MAC 414
Autômatos, Computabilidade e Complexidade
aula 17 — 18/11/2020

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 r fitas.

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

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,

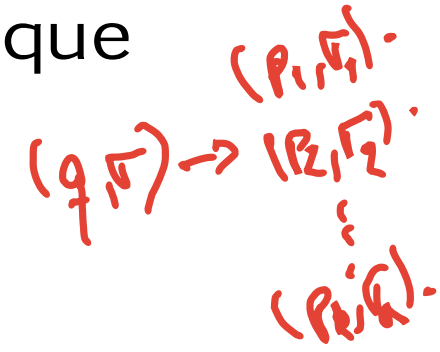
$\mathscr{M}$ para em no máximo $p(n)$ passos.

MT não determinística

MT não determinística

$M = (K, \Sigma, \Delta, s, H)$. A diferença com MT é que $\Delta \subseteq (K \backslash H \times \Sigma \cup \{\triangleright\}) \times (K \times \Sigma \cup \{\leftarrow, \rightarrow\})$.

Configuraçoes, $\vdash_M$, $\vdash_M^*$ como antes.



MT não determinística

$M = (K, \Sigma, \Delta, s, H)$. A diferença com MT é que
$\Delta \subseteq (K \backslash H \times \Sigma \cup \{\triangleright\}) \times (K \times \Sigma \cup \{\leftarrow, \rightarrow\})$.
Configuraçoes, $\vdash_M$, $\vdash_M^*$ como antes.

A partir de uma configuração, existe uma árvore de computações, que pode ter caminhos infinitos.
Nós: configurações.
Filho dada por $\vdash_M$ (transição pode rotular aresta).
Folhas: ou o estado está em H, ou simplesmente não tem o que fazer.

MT não determinística

$M = (K, \Sigma, \Delta, s, H)$. A diferença com MT é que $\Delta \subseteq (K \backslash H \times \Sigma \cup \{\triangleright\}) \times (K \times \Sigma \cup \{\leftarrow, \rightarrow\})$.
Configuraçoes, $\vdash_M$, $\vdash_M^*$ como antes.

A partir de uma configuração, existe uma árvore de computações, que pode ter caminhos infinitos.
Nós: configurações.
Filho dada por $\vdash_M$ (transição pode rotular aresta).
Folhas: ou o estado está em H, ou simplesmente não tem o que fazer.

M aceita w se alguma computação de (M, w) para em algum estado de H.

MT não determinística

$M = (K, \Sigma, \Delta, s, H)$. A diferença com MT é que
$\Delta \subseteq (K \backslash H \times \Sigma \cup \{\triangleright\}) \times (K \times \Sigma \cup \{\leftarrow, \rightarrow\})$.
Configuraçoes, $\vdash_M$, $\vdash_M^*$ como antes.

A partir de uma configuração, existe uma árvore de computações, que pode ter caminhos infinitos.
Nós: configurações.
Filho dada por $\vdash_M$ (transição pode rotular aresta).
Folhas: ou o estado está em H, ou simplesmente não tem o que fazer.

M aceita w se alguma computação de (M, w) para em algum estado de H.

M semidecide a linguagem $\{w \mid M \text{ aceita } w\}$.

Complexidade de MTND

Complexidade de MTND

Uma MTND $\mathscr{M}$ para sempre se para qualquer entrada x, a árvore de computação de $(\mathscr{M}, x)$ é finita.

Complexidade de MTND

Uma MTND $\mathscr{M}$ para sempre se para qualquer entrada x, a árvore de computação de $(\mathscr{M}, x)$ é finita.

Seja $\mathscr{M}$ uma MTND que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máxima profundidade da árvore de computação para uma entrada de tamanho n.

Complexidade de MTND

Uma MTND $\mathscr{M}$ para sempre se para qualquer entrada x, a árvore de computação de $(\mathscr{M}, x)$ é finita.

Seja $\mathscr{M}$ uma MTND que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máxima profundidade da árvore de computação para uma entrada de tamanho n.

Se $\mathscr{M}$ é uma MTND com $H = \{h\}$ que para sempre, $L(\mathscr{M})$ consiste das palavras para as quais alguma computação termina em h; $\mathscr{M}$ decide $L(\mathscr{M})$:

Se $x \in L(\mathscr{M})$, alguma computação aceita x.

Complexidade de MTND

Uma MTND $\mathscr{M}$ para sempre se para qualquer entrada x, a árvore de computação de $(\mathscr{M}, x)$ é finita.

