MAC 414
Autômatos, Computabilidade e Complexidade
aula 16 — 16/11/2020

Computabilidade X Complexidade

Computabilidade X Complexidade

Computabilidade: Estudo de funções computáveis *em princípio*.

Basta que existam algoritmos.

Computabilidade X Complexidade

Computabilidade: Estudo de funções computáveis *em princípio*.

Basta que existam algoritmos.

Vimos exemplos concretos de funções que não são computáveis.

Computabilidade X Complexidade

Computabilidade: Estudo de funções computáveis *em princípio*.

Basta que existam algoritmos.

Vimos exemplos concretos de funções que não são computáveis.

Complexidade: Termo usado e abusado em Matemática, Computação, Física, etc.

Computabilidade X Complexidade

Computabilidade: Estudo de funções computáveis *em princípio*.

Basta que existam algoritmos.

Vimos exemplos concretos de funções que não são computáveis.

Complexidade: Termo usado e abusado em Matemática, Computação, Física, etc.

Aqui é sinônimo de *Complexidade de Computação*.

Computabilidade X Complexidade

Computabilidade: Estudo de funções computáveis *em princípio*.
Basta que existam algoritmos.
Vimos exemplos concretos de funções que não são computáveis.

Complexidade: Termo usado e abusado em Matemática, Computação, Física, etc.
Aqui é sinônimo de *Complexidade de Computação*.
Classificar algoritmos de acordo com gasto de tempo e espaço, e problemas computáveis de acordo com a complexidade dos algoritmos que os resolvem.

Relembrando O

$f, g : \mathbb{N} \to \mathbb{N}$ crescentes

$f(n) = O(g(n))$ se $\exists A, n_0$ tq $f(n) \leq A g(n)$ p/ $n > n_0$

$f(n) = \Theta(g(n))$ se $f(n) = O(g(n))$ e $g(n) = O(f(n))$

Se $p(n)$ é polinômio de grau k, então,

$p(n) = \Theta(n^k) = \Theta(n^{k+1}) = \Theta(2^n)$

Complexidade de tempo

Complexidade de tempo

Considere um modelo de computação para o qual existe uma noção de "passo", que será considerado como unidade de tempo.

Complexidade de tempo

Considere um modelo de computação para o qual existe uma noção de "passo", que será considerado como unidade de tempo.

Dado um algoritmo nesse modelo, sua complexidade de tempo é

$f(n) =$ máximo número de passos para terminar com entrada de tamanho n.

Complexidade de tempo

Considere um modelo de computação para o qual existe uma noção de "passo", que será considerado como unidade de tempo.

Dado um algoritmo nesse modelo, sua complexidade de tempo é

$f(n) =$ máximo número de passos para terminar com entrada de tamanho n.

Complexidade de tempo

Considere um modelo de computação para o qual existe uma noção de "passo", que será considerado como unidade de tempo.

Dado um algoritmo nesse modelo, sua complexidade de tempo é

$f(n) =$ máximo número de passos para terminar com entrada de tamanho n.

Convenção: n é o tamanho da entrada

Complexidade de MT

Complexidade de MT

Seja $\mathscr{M}$ uma MT que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máximo número de passos para terminar com entrada de tamanho n. Nesse caso, dizemos que $\mathscr{M}$ roda em tempo $f(n)$ e também que $\mathscr{M}$ é uma MT de tempo $f(n)$.

Complexidade de MT

Seja $\mathscr{M}$ uma MT que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máximo número de passos para terminar com entrada de tamanho n. Nesse caso, dizemos que $\mathscr{M}$ roda em tempo $f(n)$ e também que $\mathscr{M}$ é uma MT de tempo $f(n)$.

Em geral, em vez de apresentar $f(n)$, apresentamos uma estimativa $\mathcal{O}(g(n))$.

Complexidade de MT

Seja $\mathscr{M}$ uma MT que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máximo número de passos para terminar com entrada de tamanho n. Nesse caso, dizemos que $\mathscr{M}$ roda em tempo $f(n)$ e também que $\mathscr{M}$ é uma MT de tempo $f(n)$.

Em geral, em vez de apresentar $f(n)$, apresentamos uma estimativa $\mathcal{O}(g(n))$.

$\mathscr{M}$ roda em tempo polinomial se para algum inteiro k ela roda em tempo $\mathcal{O}(n^k)$.

Complexidade de MT

Seja $\mathscr{M}$ uma MT que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máximo número de passos para terminar com entrada de tamanho n. Nesse caso, dizemos que $\mathscr{M}$ roda em tempo $f(n)$ e também que $\mathscr{M}$ é uma MT de tempo $f(n)$.

Em geral, em vez de apresentar $f(n)$, apresentamos uma estimativa $\mathcal{O}(g(n))$.

$\mathscr{M}$ roda em tempo polinomial se para algum inteiro k ela roda em tempo $\mathcal{O}(n^k)$.

