

Projeto e Análise de Algoritmos II (MC558)

Classes de Problemas

Prof. Dr. Ruben Interian

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Classes \mathcal{P} , \mathcal{NP} , \mathcal{NP} -difícil, \mathcal{NP} -completo
- 3 Máquina de Turing
- 4 Teorema de Cook
- 5 Síntese

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Classes \mathcal{P} , \mathcal{NP} , \mathcal{NP} -difícil, \mathcal{NP} -completo
- 3 Máquina de Turing
- 4 Teorema de Cook
- 5 Síntese

Revisão do conteúdo

- O estudo de classes de complexidade é feito para **problemas de decisão** (saída SIM ou NÃO). Se resolvemos eficientemente o problema de decisão associado a **P**, podemos resolver eficientemente **P**!

Revisão do conteúdo

- O estudo de classes de complexidade é feito para **problemas de decisão** (saída SIM ou NÃO). Se resolvemos eficientemente o problema de decisão associado a **P**, podemos resolver eficientemente **P**!
- Um exemplo muito importante de problemas difíceis é o problema da satisfatibilidade (**SAT**, *Satisfiability*), que é um problema de decisão.

Revisão do conteúdo

- O estudo de classes de complexidade é feito para **problemas de decisão** (saída SIM ou NÃO). Se resolvemos eficientemente o problema de decisão associado a **P**, podemos resolver eficientemente **P**!
- Um exemplo muito importante de problemas difíceis é o problema da satisfatibilidade (**SAT**, *Satisfiability*), que é um problema de decisão.
- Um algoritmo **não-determinístico** permite o uso do comando **Escolha(S)**. **SAT** pode ser resolvido de forma eficiente por um algoritmo polinomial **não-determinístico**.

Revisão do conteúdo

- O estudo de classes de complexidade é feito para **problemas de decisão** (saída SIM ou NÃO). Se resolvemos eficientemente o problema de decisão associado a \mathbf{P} , podemos resolver eficientemente \mathbf{P} !
- Um exemplo muito importante de problemas difíceis é o problema da satisfatibilidade (**SAT**, *Satisfiability*), que é um problema de decisão.
- Um algoritmo **não-determinístico** permite o uso do comando **Escolha(S)**. **SAT** pode ser resolvido de forma eficiente por um algoritmo polinomial **não-determinístico**.
- É uma questão em aberto se os problemas que os algoritmos **não-determinísticos** resolvem em tempo polinomial (\mathcal{NP}), podem ser resolvidos em tempo polinomial por algoritmos **determinísticos**: $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$. A maioria acredita que $\mathcal{P} \neq \mathcal{NP}$.

Objetivo

Responder à pergunta:

- Como identificar um problema complexo? Como mostrar que ele é complexo?
 - “Eu não consigo resolver” não é suficiente!

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Classes \mathcal{P} , \mathcal{NP} , \mathcal{NP} -difícil, \mathcal{NP} -completo
- 3 Máquina de Turing
- 4 Teorema de Cook
- 5 Síntese

Classes \mathcal{P} e \mathcal{NP}

- **Definição:** \mathcal{P} é o conjunto de problemas que podem ser resolvidos por um **algoritmo determinístico** em tempo polinomial.
- **Definição:** \mathcal{NP} é o conjunto de problemas que podem ser resolvidos por um **algoritmo não-determinístico** em tempo polinomial.

Classes \mathcal{P} e \mathcal{NP}

- **Definição:** \mathcal{P} é o conjunto de problemas que podem ser resolvidos por um **algoritmo determinístico** em tempo polinomial.
- **Definição:** \mathcal{NP} é o conjunto de problemas que podem ser resolvidos por um **algoritmo não-determinístico** em tempo polinomial.
- Como todo algoritmo determinístico é um caso particular de um algoritmo não-determinístico, $\mathcal{P} \subseteq \mathcal{NP}$.

Todos os problemas que possuem algoritmos polinomiais estão em \mathcal{P} e \mathcal{NP} .
Vimos que **CLIQUE** e **SAT** estão em \mathcal{NP} .

Classes \mathcal{P} e \mathcal{NP}

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

Classes \mathcal{P} e \mathcal{NP}

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- A maioria dos cientistas acredita que $\mathcal{P} \neq \mathcal{NP}$.

Classes \mathcal{P} e \mathcal{NP}

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- A maioria dos cientistas acredita que $\mathcal{P} \neq \mathcal{NP}$.
- Como mostrar que $\mathcal{P} \neq \mathcal{NP}$?