Seja $\mathscr{M}$ uma MTND que para sempre. Sua complexidade de tempo é a função $f : \mathbb{N} \to \mathbb{N}$ dada por $f(n) =$ máxima profundidade da árvore de computação para uma entrada de tamanho n.

Se $\mathscr{M}$ é uma MTND com $H = \{h\}$ que para sempre, $L(\mathscr{M})$ consiste das palavras para as quais alguma computação termina em h; $\mathscr{M}$ decide $L(\mathscr{M})$:

Se $x \in L(\mathscr{M})$, alguma computação aceita x.

Se $x \notin L(\mathscr{M})$, toda computação rejeita x.

A classe NP

Definição

NP *é a classe das linguagens decidiveis por máquinas de Turing não determinísticas que rodam em tempo polinomial.*

A classe NP

Definição

NP *é a classe das linguagens decidiveis por máquinas de Turing não determinísticas que rodam em tempo polinomial.*

A classe NP

Definição

NP *é a classe das linguagens decidiveis por máquinas de Turing não determinísticas que rodam em tempo polinomial.*

Claro: $\mathbf{P} \subseteq \mathbf{NP}$.

A classe NP

Definição

NP é a classe das linguagens decidiveis por máquinas de Turing não determinísticas que rodam em tempo polinomial.

Claro: $\mathbf{P} \subseteq \mathbf{NP}$.

Teorema

Se $L \in \mathbf{NP}$, então existe k tal que $L \in \textsc{time}(2^{n^k})$.

Exemplos em NP

- SAT: gere todas as atribuições, e avalie.

Exemplos em NP

- SAT: gere todas as atribuições, e avalie.
- HAMILTONIANO: gere todas as permutações de vértices, e confira.

Exemplos em NP

- SAT: gere todas as atribuições, e avalie.
- HAMILTONIANO: gere todas as permutações de vértices, e confira.
- CAIXEIRO VIAJANTE: idem.

Exemplos em NP

- SAT: gere todas as atribuições, e avalie.
- HAMILTONIANO: gere todas as permutações de vértices, e confira.
- CAIXEIRO VIAJANTE: idem.

Exemplos em NP

- SAT: gere todas as atribuições, e avalie.
- HAMILTONIANO: gere todas as permutações de vértices, e confira.
- CAIXEIRO VIAJANTE: idem.

Em todas elas se gera uma "prova" de que a propriedade vale.

Certificados

Da Lista3:

Certificados

Da Lista3:

Teorema

Uma linguagem $L \subseteq \Sigma^*$ *é recursivamente enumerável sse existe uma linguagem recursiva* $H \subseteq \Sigma^* \# \Sigma^*$ *tal que*
$L = \{x \in \Sigma^* \mid \text{existe } y \in \Sigma^* \text{ tal que } x\#y \in H\}$.

Certificados

Da Lista3:

Teorema

Uma linguagem $L \subseteq \Sigma^*$ é recursivamente enumerável sse existe uma linguagem recursiva $H \subseteq \Sigma^* \# \Sigma^*$ tal que
$L = \{x \in \Sigma^* \mid \text{existe } y \in \Sigma^* \text{ tal que } x\#y \in H\}$.

Certificados

Da Lista3:

Teorema

Uma linguagem $L \subseteq \Sigma^*$ é recursivamente enumerável sse existe uma linguagem recursiva $H \subseteq \Sigma^* \# \Sigma^*$ tal que
$L = \{x \in \Sigma^* \mid \text{existe } y \in \Sigma^* \text{ tal que } x\#y \in H\}$.

Se $x\#y \in H$, y é um certificado de que $x \in L$.

Certificados

Da Lista3:

Teorema

Uma linguagem $L \subseteq \Sigma^*$ é recursivamente enumerável sse existe uma linguagem recursiva $H \subseteq \Sigma^* \# \Sigma^*$ tal que
$L = \{x \in \Sigma^* \mid \text{existe } y \in \Sigma^* \text{ tal que } x\#y \in H\}$.

Se $x\#y \in H$, y é um certificado de que $x \in L$.

Dem: Suponha que existe H, decidida por $\mathscr{M}$. Então L é semidecidida por:

para cada $y \in \Sigma^*$: **se** $x\#y \in H$ **pare**

Certificados

Da Lista3:

Teorema

Uma linguagem $L \subseteq \Sigma^*$ é recursivamente enumerável sse existe uma linguagem recursiva $H \subseteq \Sigma^* \# \Sigma^*$ tal que
$L = \{x \in \Sigma^* \mid \text{existe } y \in \Sigma^* \text{ tal que } x\#y \in H\}$.