Mesma coisa para MT com k fitas.

Complexidade de MT

Seja $\mathscr{M}$ uma MT que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máximo número de passos para terminar com entrada de tamanho n. Nesse caso, dizemos que $\mathscr{M}$ roda em tempo $f(n)$ e também que $\mathscr{M}$ é uma MT de tempo $f(n)$.

Em geral, em vez de apresentar $f(n)$, apresentamos uma estimativa $\mathcal{O}(g(n))$.

$\mathscr{M}$ roda em tempo polinomial se para algum inteiro k ela roda em tempo $\mathcal{O}(n^k)$.

Mesma coisa para MT com k fitas.

E MT não determinística?

Complexidade de MT

Seja $\mathscr{M}$ uma MT que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máximo número de passos para terminar com entrada de tamanho n. Nesse caso, dizemos que $\mathscr{M}$ roda em tempo $f(n)$ e também que $\mathscr{M}$ é uma MT de tempo $f(n)$.

Em geral, em vez de apresentar $f(n)$, apresentamos uma estimativa $\mathcal{O}(g(n))$.

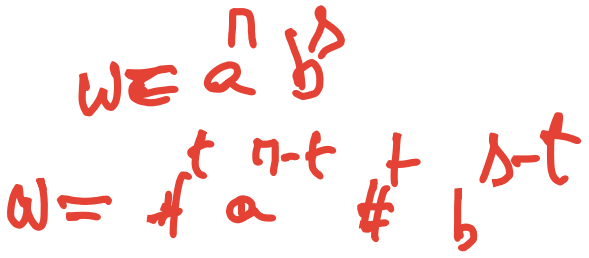
$\mathscr{M}$ roda em tempo polinomial se para algum inteiro k ela roda em tempo $\mathcal{O}(n^k)$.

Mesma coisa para MT com k fitas.

E MT não determinística? Depois...

Uma MT para reconhecer $AnBn$

1. Verifique se a entrada está em a^*b^*.
   Se não, **rejeite**
2. Procure um a, b ou $\sqcup$.
   - Encontrou a:
     - Marque com #.
     - Procure b ou $\sqcup$.
       Encontrou b: marque com #, cursor para início,
       volte para (2).
       Encontrou $\sqcup$: **rejeite**.
   - Encontrou b: **rejeite**.
   - Encontrou $\sqcup$: **aceite**.



Uma MT para reconhecer $AnBn$

1. Verifique se a entrada está em a^*b^*.
   Se não, **rejeite**
2. Procure um a, b ou $\sqcup$.
   - Encontrou a:
     - Marque com #.
     - Procure b ou $\sqcup$.
       Encontrou b: marque com #, cursor para início,
       volte para (2).
       Encontrou $\sqcup$: **rejeite**.
   - Encontrou b: **rejeite**.
   - Encontrou $\sqcup$: **aceite**.

Tempos:

1. $\mathcal{O}(n)$

Uma MT para reconhecer *AnBn*

1. Verifique se a entrada está em $a^* b^*$.
   Se não, **rejeite**
2. Procure um a, b ou $\sqcup$.
   - Encontrou a:
     - Marque com #.
     - Procure b ou $\sqcup$.
       Encontrou b: marque com #, cursor para início,
       volte para (2).
       Encontrou $\sqcup$: **rejeite**.
   - Encontrou b: **rejeite**.
   - Encontrou $\sqcup$: **aceite**.

Tempos:

1. $\mathcal{O}(n)$
2. Dada $a^r b^s$: $\min(r,s) \cdot (r+s) =$

Uma MT para reconhecer *AnBn*

1. Verifique se a entrada está em $a^* b^*$.
   Se não, **rejeite**
2. Procure um a, b ou $\sqcup$.
   - Encontrou a:
     - Marque com #.
     - Procure b ou $\sqcup$.
       Encontrou b: marque com #, cursor para início,
       volte para (2).
       Encontrou $\sqcup$: **rejeite**.
   - Encontrou b: **rejeite**.
   - Encontrou $\sqcup$: **aceite**.

Tempos:

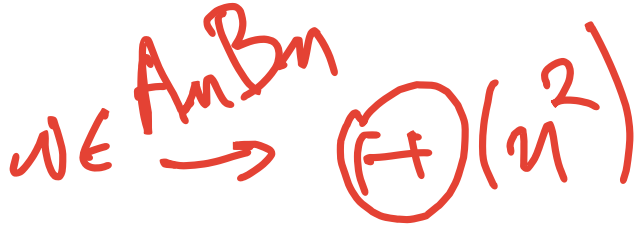
1. $\mathcal{O}(n)$
2. Dada $a^r b^s$: $\min(r,s) \cdot (r+s) =$

Uma MT para reconhecer *AnBn*

1. Verifique se a entrada está em $a^* b^*$.
   Se não, **rejeite**
2. Procure um a, b ou $\sqcup$.
   - Encontrou a:
     - Marque com #.
     - Procure b ou $\sqcup$.
       Encontrou b: marque com #, cursor para início,
       volte para (2).
       Encontrou $\sqcup$: **rejeite**.
   - Encontrou b: **rejeite**.
   - Encontrou $\sqcup$: **aceite**.

