Complexidade computacional

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

CLRS 34

Palavras

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

Palavras

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

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

Palavras

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

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

Qualquer sequência de elementos de um alfabeto
é chamada de uma palavra.

Palavras

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

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

Qualquer sequência de elementos de um alfabeto é chamada de uma palavra.

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

Palavras

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

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

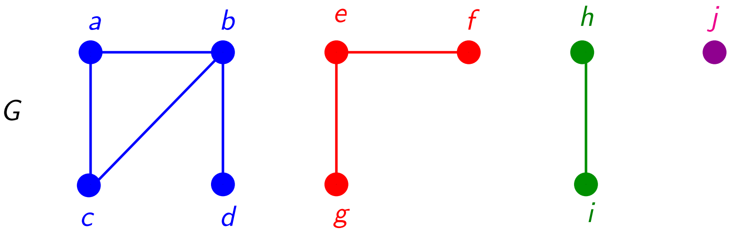
Qualquer sequência de elementos de um alfabeto
é chamada de uma palavra.

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



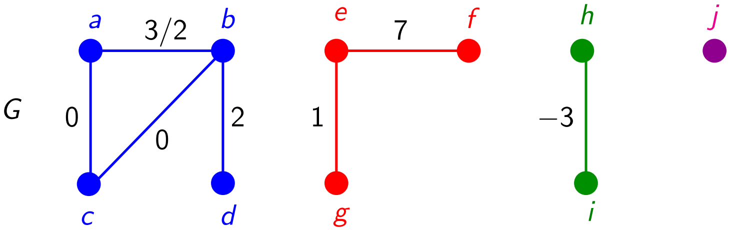
Palavra:

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

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 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 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.

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.

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.

Subsequência comum máxima

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

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco máxima = B C A B

Subsequência comum máxima

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

Exemplos: X = A B C B D A B

Y = 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 $\Theta(mn)$.

Subsequência comum máxima (decisão)

Problema: $X[1\,..\,m]$ e $Y[1\,..\,n]$ possuem uma ssco $\geq k$?

Problema de decisão: resposta sim ou não

Subsequência comum máxima (decisão)

Problema: $X[1\,..\,m]$ e $Y[1\,..\,n]$ possuem uma ssco $\geq k$?

Problema de decisão: resposta sim ou não

Instância: $X[1\,..\,m]$, $Y[1\,..\,n]$, 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 boolena 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 booleano da mochila

Problema (Knapsack Problem): Dados n, $w[1\,..\,n]$, $v[1\,..\,n]$ e W, encontrar uma mochila boolena 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, $w[1\,..\,n]$, $v[1\,..\,n]$, W e k, existe uma mochila boolena 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 booleano da mochila (decisão)

Problema (Knapsack Problem): Dados n, $w[1\,..\,n]$, $v[1\,..\,n]$, W e k, existe uma mochila boolena 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 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, $w[1\,..\,n]$ $v[\,..\,n]$, 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 fracionário da mochila (decisão)

Problema: Dados n, $w[1\,..\,n]$ $v[\,..\,n]$, 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, $w[1\,..\,n]$ $v[1\,..\,n]$, W e k

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

essencialmente $\lg n + n \lg W + n \lg V + \lg W + \lg k$

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

Modelo de computação

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

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 que um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome.

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
que 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.

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 que 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 determinado modelo de computação estima seu consumo de tempo e quantidade de espaço como função do tamanho da instância do problema.

Problemas polinomiais

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

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

Exemplos

- Subsequência comum máxima

  Tamanho da instância: $n + m$

  Consumo de tempo:

  REC-LCS-LENGTH é $\Omega(2^{\min\{m,n\}})$ (exponencial)

  LCS-LENGTH é $\Theta(mn)$ (polinomial).

Exemplos

- Subsequência comum máxima

  Tamanho da instância: $n + m$

  Consumo de tempo:

  REC-LCS-LENGTH é $\Omega(2^{\min\{m,n\}})$ (exponencial)

  LCS-LENGTH é $\Theta(mn)$ (polinomial).

- 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).

Mais exemplos

- 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).