Se $x\#y \in H$, y é um certificado de que $x \in L$.

Dem: Suponha que existe H, decidida por $\mathscr{M}$. Então L é semidecidida por:

para cada $y \in \Sigma^*$: **se** $x\#y \in H$ **pare**

Suponha agora que M semidecide L. Um certificado para x é a computação de M com entrada x.

Certificados sucintos

Certificados sucintos

Uma linguagem $H \subseteq \Sigma^*\#\Sigma^*$ (com $\# \notin \Sigma$) é polinomialmente balanceada se existe um polinômio $p(n)$ tal que $x\#y \in H$ implica $|y| \leq p(|x|)$.

Certificados sucintos

Uma linguagem $H \subseteq \Sigma^*\#\Sigma^*$ (com $\# \notin \Sigma$) é polinomialmente balanceada se existe um polinômio $p(n)$ tal que $x\#y \in H$ implica $|y| \leq p(|x|)$.

Teorema

Seja $L \subseteq \Sigma^*$, $|\Sigma| \geq 2$. Então, $L \in \mathbf{NP}$ sse existe $H \subseteq \Sigma^*\#\Sigma^*$ polinomialmente balanceada tal que $H \in \mathbf{P}$ e $L = \{x \in \Sigma^* \mid \text{existe } y \in \Sigma^* \text{ tal que } x\#y \in H\}$.

Certificados sucintos

Uma linguagem $H \subseteq \Sigma^*\#\Sigma^*$ (com $\# \notin \Sigma$) é polinomialmente balanceada se existe um polinômio $p(n)$ tal que $x\#y \in H$ implica $|y| \leq p(|x|)$.

Teorema

*Seja $L \subseteq \Sigma^*$, $|\Sigma| \geq 2$. Então, $L \in$ **NP** sse existe $H \subseteq \Sigma^*\#\Sigma^*$ polinomialmente balanceada tal que $H \in$ **P** e $L = \{x \in \Sigma^* \mid$ existe $y \in \Sigma^*$ tal que $x\#y \in H\}$.*

Certificados sucintos

Uma linguagem $H \subseteq \Sigma^*\#\Sigma^*$ (com $\# \notin \Sigma$) é polinomialmente balanceada se existe um polinômio $p(n)$ tal que $x\#y \in H$ implica $|y| \leq p(|x|)$.

Teorema

*Seja $L \subseteq \Sigma^*$, $|\Sigma| \geq 2$. Então, $L \in$ **NP** sse existe $H \subseteq \Sigma^*\#\Sigma^*$ polinomialmente balanceada tal que $H \in$ **P** e $L = \{x \in \Sigma^* \mid$ existe $y \in \Sigma^*$ tal que $x\#y \in H\}$.*

H é tal que existe um algoritmo que decide se $x\#y \in H$ em tempo polinomial em $|x|$.

Certificados sucintos

Uma linguagem $H \subseteq \Sigma^*\#\Sigma^*$ (com $\# \notin \Sigma$) é polinomialmente balanceada se existe um polinômio $p(n)$ tal que $x\#y \in H$ implica $|y| \leq p(|x|)$.

Teorema

Seja $L \subseteq \Sigma^*$, $|\Sigma| \geq 2$. Então, $L \in \mathbf{NP}$ sse existe $H \subseteq \Sigma^*\#\Sigma^*$ polinomialmente balanceada tal que $H \in \mathbf{P}$ e $L = \{x \in \Sigma^* \mid \text{existe } y \in \Sigma^* \text{ tal que } x\#y \in H\}$.

H é tal que existe um algoritmo que decide se $x\#y \in H$ em tempo polinomial em $|x|$.

Se $x\#y \in H$, y é um certificado sucinto de que $x \in L$.

Demonstração

Demonstração

Dem: Suponha que existe H, decidida em tempo polinomial por $\mathscr{M}$. Então L é decidida não-deterministicamente por:

$p(n)$

gere $y \in \Sigma^*$ com $|y| \leq p(|x|)$:
se $x\#y \in H$ **aceite e pare**
rejeite

Demonstração

Dem: Suponha que existe H, decidida em tempo polinomial por $\mathscr{M}$. Então L é decidida não-deterministicamente por:

gere $y \in \Sigma^*$ com $|y| \leq p(|x|)$:
se $x\#y \in H$ **aceite e pare**
rejeite

Suponha agora que a MTND M decide L em tempo $p(n)$. Um certificado para x é uma computação de M com entrada x que aceita x.