Tempos:

1. $\mathcal{O}(n)$
2. Dada $a^r b^s$: $\min(r,s) \cdot (r+s) = \mathcal{O}(n^2)$.



AnBn com duas fitas

1. Use a fita 2 para contar os a's (em unário)

$AnBn$ com duas fitas

1. Use a fita 2 para contar os a's (em unário)
2. Use a fita 2 para descontar o b's.

$AnBn$ com duas fitas

1. Use a fita 2 para contar os a's (em unário)
2. Use a fita 2 para descontar o b's.
3. **rejeite** se: aparecer algum a depois de ter visto um b, ou se as contagens não derem iguais.

$AnBn$ com duas fitas

1. Use a fita 2 para contar os a's (em unário)
2. Use a fita 2 para descontar o b's.
3. **rejeite** se: aparecer algum a depois de ter visto um b, ou se as contagens não derem iguais.
4. **aceite**, caso contrário.

$AnBn$ com duas fitas

1. Use a fita 2 para contar os a's (em unário)
2. Use a fita 2 para descontar o b's.
3. **rejeite** se: aparecer algum a depois de ter visto um b, ou se as contagens não derem iguais.
4. **aceite**, caso contrário.

AnBn com duas fitas

1. Use a fita 2 para contar os a's (em unário)
2. Use a fita 2 para descontar o b's.
3. **rejeite** se: aparecer algum a depois de ter visto um b, ou se as contagens não derem iguais.
4. **aceite**, caso contrário.

Tempo $\mathcal{O}(n)$.

AnBn, uma fita, mais esperto

1. Se a entrada não está em a^*b^*, **rejeite**.

AnBn, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:

AnBn, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:

   Se $w \in \#^*$, **aceite**.

$AnBn$, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:

   Se $w \in \#^*$, **aceite**.

   Se $|w|_a \not\equiv |w|_b \mod 2$, **rejeite**.

AnBn, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:

   Se $w \in \#^*$, **aceite**.

   Se $|w|_a \not\equiv |w|_b \mod 2$, **rejeite**.

   Repita: {apague a, pule a }.

$|w|_a \leftarrow |w|_a / 2$

AnBn, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:

   Se $w \in \#^*$, **aceite**.
   Se $|w|_a \not\equiv |w|_b \mod 2$, **rejeite**.
   Repita: {apague a, pule a }.
   Repita: {apague b, pule b }.

$|w|_a \leftarrow (|w|_a)/2$
$|w|_b \leftarrow |w|_b/2$

AnBn, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:

   Se $w \in \#^*$, **aceite**.
   Se $|w|_a \not\equiv |w|_b \mod 2$, **rejeite**.
   Repita: {apague a, pule a }.
   Repita: {apague b, pule b }.

AnBn, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:

   Se $w \in \#^*$, **aceite**.

   → Se $|w|_a \not\equiv |w|_b \mod 2$, **rejeite**.

   Repita: {apague a, pule a }.

   Repita: {apague b, pule b }.

Funcionamento: dada $a^r b^s$, verifica se r e s têm a mesma representação binária.

AnBn, uma fita, mais esperto

1. Se a entrada não está em $a^* b^*$, **rejeite**.
2. Loop [fita contendo $\triangleright \sqcup w \sqcup$, com $w \in (a + b + \#)^*$]:
   - Se $w \in \#^*$, **aceite**.
   - Se $|w|_a \not\equiv |w|_b \mod 2$, **rejeite**.
   - Repita: {apague a, pule a }.
   - Repita: {apague b, pule b }.

Funcionamento: dada $a^r b^s$, verifica se r e s têm a mesma representação binária.

Tempo: $\mathcal{O}(n \log n)$.

Classes

Classes

Seja $t : \mathbb{N} \to \mathbb{N}$ uma função crescente. A classe de complexidade de tempo $\textsc{time}(t(n))$ consiste das linguagens reconhecíveis por MT de complexidade $\mathcal{O}(t(n))$.

decidíveis

Classes

Seja $t : \mathbb{N} \to \mathbb{N}$ uma função crescente. A classe de complexidade de tempo $\textsc{time}(t(n))$ consiste das linguagens reconhecíveis por MT de complexidade $\mathcal{O}(t(n))$.

A classe **P** consiste das *linguagens* decidíveis em tempo polinomial. Ou seja,

$$\mathbf{P} = \bigcup_{k \geq 0} \textsc{time}(n^k).$$

Classes

Seja $t : \mathbb{N} \to \mathbb{N}$ uma função crescente. A classe de complexidade de tempo $\textsc{time}(t(n))$ consiste das linguagens reconhecíveis por MT de complexidade $\mathcal{O}(t(n))$.