Mais exemplos

- 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: $n \lg M$, onde
  $M := \max\{|A[1]|, |A[2]|, \ldots, |A[n]|\} + 1$

  Consumo de tempo:
  MERGESORT é $\Theta(n \lg n)$ (polinomial).

Classe P

Por algoritmo eficiente entende-se um algoritmo polinomial.

Classe P

Por algoritmo eficiente entende-se 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).

Classe P

Por algoritmo eficiente entende-se 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:

subsequência comum máxima e mochila fracionária

estão em P.

Classe P

Por algoritmo eficiente entende-se 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:

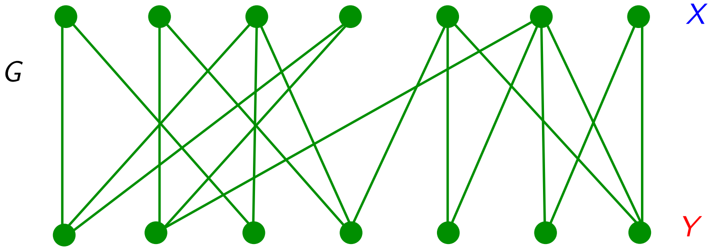
subsequência comum máxima e mochila fracionária

estão em P.

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

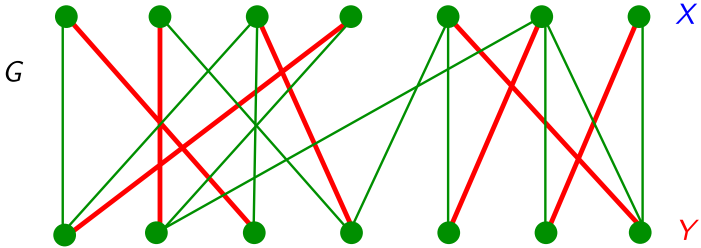
Emparelhamentos

Problema: Dado um grafo bipartido, encontrar um emparelhamento perfeito.



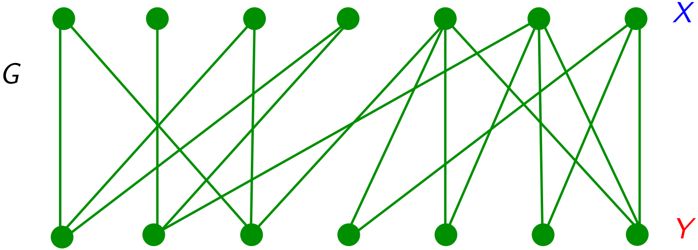
Emparelhamentos

Problema: Dado um grafo bipartido, encontrar um emparelhamento perfeito.



Emparelhamentos

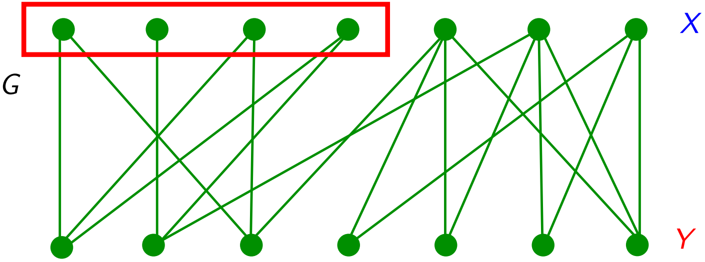
Problema: Dado um grafo bipartido, encontrar um emparelhamento perfeito.



não existe! Certificado?

Emparelhamentos

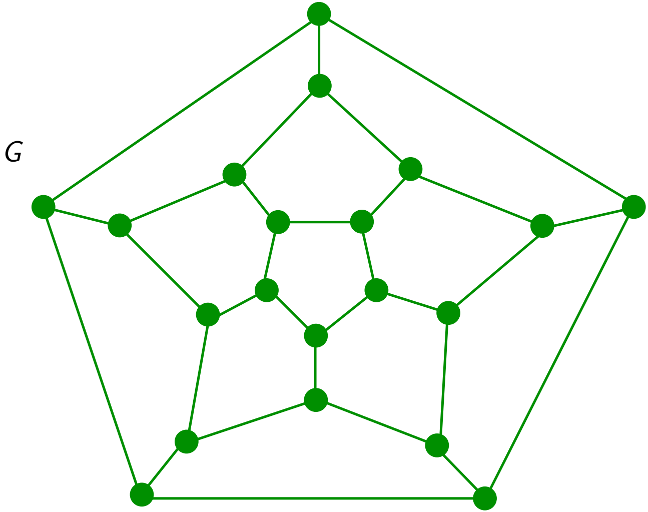
Problema: Dado um grafo bipartido, encontrar um emparelhamento bipartido.



não existe! Certificado: $S \subseteq X$ tal que $|S| > |\text{vizinhos}(S)|$.