*Mostrar que existe um problema $A \in \mathcal{NP}$ tal que **nenhum** algoritmo determinístico polinomial pode resolver A .*

Classes \mathcal{P} e \mathcal{NP}

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- A maioria dos cientistas acredita que $\mathcal{P} \neq \mathcal{NP}$.
- Como mostrar que $\mathcal{P} \neq \mathcal{NP}$?
*Mostrar que existe um problema $A \in \mathcal{NP}$ tal que **nenhum** algoritmo determinístico polinomial pode resolver A .*
- Como mostrar que $\mathcal{P} = \mathcal{NP}$?
([Aula anterior](#)) *Mostrar que todo problema $B \in \mathcal{NP}$ possui um algoritmo determinístico polinomial que o resolve.*

Classes \mathcal{P} e \mathcal{NP}

- Questão fundamental da Computação:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

- A maioria dos cientistas acredita que $\mathcal{P} \neq \mathcal{NP}$.
- Como mostrar que $\mathcal{P} \neq \mathcal{NP}$?
*Mostrar que existe um problema $A \in \mathcal{NP}$ tal que **nenhum** algoritmo determinístico polinomial pode resolver A .*
- Como mostrar que $\mathcal{P} = \mathcal{NP}$?
(Aula anterior) *Mostrar que todo problema $B \in \mathcal{NP}$ possui um algoritmo determinístico polinomial que o resolve.*
 - Na verdade, veremos que é suficiente que apenas um problema, de uma classe específica, tenha algoritmo determinístico polinomial, para mostrar que $\mathcal{P} = \mathcal{NP}$.

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

- Suponha que conseguimos encontrar um problema $A \in \mathcal{NP}$ tal que **todo problema** $B \in \mathcal{NP}$ se reduz em tempo polinomial a A .

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

- Suponha que conseguimos encontrar um problema $A \in \mathcal{NP}$ tal que **todo problema** $B \in \mathcal{NP}$ se reduz em tempo polinomial a A .
 - A e B são problemas de decisão. A redução mencionada é uma **redução de Karp**.
Lembrando a redução de Karp: Alg_B é usado uma única vez; Alg_B responde SIM para I_B se e somente se I_A é uma instância SIM para A .
 - O termo **redução** é usado daqui em diante para referir-se à **redução de Karp**.

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

- Suponha que conseguimos encontrar um problema $A \in \mathcal{NP}$ tal que **todo problema** $B \in \mathcal{NP}$ se reduz em tempo polinomial a A .
 - A e B são problemas de decisão. A redução mencionada é uma **redução de Karp**.
Lembrando a redução de Karp: Alg_B é usado uma única vez; Alg_B responde SIM para I_B **se e somente se** I_A é uma instância SIM para A .
 - O termo **redução** é usado daqui em diante para referir-se à **redução de Karp**.
- Assim, teríamos um problema $A \in \mathcal{NP}$ tal que, se $A \in \mathcal{P}$, então todo problema em \mathcal{NP} também está em \mathcal{P} !

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -difícil

A é um problema \mathcal{NP} -difícil se todo problema de \mathcal{NP} se reduz polinomialmente a A .

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -difícil

A é um problema \mathcal{NP} -difícil se todo problema de \mathcal{NP} se reduz polinomialmente a A .

- Informalmente, se A for \mathcal{NP} -difícil, então resolver A deve ser pelo menos tão difícil quanto resolver qualquer problema em \mathcal{NP} .

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -difícil

A é um problema \mathcal{NP} -difícil se todo problema de \mathcal{NP} se reduz polinomialmente a A .

- Informalmente, se A for \mathcal{NP} -difícil, então resolver A deve ser pelo menos tão difícil quanto resolver qualquer problema em \mathcal{NP} .
- O problema A pode estar ou pode não estar em \mathcal{NP} !

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -difícil

A é um problema \mathcal{NP} -difícil se todo problema de \mathcal{NP} se reduz polinomialmente a A .

- Informalmente, se A for \mathcal{NP} -difícil, então resolver A deve ser pelo menos tão difícil quanto resolver qualquer problema em \mathcal{NP} .
- O problema A pode estar ou pode não estar em \mathcal{NP} !
- Se resolvemos de forma eficiente $A \in \mathcal{NP}$ -difícil, resolveremos de forma eficiente qualquer problema $B \in \mathcal{NP}$. Ou seja, se for encontrado **apenas um algoritmo polinomial** para **apenas um problema \mathcal{NP} -difícil**, ficará provado que $\mathcal{P} = \mathcal{NP}$.