$O(p(n)^2)$

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE
- n é composto

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE
- n é composto
- n é primo (Miller 1976)

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE
- n é composto
- n é primo (Miller 1976)
- Dado sistema $Ax = b$ com coeficientes inteiros, existe solução em inteiros não-negativos? Analogamente para qualquer sistema de equações ou desigualdades.

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE
- n é composto
- n é primo (Miller 1976)
- Dado sistema $Ax = b$ com coeficientes inteiros, existe solução em inteiros não-negativos? Analogamente para qualquer sistema de equações ou desigualdades.
- Vários parâmetros de grafos.

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE
- n é composto
- n é primo (Miller 1976)
- Dado sistema $Ax = b$ com coeficientes inteiros, existe solução em inteiros não-negativos? Analogamente para qualquer sistema de equações ou desigualdades.
- Vários parâmetros de grafos.
- Várias questões da forma "existe ?".

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE
- n é composto
- n é primo (Miller 1976)
- Dado sistema $Ax = b$ com coeficientes inteiros, existe solução em inteiros não-negativos? Analogamente para qualquer sistema de equações ou desigualdades.
- Vários parâmetros de grafos.
- Várias questões da forma "existe ?".

Exemplos de problemas em NP

- SAT, HAMILTONIANO, CAIXEIRO VIAJANTE
- n é composto
- n é primo (Miller 1976)
- Dado sistema $Ax = b$ com coeficientes inteiros, existe solução em inteiros não-negativos? Analogamente para qualquer sistema de equações ou desigualdades.
- Vários parâmetros de grafos.
- Várias questões da forma "existe ?".
  Se **sim**, o objeto que existe certifica — desde que seja sucinto.

A grande questão

Garey Johnson

Existem montes de problemas importantes
na teoria e na prática que estão em **NP** e
para os quais não se conhece algoritmo polinomial.

A grande questão

Existem montes de problemas importantes
na teoria e na prática que estão em **NP** e
para os quais não se conhece algoritmo polinomial.

$$P = NP?$$

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

Se $L_1 \prec L_2$ e $L_2 \in \mathbf{P}$, então $L_1 \in \mathbf{P}$.

NP-completo

Definição

*Uma linguagem L é **NP**-completa se:*

NP-completo

Definição

*Uma linguagem L é **NP**-completa se:*

1. $L \in \mathbf{NP}$, e

NP-completo

Definição

*Uma linguagem L é **NP**-completa se:*

1. $L \in \mathbf{NP}$, e
2. *Para toda* $H \in \mathbf{NP}$, $H \prec L$.

NP-completo

Definição

*Uma linguagem L é **NP**-completa se:*

1. $L \in \mathbf{NP}$, e
2. *Para toda* $H \in \mathbf{NP}$, $H \prec L$.

Teorema

*Suponha que L é **NP**-completa. Então,*
$\mathbf{P} = \mathbf{NP}$ *se e só se* $L \in \mathbf{P}$.

O Teorema de Cook

Teorema (Cook, 1970)

*SAT é **NP**-completo.*

Dem: Vamos usar

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

$= 1$ sse exatamente um dos x_i vale 1.

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

O Teorema de Cook

Teorema (Cook, 1970)

*SAT é **NP**-completo.*

Dem: Vamos usar

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

$= 1$ sse exatamente um dos x_i vale 1.

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

O Teorema de Cook

Teorema (Cook, 1970)

*SAT é **NP**-completo.*

Dem: Vamos usar

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

$= 1$ sse exatamente um dos x_i vale 1.

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

Se j varia num conjunto conhecido, vamos simplificar a notação:

$$\mathbb{U}(x_1, x_2, \ldots, x_n) \mapsto \mathbb{U}([x_j]_j).$$

O Teorema de Cook

Teorema (Cook, 1970)

*SAT é **NP**-completo.*

Dem: Vamos usar

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

$= 1$ sse exatamente um dos x_i vale 1.

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

Se j varia num conjunto conhecido, vamos simplificar a notação:

$$\mathbb{U}(x_1, x_2, \ldots, x_n) \mapsto \mathbb{U}([x_j]_j).$$

Se $\{F_j\}$ é uma coleção finita de expr. booleanas, a expressão $\prod_j F_j$ equivale a "para todo j, F_j".

Preparando

Preparando

Seja $H \in \mathbf{NP}$. Então existe uma MTND $\mathscr{M}$ e um polinômio $p(n)$ tal que $\mathscr{M}$ decide H em tempo $p(n)$.