A classe **P** consiste das *linguagens* decidíveis em tempo polinomial. Ou seja,

$$\mathbf{P} = \bigcup_{k \geq 0} \textsc{time}(n^k).$$

Equivalentemente: L está em **P** sse

existe $\mathscr{M}$ decidindo L e um polinômio $p(n)$ tal que

para toda entrada w de comprimento n,

$\mathscr{M}$ para em no máximo $p(n)$ passos.

Dois fatos

Teorema

P é fechada por complemento.

Dois fatos

Teorema

P é fechada por complemento.

Dois fatos

Teorema

P é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$

Dois fatos

Teorema

$\mathbf{P}$ é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$

Proposição

E é recursiva, mas não está em $\mathbf{P}$.

Dois fatos

Teorema

P é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$

Proposição

E é recursiva, mas não está em P.

Dois fatos

Teorema

P é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$? $\in P$

Proposição

E é recursiva, mas não está em P.

Dem: Diagonalize — considere:

$$\hat{E} = \{\text{“}M\text{”}\text{“}M\text{”} \mid M \text{ não decide } \text{“}M\text{”} \text{ em} \leq 2^{|\text{“}M\text{”}|} \text{ passos}\}$$

Dois fatos

Teorema

P é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$

Proposição

E é recursiva, mas não está em P.

Dem: Diagonalize — considere:

$$\hat{E} = \{\text{“}M\text{”}\text{“}M\text{”} \mid M \text{ não decide “}M\text{” em} \leq 2^{|\text{“}M\text{”}|} \text{ passos}\}$$

Se $\hat{E} \in \mathbf{P}$, existe MT $\hat{M}$ que decide $\hat{E}$ em $p(n)$ passos.

Dois fatos

Teorema

P é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$

Proposição

E é recursiva, mas não está em P.

Dem: Diagonalize — considere:

$$\hat{E} = \{\text{“}M\text{”}\text{“}M\text{”} \mid M \text{ não decide “}M\text{”} \text{ em} \leq 2^{|\text{“}M\text{”}|} \text{ passos}\}$$

Se $\hat{E} \in \mathbf{P}$, existe MT $\hat{M}$ que decide $\hat{E}$ em $p(n)$ passos.
S.P.G., escolha $\hat{M}$ tal que $p(|\text{“}\hat{M}\text{”}|) \leq 2^{|\text{“}\hat{M}\text{”}|}$.

Dois fatos

Teorema

P é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$

Proposição

E é recursiva, mas não está em P.

Dem: Diagonalize — considere:

$$\hat{E} = \{\text{“}M\text{”}\text{“}M\text{”} \mid M \text{ não decide “}M\text{”} \text{ em} \leq 2^{|\text{“}M\text{”}|} \text{ passos}\}$$

Se $\hat{E} \in \mathbf{P}$, existe MT $\hat{M}$ que decide $\hat{E}$ em $p(n)$ passos.
S.P.G., escolha $\hat{M}$ tal que $p(|\text{“}\hat{M}\text{”}|) \leq 2^{|\text{“}\hat{M}\text{”}|}$.
“$\hat{M}$” está em $\hat{E}$?

Dois fatos

Teorema

P é fechada por complemento.

Seja $E = \{\text{“}M\text{”}\text{“}x\text{”} \mid M \text{ decide } x \text{ em} \leq 2^{|x|} \text{ passos}\}$

Proposição

E é recursiva, mas não está em P.

Dem: Diagonalize — considere:

$$\hat{E} = \{\text{“}M\text{”}\text{“}M\text{”} \mid M \text{ não decide “}M\text{”} \text{ em} \leq 2^{|\text{“}M\text{”}|} \text{ passos}\}$$

Se $\hat{E} \in \mathbf{P}$, existe MT $\hat{M}$ que decide $\hat{E}$ em $p(n)$ passos.
S.P.G., escolha $\hat{M}$ tal que $p(|\text{“}\hat{M}\text{”}|) \leq 2^{|\text{“}\hat{M}\text{”}|}$.
“$\hat{M}$” está em $\hat{E}$? Análogo: $\textsc{time}(n^k) \subsetneq \textsc{time}(n^{k+1})$.

Robustez da classe P

Robustez da classe P

"A" classe $\textsc{time}(t(n))$ depende fortemente do modelo de computação.

Robustez da classe P

"A" classe $\textsc{time}(t(n))$ depende fortemente do modelo de computação.

A classe **P** é mais robusta: é invariante para vários modelos de computação determinísticos.

Robustez da classe P

"A" classe $\textsc{time}(t(n))$ depende fortemente do modelo de computação.

A classe **P** é mais robusta: é invariante para vários modelos de computação determinísticos.

Exemplo: seja $\textsc{time}_k(t(n))$ a classe de linguagens decididas por MTs com k fitas em tempo $t(n)$.