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -difícil

A é um problema \mathcal{NP} -difícil se todo problema de \mathcal{NP} se reduz polinomialmente a A .

- Informalmente, se A for \mathcal{NP} -difícil, então resolver A deve ser pelo menos tão difícil quanto resolver qualquer problema em \mathcal{NP} .
- O problema A pode estar ou pode não estar em \mathcal{NP} !
- Se resolvemos de forma eficiente $A \in \mathcal{NP}$ -difícil, resolveremos de forma eficiente qualquer problema $B \in \mathcal{NP}$. Ou seja, se for encontrado **apenas um algoritmo polinomial** para **apenas um problema \mathcal{NP} -difícil**, ficará provado que $\mathcal{P} = \mathcal{NP}$.

Ótimo, mas onde podemos encontrar um problema \mathcal{NP} -difícil?

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -completo

A é um problema \mathcal{NP} -completo (ou A está em \mathcal{NP} -completo) se:

- 1 $A \in \mathcal{NP}$, e
- 2 $A \in \mathcal{NP}$ -difícil.

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -completo

A é um problema \mathcal{NP} -completo (ou A está em \mathcal{NP} -completo) se:

- 1 $A \in \mathcal{NP}$, e
- 2 $A \in \mathcal{NP}$ -difícil.

Observações:

- Por que precisamos desta definição? Há problemas $\notin \mathcal{NP}$?

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -completo

A é um problema \mathcal{NP} -completo (ou A está em \mathcal{NP} -completo) se:

- 1 $A \in \mathcal{NP}$, e
- 2 $A \in \mathcal{NP}$ -difícil.

Observações:

- Por que precisamos desta definição? Há problemas $\notin \mathcal{NP}$? – SIM. **Exemplos:**
 - Problemas que **não são verificáveis** em tempo polinomial. São problemas para os quais não pode ser gerado um certificado curto (certificado conciso, ou polinomial).
 - Problemas **indecidíveis** (não possuem algoritmo que os resolva). → próximas aulas

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -completo

A é um problema \mathcal{NP} -completo (ou A está em \mathcal{NP} -completo) se:

- 1 $A \in \mathcal{NP}$, e
- 2 $A \in \mathcal{NP}$ -difícil.

Observações:

- Por que precisamos desta definição? Há problemas $\notin \mathcal{NP}$? – SIM. **Exemplos:**
 - Problemas que **não são verificáveis** em tempo polinomial. São problemas para os quais não pode ser gerado um certificado curto (certificado conciso, ou polinomial).
 - Problemas **indecidíveis** (não possuem algoritmo que os resolva). → próximas aulas
- **Lembrando:** os problemas P e Q são polinomialmente equivalentes se $P \propto_{\text{poli}} Q$ e $Q \propto_{\text{poli}} P$. Todos os problemas \mathcal{NP} -completo são polinomialmente equivalentes.

Classes \mathcal{NP} -difícil e \mathcal{NP} -completo

\mathcal{NP} -completo

A é um problema \mathcal{NP} -completo (ou A está em \mathcal{NP} -completo) se:

- 1 $A \in \mathcal{NP}$, e
- 2 $A \in \mathcal{NP}$ -difícil.

Observações:

- Por que precisamos desta definição? Há problemas $\notin \mathcal{NP}$? – SIM. **Exemplos:**
 - Problemas que **não são verificáveis** em tempo polinomial. São problemas para os quais não pode ser gerado um certificado curto (certificado conciso, ou polinomial).
 - Problemas **indecidíveis** (não possuem algoritmo que os resolva). → próximas aulas
- **Lembrando:** os problemas P e Q são polinomialmente equivalentes se $P \propto_{\text{poli}} Q$ e $Q \propto_{\text{poli}} P$. Todos os problemas \mathcal{NP} -completos são polinomialmente equivalentes.
- Por definição, \mathcal{NP} -completo $\subseteq \mathcal{NP}$ -difícil.

Provas de \mathcal{NP} -completude

Resultado importante: Seja A um problema \mathcal{NP} -difícil e B um problema em \mathcal{NP} . Se existe uma redução polinomial de A para B ($A \propto_{\text{poli}} B$), então B é \mathcal{NP} -completo.

Provas de \mathcal{NP} -completude

Resultado importante: Seja A um problema \mathcal{NP} -difícil e B um problema em \mathcal{NP} . Se existe uma redução polinomial de A para B ($A \propto_{\text{poli}} B$), então B é \mathcal{NP} -completo.