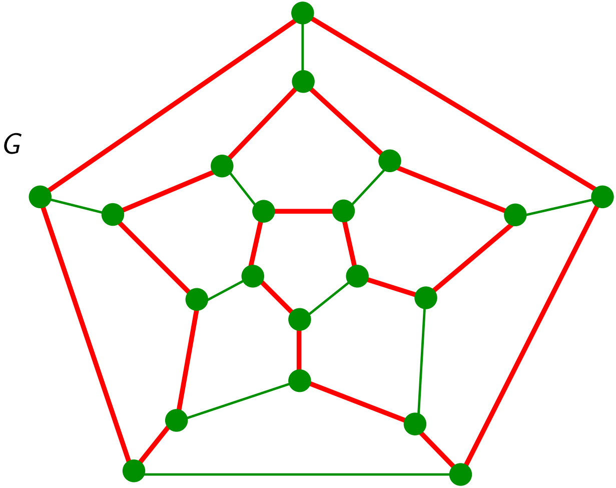
Grafos hamiltonianos

Problema: Dado um grafo, encontrar um ciclo hamiltoniano.



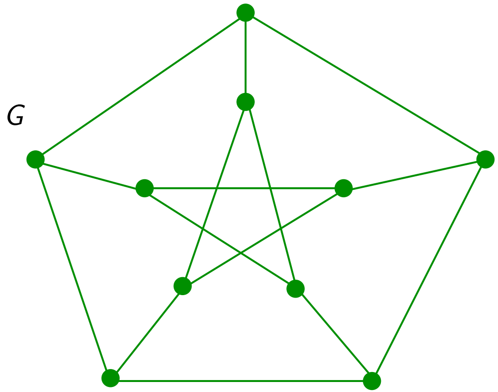
Grafos hamiltonianos

Problema: Dado um grafo, encontrar um ciclo hamiltoniano.



Grafos hamiltonianos

Problema: Dado um grafo, encontrar um ciclo hamiltoniano.



não existe! Certificado? Hmmm ...

Verificador polinomial para sim

Um verificador polinomial para a resposta sim a um problema Π é um algoritmo polinomial ALG que recebe

> *uma instância* I *de* Π *e um objeto* C,
> *tal que* $\langle C \rangle$ *é* $\mathrm{O}(\langle I \rangle^c)$ *para alguma constante* c

Verificador polinomial para sim

Um verificador polinomial para a resposta sim a um problema Π é um algoritmo polinomial ALG que recebe

uma instância I *de* Π *e um objeto* C,
tal que $\langle C\rangle$ *é* $\mathrm{O}(\langle I\rangle^c)$ *para alguma constante* c

e devolve

sim *para algum* C *se a resposta a* $\Pi(I)$ *é* sim*;*
não *para todo* C *se a resposta a* $\Pi(I)$ *é* não*.*

Verificador polinomial para sim

Um verificador polinomial para a resposta sim a um problema Π é um algoritmo polinomial ALG que recebe

> *uma instância I de Π e um objeto C,*
> *tal que $\langle C\rangle$ é $\mathrm{O}(\langle I\rangle^c)$ para alguma constante c*

e devolve

> sim *para algum C se a resposta a $\Pi(I)$ é* sim*;*
> não *para todo C se a resposta a $\Pi(I)$ é* não*.*

No caso de resposta sim, o objeto C é dito um certificado polinomial ou certificado curto da resposta sim a $\Pi(I)$.