Preparando

Seja $H \in \mathbf{NP}$. Então existe uma MTND $\mathscr{M}$ e um polinômio $p(n)$ tal que $\mathscr{M}$ decide H em tempo $p(n)$.

Redução: vamos construir, para cada $x \in \Sigma^*$, uma expressão booleana $\mathscr{E}(x)$ tal que $x \in H$ sse $\mathscr{E}$ tem uma avaliação que dá 1.

Preparando

Seja $H \in \mathbf{NP}$. Então existe uma MTND $\mathscr{M}$ e um polinômio $p(n)$ tal que $\mathscr{M}$ decide H em tempo $p(n)$.

Redução: vamos construir, para cada $x \in \Sigma^*$, uma expressão booleana $\mathscr{E}(x)$ tal que $x \in H$ sse $\mathscr{E}$ tem uma avaliação que dá 1.

Seja $n = |x|$. Dada uma computação de H com entrada x, sabemos que no máximo $p(n)$ células da fita serão usadas, e a computação para em $\leq p(n)$ passos.

Preparando

Seja $H \in \mathbf{NP}$. Então existe uma MTND $\mathscr{M}$ e um polinômio $p(n)$ tal que $\mathscr{M}$ decide H em tempo $p(n)$.

Redução: vamos construir, para cada $x \in \Sigma^*$, uma expressão booleana $\mathscr{E}(x)$ tal que $x \in H$ sse $\mathscr{E}$ tem uma avaliação que dá 1.

Seja $n = |x|$. Dada uma computação de H com entrada x, sabemos que no máximo $p(n)$ células da fita serão usadas, e a computação para em $\leq p(n)$ passos.

Truque: modifique H para entrar em loop quando chega em h; pare a computação ao executar exatamente $p(n)$ passos.

Continuando

Continuando

O trace: considere a matriz com $p(n)$ linhas onde cada linha é uma fita de $p(n)$ posições. Vamos entende a linha t como sendo a fita no instante t; $[ti]$ é o conteúdo da célula i no instante t.

Continuando

O trace: considere a matriz com $p(n)$ linhas onde cada linha é uma fita de $p(n)$ posições. Vamos entende a linha t como sendo a fita no instante t; $[ti]$ é o conteúdo da célula i no instante t.

Numerar

- os estados de $\mathscr{M}$ como $q_1, \ldots, q_s$, com estado inicial q_1, estado de aceitação q_n;

Continuando

O trace: considere a matriz com $p(n)$ linhas onde cada linha é uma fita de $p(n)$ posições. Vamos entende a linha t como sendo a fita no instante t; $[ti]$ é o conteúdo da célula i no instante t.

Numerar

- os estados de $\mathscr{M}$ como $q_1, \ldots, q_s$, com estado inicial q_1, estado de aceitação q_m;
- os símbolos da fita como $a_1, \ldots, a_m$.

Variáveis

Variáveis

Vamos definir algumas variáveis, e a intenção semântica:

1. $C\langle i, j, t\rangle$ vale 1 se $[ti] = a_j$.
   $1 \leq i \leq p(n), 1 \leq j \leq m, 0 \leq t \leq p(n)$.

Variáveis

Vamos definir algumas variáveis, e a intenção semântica:

1. $C\langle i, j, t\rangle$ vale 1 se $[ti] = a_j$.
   $1 \leq i \leq p(n), 1 \leq j \leq m, 0 \leq t \leq p(n)$.
2. $S\langle k, t\rangle$ vale 1 se $\mathscr{M}$ está no estado q_k no instante t.
   $1 \leq k \leq s, 0 \leq t \leq p(n)$.

$O(p(n))$

Variáveis

Vamos definir algumas variáveis, e a intenção semântica:

1. $C\langle i, j, t\rangle$ vale 1 se $[ti] = a_j$.
   $1 \leq i \leq p(n), 1 \leq j \leq m, 0 \leq t \leq p(n)$.
2. $S\langle k, t\rangle$ vale 1 se $\mathscr{M}$ está no estado q_k no instante t.
   $1 \leq k \leq s, 0 \leq t \leq p(n)$.
3. $H\langle i, t\rangle$ vale 1 se no instante t o cursor está na posição i.
   $1 \leq i \leq p(n), 0 \leq t \leq p(n)$.

Variáveis

Vamos definir algumas variáveis, e a intenção semântica:

1. $C\langle i, j, t\rangle$ vale 1 se $[ti] = a_j$.
   $1 \leq i \leq p(n), 1 \leq j \leq m, 0 \leq t \leq p(n)$.
2. $S\langle k, t\rangle$ vale 1 se $\mathscr{M}$ está no estado q_k no instante t.
   $1 \leq k \leq s, 0 \leq t \leq p(n)$.
3. $H\langle i, t\rangle$ vale 1 se no instante t o cursor está na posição i.
   $1 \leq i \leq p(n), 0 \leq t \leq p(n)$.