Robustez da classe P

"A" classe $\textsc{time}(t(n))$ depende fortemente do modelo de computação.

A classe **P** é mais robusta: é invariante para vários modelos de computação determinísticos.

Exemplo: seja $\textsc{time}_k(t(n))$ a classe de linguagens decididas por MTs com k fitas em tempo $t(n)$. Trivialmente, $\textsc{time}_1(t(n)) \subseteq \textsc{time}_k(t(n))$.

Robustez da classe P

"A" classe $\textsc{time}(t(n))$ depende fortemente do modelo de computação.

A classe **P** é mais robusta: é invariante para vários modelos de computação determinísticos.

Exemplo: seja $\textsc{time}_k(t(n))$ a classe de linguagens decididas por MTs com k fitas em tempo $t(n)$.
Trivialmente, $\textsc{time}_1(t(n)) \subseteq \textsc{time}_k(t(n))$.
A simulação mostra que $\textsc{time}_k(t(n)) \subseteq \textsc{time}_1(t(n)^2)$.

Robustez da classe P

"A" classe $\textsc{time}(t(n))$ depende fortemente do modelo de computação.

A classe **P** é mais robusta: é invariante para vários modelos de computação determinísticos.

Exemplo: seja $\textsc{time}_k(t(n))$ a classe de linguagens decididas por MTs com k fitas em tempo $t(n)$.
Trivialmente, $\textsc{time}_1(t(n)) \subseteq \textsc{time}_k(t(n))$.
A simulação mostra que $\textsc{time}_k(t(n)) \subseteq \textsc{time}_1(t(n)^2)$.
Ou seja, **P** não depende do número de fitas.

Reduções

Def: Sejam $L_1, L_2 \subseteq \Sigma^*$. Uma redução de L_1 a L_2 é uma função recursiva $\tau : \Sigma^* \to \Sigma^*$ tal que para todo $x \in \Sigma^*$, $\tau(x) \in L_2$ sse $x \in L_1$.

Reduções

Def: Sejam $L_1, L_2 \subseteq \Sigma^*$. Uma redução polinomial de L_1 a L_2 é uma função computável em tempo polinomial $\tau : \Sigma^* \to \Sigma^*$ tal que para todo $x \in \Sigma^*$, $\tau(x) \in L_2$ sse $x \in L_1$.

Reduções

Def: Sejam $L_1, L_2 \subseteq \Sigma^*$. Uma redução polinomial de L_1 a L_2 é uma função computável em tempo polinomial $\tau : \Sigma^* \to \Sigma^*$ tal que para todo $x \in \Sigma^*$, $\tau(x) \in L_2$ sse $x \in L_1$.

Notação: $L_1 \prec L_2$ se existe redução polinomial de L_1 a L_2.

Reduções

Def: Sejam $L_1, L_2 \subseteq \Sigma^*$. Uma redução polinomial de L_1 a L_2 é uma função computável em tempo polinomial $\tau : \Sigma^* \to \Sigma^*$ tal que para todo $x \in \Sigma^*$, $\tau(x) \in L_2$ sse $x \in L_1$.

Notação: $L_1 < L_2$ se existe redução polinomial de L_1 a L_2.

$<$ é reflexiva e transitiva. L_1 e L_2 são polinomialmente equivalentes se cada uma se reduz à outra.

Problemas em P

Problemas em P

Problema: definido por um conjunto $\mathscr{I}$ de instâncias e uma função resposta $\mathscr{R}$. $\mathscr{I}$ e $\mathscr{R}$ podem consistir de objetos matemáticos abstratos.

Problemas em P

Problema: definido por um conjunto $\mathscr{I}$ de instâncias e uma função resposta $\mathscr{R}$. $\mathscr{I}$ e $\mathscr{R}$ podem consistir de objetos matemáticos abstratos.

Problema de computação:

- são dadas *codificações* $\mathscr{I}, \mathrm{Im}(\mathscr{R}) \longrightarrow \Sigma^*$

Problemas em P

Problema: definido por um conjunto $\mathscr{I}$ de instâncias e uma função resposta $\mathscr{R}$. $\mathscr{I}$ e $\mathscr{R}$ podem consistir de objetos matemáticos abstratos.

Problema de computação:

- são dadas *codificações* $\mathscr{I}, \mathrm{Im}(\mathscr{R}) \longrightarrow \Sigma^*$
- uma solução é um algoritmo que computa $\mathscr{R}(A)$, dada uma instância A, nas respectivas codificações.

Problemas em P

Problema: definido por um conjunto $\mathscr{I}$ de instâncias e uma função resposta $\mathscr{R}$. $\mathscr{I}$ e $\mathscr{R}$ podem consistir de objetos matemáticos abstratos.

Problema de computação:

- são dadas *codificações* $\mathscr{I}, \mathrm{Im}(\mathscr{R}) \longrightarrow \Sigma^*$
- uma solução é um algoritmo que computa $\mathscr{R}(A)$, dada uma instância A, nas respectivas codificações.