Exemplos

- se G é hamiltoniano,
  então um ciclo hamiltoniano de G é um certificado polinomial:

  dados um grafo G *e* C, *pode-se verificar*
  em tempo $\mathrm{O}(\langle G\rangle)$ *se* C *é um ciclo hamiltoniano.*

Exemplos

- se G é hamiltoniano,
  então um ciclo hamiltoniano de G é um certificado polinomial:

  dados um grafo G e C, pode-se verificar
  em tempo $\mathrm{O}(\langle G \rangle)$ se C é um ciclo hamiltoniano.

- se $X[1\,..\,m]$ e $Y[1\,..\,n]$ possuem uma ssco $\geq k$, então uma subsequência comum $Z[1\,..\,k]$ é um certificado polinomial:

  dados $X[1\,..\,m]$, $Y[1\,..\,n]$ e $Z[1\,..\,k]$, pode-se verificar
  em tempo $\mathrm{O}(m+n)$ se Z é ssco de X e Y.

Exemplos

- se G é hamiltoniano,
  então um ciclo hamiltoniano de G é um certificado polinomial:

  dados um grafo G e C, pode-se verificar
  em tempo $\mathrm{O}(\langle G \rangle)$ se C é um ciclo hamiltoniano.

- se $X[1\,.\,.\,m]$ e $Y[1\,.\,.\,n]$ possuem uma ssco $\geq k$, então uma subsequência comum $Z[1\,.\,.\,k]$ é um certificado polinomial:

  dados $X[1\,.\,.\,m]$, $Y[1\,.\,.\,n]$ e $Z[1\,.\,.\,k]$, pode-se verificar
  em tempo $\mathrm{O}(m+n)$ se Z é ssco de X e Y.

- se n é um número composto, então
  um divisor próprio $d > 1$ de n é um certificado polinomial.

Verificado polinomial para não

Um verificador polinomial para a resposta não de um problema Π é um algoritmo polinomial ALG que recebe

> *uma instância I de Π e um objeto C,*
> *tal que $\langle C\rangle$ é $\mathrm{O}(\langle I\rangle^c)$ para alguma constante c*

e devolve

> sim *para algum C se a resposta a $\Pi(I)$ é* não*;*
> não *para todo C se a resposta a $\Pi(I)$ é* sim*.*

No caso de resposta sim, o objeto C é dito um certificado polinomial ou certificado curto da resposta não a $\Pi(I)$.

Classe NP

Formada pelos problemas de decisão que possuem um verificador polinomial para a resposta sim.

Classe NP

Formada pelos problemas de decisão que possuem um verificador polinomial para a resposta sim.

Em outras palavras, a classe NP é formada pelos problemas de decisão Π para os quais existe um problema Π' em P e uma função polinomial $p(n)$ tais que, para cada instância I do problema Π, existe um objeto C com $\langle C \rangle \leq p(\langle I \rangle)$ tal que

a resposta a $\Pi(I)$ *é* sim *se e somente se*
a resposta a $\Pi'(I, C)$ *é* sim.

Classe NP

Formada pelos problemas de decisão que possuem
um verificador polinomial para a resposta sim.

Em outras palavras, a classe NP é formada pelos problemas de decisão Π para os quais existe um problema Π' em P e uma função polinomial $p(n)$ tais que, para cada instância I do problema Π, existe um objeto C com $\langle C \rangle \leq p(\langle I \rangle)$ tal que

a resposta a $\Pi(I)$ é sim *se e somente se*
a resposta a $\Pi'(I, C)$ é sim*.*

O objeto C é dito um certificado polinomial
ou certificado curto da resposta sim a $\Pi(I)$.

Exemplos

Problemas de decisão com certificado polinomial para sim:

- existe subsequência crescente $\geq k$?
- existe subcoleção disjunta $\geq k$ de intervalos?
- existe mochila booleana de valor $\geq k$?
- existe mochila de valor $\geq k$?
- existe subsequência comum $\geq k$?
- grafo tem ciclo de comprimento $\geq k$?
- grafo tem ciclo hamiltoniano?
- grafo tem emparelhamento (casamento) perfeito?

Todos esses problemas estão em NP.

P $\subseteq$ NP

Prova:

se Π é um problema em P, então pode-se tomar a sequência de instruções realizadas por um algoritmo polinomial para resolver $\Pi(I)$ como certificado polinomial da resposta sim a $\Pi(I)$.