Variáveis

Vamos definir algumas variáveis, e a intenção semântica:

1. $C\langle i, j, t\rangle$ vale 1 se $[ti] = a_j$.
   $1 \leq i \leq p(n), 1 \leq j \leq m, 0 \leq t \leq p(n)$.
2. $S\langle k, t\rangle$ vale 1 se $\mathscr{M}$ está no estado q_k no instante t.
   $1 \leq k \leq s, 0 \leq t \leq p(n)$.
3. $H\langle i, t\rangle$ vale 1 se no instante t o cursor está na posição i.
   $1 \leq i \leq p(n), 0 \leq t \leq p(n)$.

Total de $\mathcal{O}(p(n)^2)$ variáveis.

$C_0, \ldots, C_{p(n)}$ é uma computação que aceita x sse:

1. O cursor está numa única posição a cada instante.

$C_0, \ldots, C_{p(n)}$ é uma computação que aceita x sse:

1. O cursor está numa única posição a cada instante.
2. A cada instante cada célula tem um único símbolo.

$C_0, \ldots, C_{p(n)}$ é uma computação que aceita x sse:

1. O cursor está numa única posição a cada instante.
2. A cada instante cada célula tem um único símbolo.
3. A cada instante $\mathscr{M}$ está num único estado.

$C_0, \ldots, C_{p(n)}$ é uma computação que aceita x sse:

1. O cursor está numa única posição a cada instante.
2. A cada instante cada célula tem um único símbolo.
3. A cada instante $\mathscr{M}$ está num único estado.
4. De um instante para o seguinte, no máximo uma célula muda.

$C_0, \ldots, C_{p(n)}$ é uma computação que aceita x sse:

1. O cursor está numa única posição a cada instante.
2. A cada instante cada célula tem um único símbolo.
3. A cada instante $\mathscr{M}$ está num único estado.
4. De um instante para o seguinte, no máximo uma célula muda.
5. De um instante para o seguinte, o que muda segue as instruções de $\mathscr{M}$.

$C_0, \ldots, C_{p(n)}$ é uma computação que aceita x sse:

1. O cursor está numa única posição a cada instante.
2. A cada instante cada célula tem um único símbolo.
3. A cada instante $\mathscr{M}$ está num único estado.
4. De um instante para o seguinte, no máximo uma célula muda.
5. De um instante para o seguinte, o que muda segue as instruções de $\mathscr{M}$.
6. No instante 0 a fita representa $\triangleright \underline{\sqcup} x$, completada com $\sqcup$'s, e o estado é q_0.

$C_0, \ldots, C_{p(n)}$ é uma computação que aceita x sse:

1. O cursor está numa única posição a cada instante.
2. A cada instante cada célula tem um único símbolo.
3. A cada instante $\mathscr{M}$ está num único estado.
4. De um instante para o seguinte, no máximo uma célula muda.
5. De um instante para o seguinte, o que muda segue as instruções de $\mathscr{M}$.
6. No instante 0 a fita representa $\triangleright \underline{\sqcup} x$, completada com $\sqcup$'s, e o estado é q_0.
7. No instante $p(n)$ o estado é q_{sim}.

Implementação

Implementação

1) $A = \prod_t \mathbb{U}([H\langle i, t\rangle]_i)$ compr. $\mathcal{O}(p(n)^3)$.

Implementação

1) $A = \prod_t \mathbb{U}([H\langle i, t\rangle]_i)$ compr. $\mathcal{O}(p(n)^3)$.

2) $B = \prod_t \prod_i \mathbb{U}([C\langle i, j, t\rangle]_j)$ compr. $\mathcal{O}(p(n)^2)$.

Implementação

1) $A = \prod_t \mathbb{U}([H\langle i, t\rangle]_i)$ compr. $\mathcal{O}(p(n)^3)$.

2) $B = \prod_t \prod_i \mathbb{U}([C\langle i, j, t\rangle]_j)$ compr. $\mathcal{O}(p(n)^2)$.

3) $C = \prod_t \mathbb{U}([S\langle k, t\rangle]_k)$ compr. $\mathcal{O}(p(n))$.

Implementação

1) $A = \prod_t \mathbb{U}([H\langle i,t\rangle]_i)$ compr. $\mathcal{O}(p(n)^3)$.