Em particular, um problema é de decisão se a resposta é **sim** ou **não**.

Problemas em P

Problema: definido por um conjunto $\mathscr{I}$ de instâncias e uma função resposta $\mathscr{R}$. $\mathscr{I}$ e $\mathscr{R}$ podem consistir de objetos matemáticos abstratos.

Problema de computação:

- são dadas *codificações* $\mathscr{I}, \mathrm{Im}(\mathscr{R}) \longrightarrow \Sigma^*$
- uma solução é um algoritmo que computa $\mathscr{R}(A)$, dada uma instância A, nas respectivas codificações.

Em particular, um problema é de decisão se a resposta é **sim** ou **não**. Equivale a decidir as codificaçoes de instâncias com resposta **sim**.

Problemas em P

Problema: definido por um conjunto $\mathscr{I}$ de instâncias e uma função resposta $\mathscr{R}$. $\mathscr{I}$ e $\mathscr{R}$ podem consistir de objetos matemáticos abstratos.

Problema de computação:

- são dadas *codificações* $\mathscr{I}, \mathrm{Im}(\mathscr{R}) \longrightarrow \Sigma^*$
- uma solução é um algoritmo que computa $\mathscr{R}(A)$, dada uma instância A, nas respectivas codificações.

Em particular, um problema é de decisão se a resposta é **sim** ou **não**. Equivale a decidir as codificaçoes de instâncias com resposta **sim**.

Para muitos fins, codificações são especificadas a menos de equivalência polinomial.

Codificações

Codificações

Números naturais: usando uma base $b \geq 2$ fixa.

Codificações

Números naturais: usando uma base $b \geq 2$ fixa.

Algoritmos de mudança de base são polinomiais – todas as bases são polinomialmente equivalentes.

Codificações

Números naturais: usando uma base $b \geq 2$ fixa.

Algoritmos de mudança de base são polinomiais – todas as bases são polinomialmente equivalentes.

Isso não funciona com representação em unário.

Codificações

Números naturais: usando uma base $b \geq 2$ fixa.

Algoritmos de mudança de base são polinomiais – todas as bases são polinomialmente equivalentes.

Isso não funciona com representação em unário.

Grafos: matrizes de adjacência, de incidência, listas de arestas, etc.

Todas polinomialmente equivalentes.

Codificações

Números naturais: usando uma base $b \geq 2$ fixa.

Algoritmos de mudança de base são polinomiais – todas as bases são polinomialmente equivalentes.

Isso não funciona com representação em unário.

Grafos: matrizes de adjacência, de incidência, listas de arestas, etc.

Todas polinomialmente equivalentes.

Algoritmos concretos trabalham com uma codificação específica. Para ser usados com outra codificação, ou têm que ser adaptados, ou precedido de uma recodificação - a redução.

Mais sobre complexidade

As instâncias podem ter *parâmetros* naturais;
nesse caso, é costume avaliar as codificações e os tempos em termos dos parâmetros.

Mais sobre complexidade

As instâncias podem ter *parâmetros* naturais;
nesse caso, é costume avaliar as codificações e os tempos em termos dos parâmetros.

Grafos nº vértices, nº arestas

Reduções polinomiais preservam a classe **P**.
Já as classes $\textsc{time}(n^k)$ não.
Para preservar o grau, é preciso que a redução esteja no mesmo grau.

Alguns problemas famosos

Alguns problemas famosos

SAT: Dada uma expressão booleana em forma normal conjuntiva (produto de somas), determinar se existe uma atribuição às variáveis que leva a um valor **V**.

Alguns problemas famosos

SAT: Dada uma expressão booleana em forma normal conjuntiva (produto de somas), determinar se existe uma atribuição às variáveis que leva a um valor **V**.

3-SAT: Caso particular de SAT em que cada termo tem 3 somandos.

Alguns problemas famosos

SAT: Dada uma expressão booleana em forma normal conjuntiva (produto de somas), determinar se existe uma atribuição às variáveis que leva a um valor **V**.

3-SAT: Caso particular de SAT em que cada termo tem 3 somandos.

HAMILTONIANO: dado um grafo, ele é hamiltoniano (existe um circuito que passa por todos os vértices)?

Alguns problemas famosos

SAT: Dada uma expressão booleana em forma normal conjuntiva (produto de somas), determinar se existe uma atribuição às variáveis que leva a um valor **V**.

3-SAT: Caso particular de SAT em que cada termo tem 3 somandos.

HAMILTONIANO: dado um grafo, ele é hamiltoniano (existe um circuito que passa por todos os vértices)?

CAIXEIRO VIAJANTE: dados inteiros $n, k > 0$ e uma função peso $p : E(K_n) \to \mathbb{N}$, existe um passeio fechado que passa por todos os vértices do K_n com peso total $\leq k$?