P $\subseteq$ NP

Prova:

se Π é um problema em P, então pode-se tomar a sequência de instruções realizadas por um algoritmo polinomial para resolver $\Pi(I)$ como certificado polinomial da resposta sim a $\Pi(I)$.

Outra prova:

Pode-se construir um verificador polinomial para a resposta sim a Π utilizando-se um algoritmo polinomial para Π como subrotina e ignorando-se o certificado C.

P $\neq$ NP?

É crença de muitos que a classe NP é maior que a classe P, ainda que isso

não tenha sido provado até agora.

$P \neq NP$?

É crença de muitos que a classe NP é maior que a classe P, ainda que isso

não tenha sido provado até agora.

Este é o intrigante problema matemático conhecido pelo rótulo "$P \neq NP$?"

$P \neq NP$?

É crença de muitos que a classe NP é maior que a classe P, ainda que isso

não tenha sido provado até agora.

Este é o intrigante problema matemático conhecido pelo rótulo “$P \neq NP$?”

Não confunda NP com “não-polinomial”.

Classe co-NP

A classe co-NP é definida
trocando-se sim por não na definição de NP.

Classe co-NP

A classe co-NP é definida
trocando-se sim por não na definição de NP.

Um problema de decisão Π está em co-NP se admite
um certificado polinomial para a resposta não.

Classe co-NP

A classe co-NP é definida
trocando-se sim por não na definição de NP.

Um problema de decisão Π está em co-NP se admite
um certificado polinomial para a resposta não.

Os problemas em NP $\cap$ co-NP admitem
certificados polinomiais para as respostas sim e não.

Classe co-NP

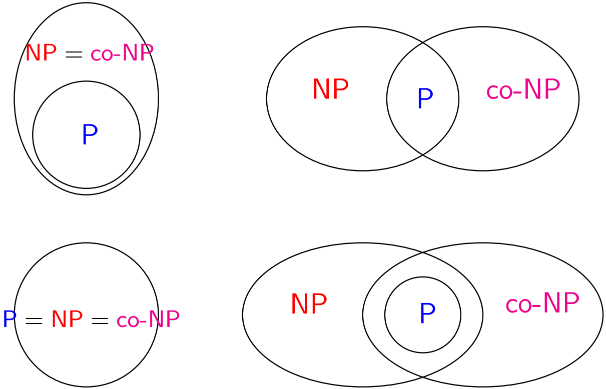
A classe co-NP é definida
trocando-se sim por não na definição de NP.

Um problema de decisão Π está em co-NP se admite
um certificado polinomial para a resposta não.

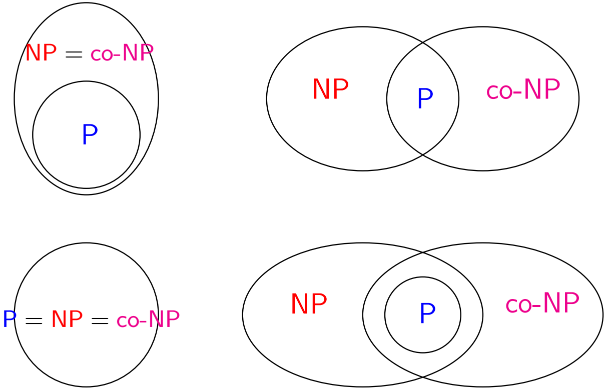
Os problemas em $\mathrm{NP} \cap \text{co-NP}$ admitem
certificados polinomiais para as respostas sim e não.

Em particular, $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$.

P, NP e co-NP



P, NP e co-NP



$P \neq NP$?

$NP \cap$ co-$NP \neq P$?

$NP \neq$ co-NP?

Redução polinomial

Permite comparar
o “grau de complexidade” de problemas diferentes.

Redução polinomial

Permite comparar
o “grau de complexidade” de problemas diferentes.

Π, Π': problemas

Uma redução de Π a Π' é um algoritmo ALG que resolve Π usando uma subrotina hipotética ALG$'$ que resolve Π', de forma que, se ALG$'$ é um algoritmo polinomial, então ALG é um algoritmo polinomial.