Dificuldade: encontrar um problema em \mathcal{NP} que seja \mathcal{NP} -difícil, e portanto \mathcal{NP} -completo. Será que ele existe?...

Provas de \mathcal{NP} -completude

Resultado importante: Seja A um problema \mathcal{NP} -difícil e B um problema em \mathcal{NP} . Se existe uma redução polinomial de A para B ($A \propto_{\text{poli}} B$), então B é \mathcal{NP} -completo.

Dificuldade: encontrar um problema em \mathcal{NP} que seja \mathcal{NP} -difícil, e portanto \mathcal{NP} -completo. Será que ele existe?...

Stephen Cook provou em 1971 que **SAT** é \mathcal{NP} -completo.

Teorema de Cook e classe \mathcal{NP}

Cook usou uma **definição equivalente** da classe \mathcal{NP} :

- \mathcal{NP} é o conjunto dos problemas de decisão para os quais existe um algoritmo **determinístico** que, dado uma instância e um certificado conciso, **verifica** em **tempo polinomial** no tamanho da instância se o certificado é **válido** (se ele de fato resolve o problema).

Teorema de Cook e classe \mathcal{NP}

Cook usou uma **definição equivalente** da classe \mathcal{NP} :

- \mathcal{NP} é o conjunto dos problemas de decisão para os quais existe um algoritmo **determinístico** que, dado uma instância e um certificado conciso, **verifica** em **tempo polinomial** no tamanho da instância se o certificado é **válido** (se ele de fato resolve o problema). Veja as **fases de verificação** dos algoritmos ND já vistos!

Teorema de Cook e classe \mathcal{NP}

Cook usou uma **definição equivalente** da classe \mathcal{NP} :

- \mathcal{NP} é o conjunto dos problemas de decisão para os quais existe um algoritmo **determinístico** que, dado uma instância e um certificado conciso, **verifica** em **tempo polinomial** no tamanho da instância se o certificado é **válido** (se ele de fato resolve o problema). Veja as **fases de verificação** dos algoritmos ND já vistos!
- Seja $A \in \mathcal{NP}$ um problema de decisão que admite algoritmo *não-determinístico* polinomial que resolve A . Como a fase de verificação do algoritmo apenas realiza operações determinísticas e é polinomial, o **certificado** $c(x)$ gerado pela fase de construção tem tamanho polinomial no tamanho da instância x : ele é um **certificado conciso**, ou **curto**. Ou seja, $|c(x)| \leq p(|x|)$, onde p é um polinômio.

Teorema de Cook e classe \mathcal{NP}

As duas definições de \mathcal{NP} são equivalentes: definem a mesma classe de problemas!

- \mathcal{NP} é o conjunto de problemas de decisão que admitem um algoritmo não determinístico polinomial.
- \mathcal{NP} é o conjunto de problemas de decisão que admitem um algoritmo verificador determinístico polinomial.

Teorema de Cook e classe \mathcal{NP}

As duas definições de \mathcal{NP} são equivalentes: definem a mesma classe de problemas!

- \mathcal{NP} é o conjunto de problemas de decisão que admitem um algoritmo não determinístico polinomial.
- \mathcal{NP} é o conjunto de problemas de decisão que admitem um algoritmo verificador determinístico polinomial.

Se temos **apenas** um verificador polinomial, é possível definir um algoritmo não determinístico que irá **gerar um certificado válido** em tempo polinomial.

Veja que o certificado é **conciso**!

Teorema de Cook: principais ideias da prova

Ideias da prova de Cook:

- Todo algoritmo determinístico pode ser descrito por um **modelo de computação** conhecido como uma **Máquina de Turing** (MT). Em particular, para qualquer problema \mathcal{NP} , o algoritmo verificador pode ser descrito por este modelo.

Teorema de Cook: principais ideias da prova

Ideias da prova de Cook:

- Todo algoritmo determinístico pode ser descrito por um **modelo de computação** conhecido como uma **Máquina de Turing** (MT). Em particular, para qualquer problema \mathcal{NP} , o algoritmo verificador pode ser descrito por este modelo.
- É possível mostrar que, para cada entrada x da Máquina de Turing (= algoritmo), há uma fórmula booleana \mathcal{F} , polinomial no tamanho de x , tal que \mathcal{F} pode ser satisfeita **se e somente se** a Máquina de Turing retorna Aceitar para x .