Alguns problemas famosos

SAT: Dada uma expressão booleana em forma normal conjuntiva (produto de somas), determinar se existe uma atribuição às variáveis que leva a um valor **V**.

3-SAT: Caso particular de SAT em que cada termo tem 3 somandos.

HAMILTONIANO: dado um grafo, ele é hamiltoniano (existe um circuito que passa por todos os vértices)?

CAIXEIRO VIAJANTE: dados inteiros $n, k > 0$ e uma função peso $p : E(K_n) \to \mathbb{N}$, existe um passeio fechado que passa por todos os vértices do K_n com peso total $\leq k$? Passeio de peso mínimo.

Reduções

Reduções

3-SAT $\leq$ SAT: caso particular.

Reduções

3-SAT $\leq$ SAT: caso particular.

SAT $\leq$ 3-SAT: vamos ver depois.

Reduções

3-SAT $\prec$ SAT: caso particular.

SAT $\prec$ 3-SAT: vamos ver depois.

HAMILTONIANO $\prec$ CAIXEIRO VIAJANTE:

Reduções

3-SAT $\prec$ SAT: caso particular.

SAT $\prec$ 3-SAT: vamos ver depois.

HAMILTONIANO $\prec$ CAIXEIRO VIAJANTE:

Dado G com n vértices. No K_n, defina $p(e) = 1$ se $e \in E(G)$, n caso contrário. Defina $k = n$.

SAT é muito expressivo

SAT é muito expressivo

Usando:

$+$	para $\vee$, **or**, $\vert\vert$,
$\cdot$	para $\wedge$, **and**, &&,
$\bar{x}$	para $\neg x$, **not** x, !x.

SAT é muito expressivo

Usando:

$+$	para $\vee$, **or**, $\vert\vert$,
$\cdot$	para $\wedge$, **and**, &&,
$\bar{x}$	para $\neg x$, **not** x, !x.

$\mathbb{U}(x_1, x_2, \ldots, x_n) = 1$ sse exatamente um dos x_i vale 1.

SAT é muito expressivo

Usando:

$+$	para $\vee$, **or**, $\vert\vert$,
$\cdot$	para $\wedge$, **and**, &&,
$\bar{x}$	para $\neg x$, **not** x, !x.

$\mathbb{U}(x_1, x_2, \ldots, x_n) = 1$ sse exatamente um dos x_i vale 1.

$= (x_1 + x_2 + \cdots + x_n) \prod_{i<j} (\bar{x}_i + \bar{x}_j)$.

SAT é muito expressivo

Usando:

$+$	para $\vee$, **or**, $\vert\vert$,
$\cdot$	para $\wedge$, **and**, &&,
$\bar{x}$	para $\neg x$, **not** x, !x.

$\mathbb{U}(x_1, x_2, \ldots, x_n) = 1$ sse exatamente um dos x_i vale 1.

$= (x_1 + x_2 + \cdots + x_n) \prod_{i<j} (\bar{x}_i + \bar{x}_j)$.

$|\mathbb{U}(x_1, x_2, \ldots, x_n)| = \Theta(n^2)$.

SAT é muito expressivo

Usando:

$+$	para $\vee$, **or**, $\vert\vert$,
$\cdot$	para $\wedge$, **and**, &&,
$\bar{x}$	para $\neg x$, **not** x, !x.

$\mathbb{U}(x_1, x_2, \ldots, x_n) = 1$ sse exatamente um dos x_i vale 1.

$= (x_1 + x_2 + \cdots + x_n) \prod_{i<j} (\bar{x}_i + \bar{x}_j)$.

$|\mathbb{U}(x_1, x_2, \ldots, x_n)| = \Theta(n^2)$.

Se várias fórmulas exprimem regras, seu produto exprime a conjunção delas.

HAMILTONIANO$\prec$SAT

HAMILTONIANO$\prec$SAT

Variáveis: $\{x_{ij} \mid 1 \leq i, j \leq n\}$.

HAMILTONIANO$\prec$SAT

Variáveis: $\{x_{ij} \mid 1 \leq i, j \leq n\}$. Idéia: $x_{ij} = 1$ se vértice i é o j-ésimo de um circuito hamiltoniano; $x_{i(n+1)} \equiv x_{i1}$.

HAMILTONIANO$\prec$SAT

Variáveis: $\{x_{ij} \mid 1 \leq i, j \leq n\}$. Idéia: $x_{ij} = 1$ se vértice i é o j-ésimo de um circuito hamiltoniano; $x_{i(n+1)} \equiv x_{i1}$.

Multiplicando requisitos:

1. Para cada j tem um único i: $\bigwedge_i \mathbb{U}(x_{1j}, \ldots, x_{nj})$

HAMILTONIANO$\prec$SAT

Variáveis: $\{x_{ij} \mid 1 \leq i, j \leq n\}$. Idéia: $x_{ij} = 1$ se vértice i é o j-ésimo de um circuito hamiltoniano; $x_{i(n+1)} \equiv x_{i1}$.

