m_1\ m_2\ m_3\ m_4\ \ldots\ m_n \\ p_1\ p_2\ p_3\ p_4\ \ldots\ p_n \\ \\ \qquad\qquad\qquad\qquad\quad X \\ \hline \end{array}$$

Teremos cada elemento de p multiplicando n elementos

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

n . n → n^2 multiplicações.

E então a soma das parcelas: n . (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**

A B
C D
X

A . C = X X

(A+B) . (C+D) – A.C – B.D = X X X

B . D = X X

X X X X

23	
62	
1 2	
0 2 2	
0 6	
1 4 2 6	

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**

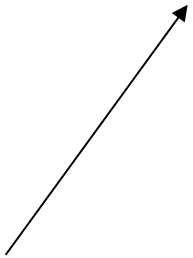
Assim se a multiplicação de 2 números tem complexidade (em função do número de operações) definida por T(n) = n2 + 8n operações

então T(n) = O(n2)

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)

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

$f(n) = n^2$

$d = 3$

Então:

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

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

e então: $O(f(n)) = O(n^2)$

$m = 5$

1.3 Problemas P, NP e NPC

Moshe Y.Vardi, On P, NP, and Computational Complexity. Comm. Of the ACM, Nov. 2010, vol.53, pág. 5.
Paulo Ferfiloff, IME-USP, www.ime.usp.br/~pf/analise_de_algoritmos/aulas/NPcompleto2.html

Definição 1

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

$O(n^c)$

Problemas Tratáveis

Onde **n** indica o tamanho do problema e **c** é uma constante qualquer.

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

$O(c^n)$

Problemas não Tratáveis

Se já provado que melhor solução é $O(c^n)$

Observação

Os Problemas $O(c^n)$ para os quais ainda não se descartou a existência de algoritmos $O(n^c)$ para tratá-los, serão considerados **tratáveis**.

Se já foi provado que não existem algoritmos $O(n^c)$

então >> **são Problemas não tratáveis**.

Retomando a questão **P** versus **NP**:

Pergunta de grande interesse teórico:

P = NP ?

Não há resposta !

Exemplos :

Problema P

- algoritmos de:
  - busca linear - O(n) - e de
  - busca binária - $O(\log_2 n)$

  em sequências ordenadas de números.

Problema NP (---> NPC)

- 3-cores: colorir com 3 cores diferentes os vértices de um grafo, de forma que 2 vértices compartilhando a mesma aresta não tem a mesma cor.

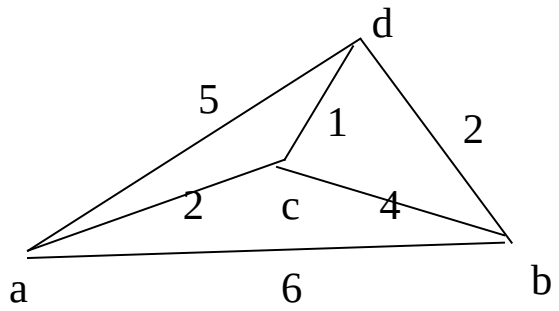
Outra Classe de Problemas

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

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 um percurso para um vendedor que ele complete dentro de uma dada quilometragem, visitando cada cidade uma única vez e retornando ao ponto de partida?



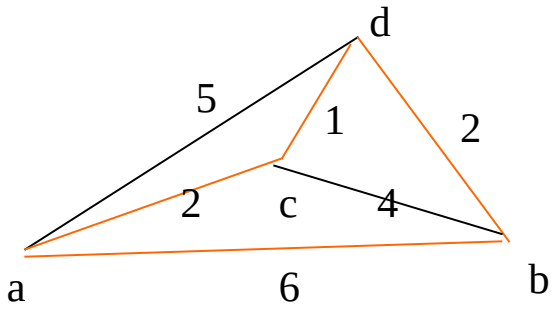
Na verdade podemos pensar em 3 problemas:

- Decisão: verificar se há viagem ≤ K km
- Localização: localizar viagem ≤ 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

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

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

Com isto terá sido provado que P=NPC e consequentemente P=NP

Assim:

Os problemas NPC são alvo de estudo contínuo no tratamento da questão se NP = P ou NP ≠ 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 em função do tamanho da entrada.

ou seja:

P = {q, onde q é um problema de decisão e existe um algoritmo A e constantes d , c tal que para todas as entradas i de tamanho n, o algoritmo A termina em tempo inferior a d.nc e gera uma resposta correta entre 2 possíveis (sim, não; verdadeiro, falso; ...) }

Exemplo: decidir se um valor pertence a uma sequência.

Definição 3: (Verificação de Problemas)

. Seja q um problema de decisão com entrada "i" e V um algoritmo que utiliza duas entradas: i (entrada qualquer de q) e y (entrada qualquer);

. q é verificado pelo algoritmo V se a seguinte condição for satisfeita: V(i,y)=yes se e só se q(i)=yes.

(obs.: as duas entradas seriam, por exemplo no caso do caixeiro viajante, o percurso "i" e a distância "y" desejada para o percurso.)

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

NP = {q, onde q é um problema de decisão para o qual existe um algoritmo de verificação V em tempo polinomial em função do tamanho da entrada de q}

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

Classe NPC

Esta classe contém os problemas da classe NP que potencialmente demandam um maior tempo para resolver.

Definição 5:

Um problema p é NPC se:

a) p está em NP

b) todos os outros problemas q em NP podem ser reduzidos (transformados) em tempo polinomial para resolver p. Ou seja, há uma função f computável em tempo polinomial tal que a resposta correta para q(i) é yes se e somente se a resposta correta de p(f(i)) for yes.

Ou seja, o tempo para resolver o problema q é potencialmente menor do que a soma dos tempos para resolver o problema p e calcular f(i). O problema p é "mais custoso" ou "harder".

Obs.: a definição da transparência anterior exige que se faça a redução de todos os problemas em NP para o problema p candidato a NPC. Isto seria extremamente custoso. Um lema trata isto na próxima transparência.

Definição 6: redução de problemas de decisão

Redução é uma transformação de um problema em outro.

Um problema de decisão Y é redutível a um problema de decisão X se existe um algoritmo que transforma qualquer instância J de Y em uma instância I de X, tal que J é verdadeira se e só se I for verdadeira. Informalmente, Y é redutível a X se Y for um subproblema ou caso particular de X.

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 custoso (harder) do que resolver B (ou seja, não pode exigir tempo maior).

Exemplo: o problema Quadrado-perfeito (Y) é polinomialmente redutível ao problema Equação de grau 2 inteira (X).

Quadrado perfeito: dada uma entrada N, número natural, decidir se existe natural X tal que X2 = N.

Equação de grau 2 inteira: dada uma entrada composta pelos inteiros a,b,c decidir se existe um inteiro x tal que ax2 + bx + c = 0.

Lema:

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

Ou seja:

p_{NPC} reduzido para $\longrightarrow$ r_{NP} então r é NPC.

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

Cook, 1971, apontou o primeiro problema NPC definindo e provando que o Problema da Satisfabilidade é NPC.

Problema da satisfabilidade: é o problema de decidir se uma fórmula expressa em lógica (booleana) proposicional é satisfazível, ou seja, se pode ser verdadeira. Este problema pode ser resolvido através da construção da Tabela Verdade e verificação se uma linha é verdadeira (lembre-se que há 2n linhas).

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."

4. Mostre que o problema Quadrado-perfeito é polinomialmente redutível ao problema Equação de grau 2 inteira".