Redução polinomial

Permite comparar
o "grau de complexidade" de problemas diferentes.

Π, Π': problemas

Uma redução de Π a Π' é um algoritmo ALG que resolve Π usando uma subrotina hipotética ALG$'$ que resolve Π', de forma que, se ALG$'$ é um algoritmo polinomial, então ALG é um algoritmo polinomial.

$\Pi \leq_P \Pi'$ $=$ existe uma redução de Π a Π'.

Se $\Pi \leq_P \Pi'$ e Π' está em P, então Π está em P.

Exemplo

Π = encontrar um ciclo hamiltoniano
Π' = existe um ciclo hamiltoniano?

Exemplo

Π = encontrar um ciclo hamiltoniano
Π' = existe um ciclo hamiltoniano?

Redução de Π a Π': ALG$'$ é um algoritmo que resolve Π'

ALG (G)
se ALG$'(G)$ = não
  então devolva "G não é hamiltoniano"
para cada aresta uv de G **faça**
  $H \leftarrow G - uv$
  se ALG$'(H)$ = sim
    então $G \leftarrow G - uv$
devolva G

Exemplo

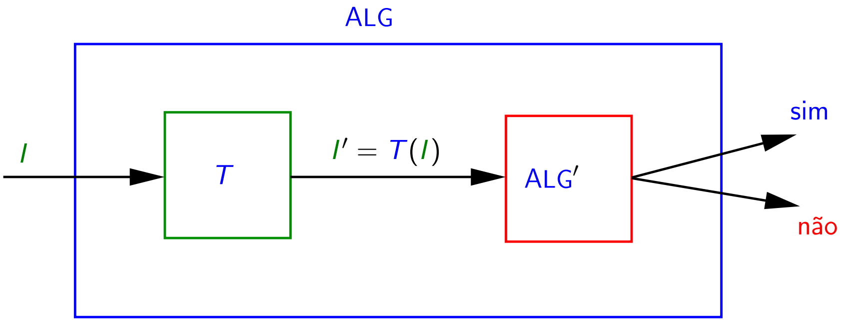
Π = encontrar um ciclo hamiltoniano
Π' = existe um ciclo hamiltoniano?

Redução de Π a Π': ALG$'$ é um algoritmo que resolve Π'

```
ALG (G)
se ALG'(G) = não
    então devolva "G não é hamiltoniano"
para cada aresta uv de G faça
    H ← G − uv
    se ALG'(H) = sim
        então G ← G − uv
devolva G
```

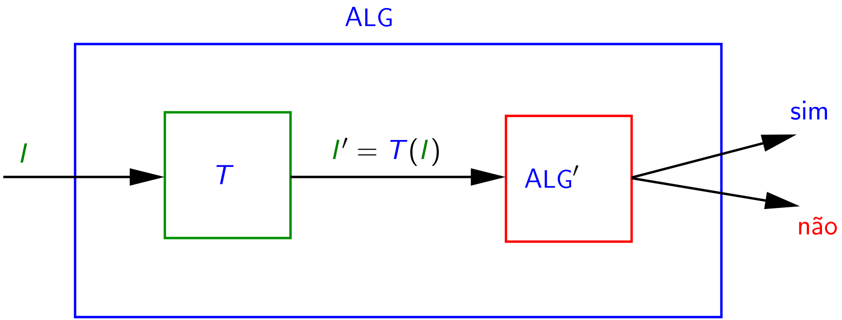
Se ALG$'$ consome tempo $\mathrm{O}(p(n))$, então ALG consome tempo $\mathrm{O}(m\, p(\langle G \rangle))$, onde m = número de arestas de G.

Esquema comum de redução



Faz apenas uma chamada ao algoritmo ALG′.