Multiplicando requisitos:

1. Para cada j tem um único i: $\mathbb{U}(x_{1j}, \ldots, x_{nj})$
2. Para cada i tem um único j: $\mathbb{U}(x_{i1}, \ldots, x_{in})$

HAMILTONIANO$\prec$SAT

Variáveis: $\{x_{ij} \mid 1 \leq i, j \leq n\}$. Idéia: $x_{ij} = 1$ se vértice i é o j-ésimo de um circuito hamiltoniano; $x_{i(n+1)} \equiv x_{i1}$.
Multiplicando requisitos:

1. Para cada j tem um único i: $\mathbb{U}(x_{1j}, \ldots, x_{nj})$
2. Para cada i tem um único j: $\mathbb{U}(x_{i1}, \ldots, x_{in})$
3. Se x_{ij} e $x_{k(j+1)}$ são verdadeiros, então i e k são adjacentes.

HAMILTONIANO$\prec$SAT

Variáveis: $\{x_{ij} \mid 1 \leq i, j \leq n\}$. Idéia: $x_{ij} = 1$ se vértice i é o j-ésimo de um circuito hamiltoniano; $x_{i(n+1)} \equiv x_{i1}$.
Multiplicando requisitos:

1. Para cada j tem um único i: $\mathbb{U}(x_{1j}, \ldots, x_{nj})$
2. Para cada i tem um único j: $\mathbb{U}(x_{i1}, \ldots, x_{in})$
3. Se x_{ij} e $x_{k(j+1)}$ são verdadeiros, então i e k são adjacentes.

HAMILTONIANO$\prec$SAT

Variáveis: $\{x_{ij} \mid 1 \le i,j \le n\}$. Idéia: $x_{ij} = 1$ se vértice i é o j-ésimo de um circuito hamiltoniano; $x_{i(n+1)} \equiv x_{i1}$.
Multiplicando requisitos:

1. Para cada j tem um único i: $\mathbb{U}(x_{1j}, \ldots, x_{nj})$
2. Para cada i tem um único j: $\mathbb{U}(x_{i1}, \ldots, x_{in})$
3. Se x_{ij} e $x_{k(j+1)}$ são verdadeiros, então i e k são adjacentes. Equivalente: se i e k não são adjacentes, então x_{ij} e $x_{k(j+1)}$ não são ambos verdadeiros.
   Para todos i,k não adjacentes, e todo j, $\bar{x}_{ij} + \bar{x}_{k(j+1)}$. Ou seja:

$$\prod_{\substack{1 \le i,j,k \le n \\ i,k \text{ não adjacentes}}} \left(\bar{x}_{ij} + \bar{x}_{k(j+1)}\right)$$

O custo da redução

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$.

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$. Números usados nos índices usam $\lg n$ bits – devia multiplicar tudo por esse valor.

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$. Números usados nos índices usam $\lg n$ bits – devia multiplicar tudo por esse valor.

Redução: n^2 variáveis.

1. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$. Números usados nos índices usam $\lg n$ bits – devia multiplicar tudo por esse valor.

Redução: n^2 variáveis.

1. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
2. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$. Números usados nos índices usam $\lg n$ bits – devia multiplicar tudo por esse valor.

Redução: n^2 variáveis.

1. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
2. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
3. $n\mathcal{O}(\binom{n}{2} - m) = \mathcal{O}(n^3)$

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$. Números usados nos índices usam $\lg n$ bits – devia multiplicar tudo por esse valor.

Redução: n^2 variáveis.

1. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
2. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
3. $n\mathcal{O}(\binom{n}{2} - m) = \mathcal{O}(n^3)$

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$. Números usados nos índices usam $\lg n$ bits – devia multiplicar tudo por esse valor.

Redução: n^2 variáveis.

1. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
2. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
3. $n\mathcal{O}(\binom{n}{2} - m) = \mathcal{O}(n^3)$

Se o grafo era denso ($m = \Theta(n^2)$), a representação original tinha tamanho $R = \Theta(n^2)$, foi para $R^{\frac{3}{2}}$.

O custo da redução

Grafo: n vértices, m arestas. Representação mais comum: listas de arestas. Comprimento $\Theta(n+m)$. Números usados nos índices usam $\lg n$ bits – devia multiplicar tudo por esse valor.

Redução: n^2 variáveis.

1. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
2. $n\mathcal{O}(n^2) = \mathcal{O}(n^3)$.
3. $n\mathcal{O}(\binom{n}{2} - m) = \mathcal{O}(n^3)$

Se o grafo era denso ($m = \Theta(n^2)$), a representação original tinha tamanho $R = \Theta(n^2)$, foi para $R^{\frac{3}{2}}$.

Se o grafo era bem esparso ($m = \mathcal{O}(n)$), foi de $R = \mathcal{O}(n)$ para R^3.

Vamos ver

Todos eles se reduzem uns aos outros!