2) $B = \prod_t \prod_i \mathbb{U}([C\langle i,j,t\rangle]_j)$ compr. $\mathcal{O}(p(n)^2)$.

3) $C = \prod_t \mathbb{U}([S\langle k,t\rangle]_k)$ compr. $\mathcal{O}(p(n))$.

4) $D = \prod_{i,j,t} (C\langle i,j,t\rangle \equiv C\langle i,j,t+1\rangle) + H\langle i,t\rangle$ compr. $\mathcal{O}(p(n)^2)$.

$$x \equiv y \quad = \quad xy + \bar{x}\bar{y}$$

5) Expressão E_{ijkt} significa

- No instante t a célula i não contem o símbolo j, ou o cursor não está na posição i, ou $\mathscr{M}$ não está no estado k, ou

5) Expressão E_{ijkt} significa

- No instante t a célula i não contem o símbolo j, ou o cursor não está na posição i, ou $\mathscr{M}$ não está no estado k, ou
- A próxima configuração segue da instrução aplicável de $\mathscr{M}$.

5) Expressão E_{ijkt} significa

- No instante t a célula i não contem o símbolo j, ou o cursor não está na posição i, ou $\mathscr{M}$ não está no estado k, ou
- A próxima configuração segue da instrução aplicável de $\mathscr{M}$.

5) Expressão E_{ijkt} significa

- No instante t a célula i não contem o símbolo j, ou o cursor não está na posição i, ou $\mathscr{M}$ não está no estado k, ou
- A próxima configuração segue da instrução aplicável de $\mathscr{M}$.

$$E_{ijkt} = \overline{C\langle i,j,t\rangle} + \overline{H\langle i,t\rangle} + \overline{S\langle k,t\rangle} + \sum_\ell C\langle i,j_\ell,t+1\rangle S\langle k_\ell,t+1\rangle H\langle i_l,t+1\rangle$$

onde ℓ percorre todas as quádruplas da forma $(q_i, a_j, k_\ell, \bullet)$.
Se $\bullet$ é a letra a_{j_ℓ} isso afeta o termo $C\langle\rangle$.
Se $\bullet$ é flecha, isso afeta o termo $H\langle\rangle$.

5) Expressão E_{ijkt} significa

- No instante t a célula i não contem o símbolo j, ou o cursor não está na posição i, ou $\mathscr{M}$ não está no estado k, ou
- A próxima configuração segue da instrução aplicável de $\mathscr{M}$.

$$E_{ijkt} = \overline{C\langle i,j,t\rangle} + \overline{H\langle i,t\rangle} + \overline{S\langle k,t\rangle} + \sum_\ell C\langle i,j_\ell,t+1\rangle S\langle k_\ell,t+1\rangle H\langle i_l,t+1\rangle$$

onde ℓ percorre todas as quádruplas da forma $(q_i, a_j, k_\ell, \bullet)$.
Se $\bullet$ é a letra a_{j_ℓ} isso afeta o termo $C\langle\rangle$.
Se $\bullet$ é flecha, isso afeta o termo $H\langle\rangle$.

$$E = \prod_{ijkt} E_{ijkt}$$

compr. $\mathcal{O}(p(n)^2)$.

Finalmente

Finalmente

6) Sejam $w_1, \ldots, w_{p(n)}$ os índices dos símbolos da fita inicial.

$$F = S\langle 1,0\rangle H\langle 2,0\rangle \prod_j C\langle i, w_j, 0\rangle$$

compr. $\mathcal{O}(p(n))$.

Finalmente

6) Sejam $w_1, \ldots, w_{p(n)}$ os índices dos símbolos da fita inicial.
$F = S\langle 1,0\rangle H\langle 2,0\rangle \prod_j C\langle i, w_j, 0\rangle$ compr. $\mathcal{O}(p(n))$.

7) $G = S\langle m, p(n)\rangle$

Finalmente

6) Sejam $w_1, \ldots, w_{p(n)}$ os índices dos símbolos da fita inicial.
$F = S\langle 1,0\rangle H\langle 2,0\rangle \prod_j C\langle i, w_j, 0\rangle$ compr. $\mathcal{O}(p(n))$.

7) $G = S\langle m, p(n)\rangle$

E então:

$$\mathscr{E}(x) = ABCDEF$$

Um pouco de história

Um pouco de história

Antes de 1960: algo sobre eficiência, informal.

Um pouco de história

Antes de 1960: algo sobre eficiência, informal.
~1965 Hartmanis, Stearns, Blum: complexidade.