Esquema comum de redução

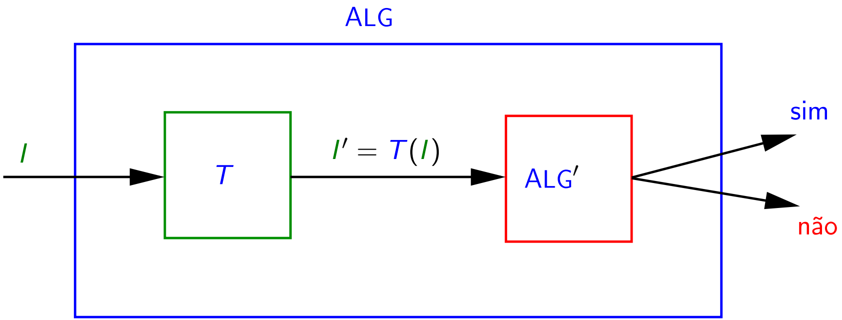


Faz apenas uma chamada ao algoritmo ALG$'$.

T transforma uma instância I de Π em
uma instância $I' = T(I)$ de Π' tal que

$$\Pi(I) = \text{sim se e somente se } \Pi'(I') = \text{sim}$$

Esquema comum de redução



Faz apenas uma chamada ao algoritmo ALG′.

T transforma uma instância I de Π em
uma instância $I' = T(I)$ de Π' tal que

$$\Pi(I) = \text{sim se e somente se } \Pi'(I') = \text{sim}$$

T é uma espécie de “filtro” ou “compilador”.

Satisfatibilidade

Problema: Dada uma fórmula booleana ϕ nas variáveis $x_1, \ldots, x_n$, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \rightarrow \{\text{verdade}, \text{falso}\}$$

que torna ϕ verdadeira?

Satisfatibilidade

Problema: Dada uma fórmula booleana ϕ
nas variáveis $x_1, \ldots, x_n$, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \rightarrow \{\text{verdade}, \text{falso}\}$$

que torna ϕ verdadeira?

Exemplo:

$$\phi = (x_1) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_3)$$

Satisfatibilidade

Problema: Dada uma fórmula booleana ϕ nas variáveis $x_1, \ldots, x_n$, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \to \{\text{verdade}, \text{falso}\}$$

que torna ϕ verdadeira?

Exemplo:

$$\phi = (x_1) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_3)$$

Se $t(x_1) = \text{verdade}, t(x_2) = \text{falso}, t(x_3) = \text{falso}$,
então $t(\phi) = \text{verdade}$

Satisfatibilidade

Problema: Dada uma fórmula booleana ϕ
nas variáveis $x_1, \ldots, x_n$, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \to \{\text{verdade}, \text{falso}\}$$

que torna ϕ verdadeira?

Exemplo:

$$\phi = (x_1) \land (\neg x_1 \lor \neg x_2 \lor x_3) \land (\neg x_3)$$

Se $t(x_1) = \text{verdade}, t(x_2) = \text{falso}, t(x_3) = \text{falso}$,
então $t(\phi) = \text{verdade}$

Se $t(x_1) = \text{verdade}, t(x_2) = \text{verdade}, t(x_3) = \text{falso}$,
então $t(\phi) = \text{falso}$

Sistemas lineares 0-1

Problema: Dadas uma matriz A e um vetor b,

$$Ax \geq b$$

possui uma solução tal que $x_i = 0$ ou $x_i = 1$ para todo i?

Sistemas lineares 0-1

Problema: Dadas uma matriz A e um vetor b,

$$Ax \geq b$$

possui uma solução tal que $x_i = 0$ ou $x_i = 1$ para todo i?

Exemplo:

$$\begin{array}{rrrrrrrr} & x_1 & & & & & \geq & 1 \\ - & x_1 & - & x_2 & + & x_3 & \geq & -1 \\ & & & & - & x_3 & \geq & 0 \end{array}$$

tem uma solução 0-1?

Sistemas lineares 0-1

Problema: Dadas uma matriz A e um vetor b,

$$Ax \geq b$$

possui uma solução tal que $x_i = 0$ ou $x_i = 1$ para todo i?

Exemplo:

$$\begin{array}{rrrrrrrl} & x_1 & & & & & \geq & 1 \\ - & x_1 & - & x_2 & + & x_3 & \geq & -1 \\ & & & & - & x_3 & \geq & 0 \end{array}$$

tem uma solução 0-1?

Sim! $x_1 = 1, x_2 = 0$ e $x_3 = 0$ é solução.

Exemplo 1

Satisfatibilidade $\leq_P$ Sistemas lineares 0-1