Um pouco de história

Antes de 1960: algo sobre eficiência, informal.

~1965 Hartmanis, Stearns, Blum: complexidade.

1965: Edmonds sugere "eficiente"=**P**, certificados sucintos, questionando se existência de certificado implica a de algoritmo. Paper famoso.

Um pouco de história

Antes de 1960: algo sobre eficiência, informal.

~1965 Hartmanis, Stearns, Blum: complexidade.

1965: Edmonds sugere "eficiente"=P, certificados sucintos, questionando se existência de certificado implica a de algoritmo. Paper famoso.

1965: Cobham apresenta idéias semelhantes em termos de funções recursivas. Ninguém notou.

Um pouco de história

Antes de 1960: algo sobre eficiência, informal.

~1965 Hartmanis, Stearns, Blum: complexidade.

1965: Edmonds sugere "eficiente"=**P**, certificados sucintos, questionando se existência de certificado implica a de algoritmo. Paper famoso.

1965: Cobham apresenta idéias semelhantes em termos de funções recursivas. Ninguém notou.

1971: Cook apresenta seu teorema. Muitos viram, acharam interessante em princípio.

Um pouco de história

Antes de 1960: algo sobre eficiência, informal.

~1965 Hartmanis, Stearns, Blum: complexidade.

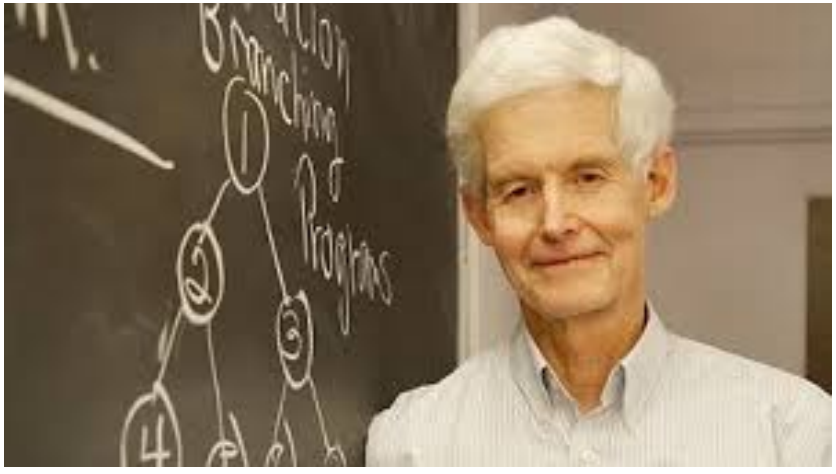
1965: Edmonds sugere "eficiente"=**P**, certificados sucintos, questionando se existência de certificado implica a de algoritmo. Paper famoso.

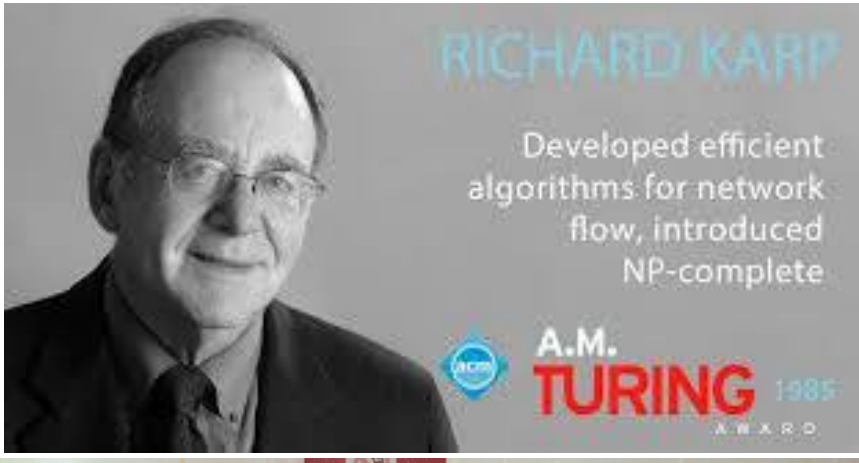
1965: Cobham apresenta idéias semelhantes em termos de funções recursivas. Ninguém notou.

1971: Cook apresenta seu teorema. Muitos viram, acharam interessante em princípio.

1972: Karp, que entendeu o Cook, mostra que um monte de problemas importantes eram **NP**-completos. O assunto pegou fogo!

Estão vivos




RICHARD KARP
Developed efficient algorithms for network flow, introduced NP-complete
A.M. TURING
AWARD