Teorema de Cook: principais ideias da prova

Ideias da prova de Cook:

- Todo algoritmo determinístico pode ser descrito por um **modelo de computação** conhecido como uma **Máquina de Turing** (MT). Em particular, para qualquer problema \mathcal{NP} , o algoritmo verificador pode ser descrito por este modelo.
- É possível mostrar que, para cada entrada x da Máquina de Turing (= algoritmo), há uma fórmula booleana \mathcal{F} , polinomial no tamanho de x , tal que \mathcal{F} pode ser satisfeita **se e somente se** a Máquina de Turing retorna Aceitar para x .

Para um determinado algoritmo determinístico, para cada entrada x ,
há uma fórmula booleana “**equivalente**” a eles!

Teorema de Cook: principais ideias da prova

Ideias da prova de Cook:

- Todo algoritmo determinístico pode ser descrito por um **modelo de computação** conhecido como uma **Máquina de Turing** (MT). Em particular, para qualquer problema \mathcal{NP} , o algoritmo verificador pode ser descrito por este modelo.
- É possível mostrar que, para cada entrada x da Máquina de Turing (= algoritmo), há uma fórmula booleana \mathcal{F} , polinomial no tamanho de x , tal que \mathcal{F} pode ser satisfeita **se e somente se** a Máquina de Turing retorna Aceitar para x .

Para um determinado algoritmo determinístico, para cada entrada x ,
há uma fórmula booleana “**equivalente**” a eles!

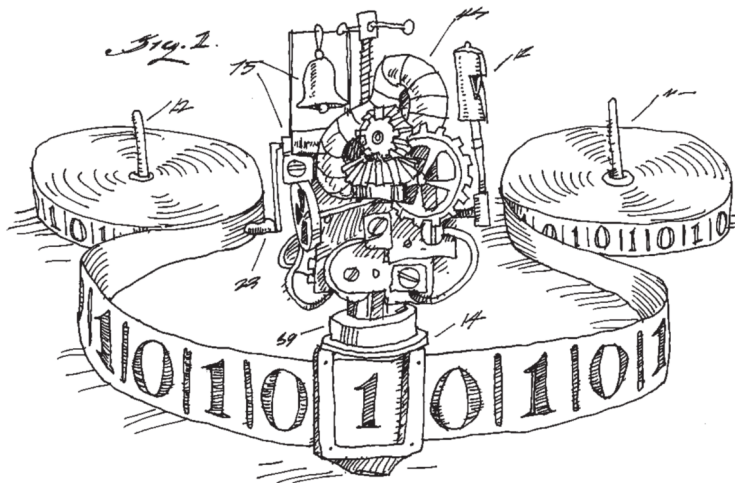
- Existe uma fórmula booleana \mathcal{F} , para cada problema $A \in \mathcal{NP}$ e entrada x , tal que: x é instância SIM para A **se e somente se** \mathcal{F} é instância SIM para **SAT**.

SAT é \mathcal{NP} -difícil.

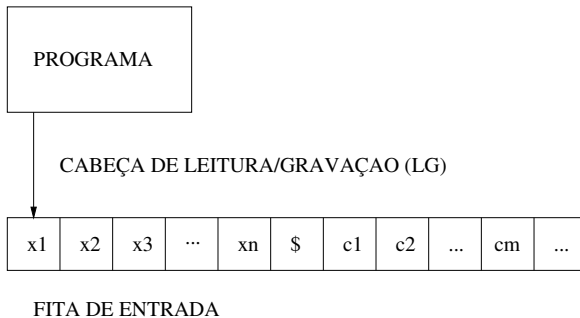
Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Classes \mathcal{P} , \mathcal{NP} , \mathcal{NP} -difícil, \mathcal{NP} -completo
- 3 Máquina de Turing**
- 4 Teorema de Cook
- 5 Síntese

Máquina de Turing



Máquina de Turing



Máquina de Turing

Foi definida por Alan Turing em 1937 como modelo universal de computação.

Máquina de Turing

No início da execução, a cabeça de LG encontra-se no primeiro símbolo da *string* de entrada (a posição mais à esquerda dos caracteres não-vazios da fita).
Cada **instrução** ou **estado** do programa (exceto a última) é da forma:

$$l : \text{ se } \sigma \text{ então } (\sigma', o, \ell'),$$

- l e ℓ' são números de instruções;
- σ e σ' são símbolos do alfabeto Σ da fita;
- $o \in \{-1, 0, 1\}$.

Máquina de Turing

No início da execução, a cabeça de LG encontra-se no primeiro símbolo da *string* de entrada (a posição mais à esquerda dos caracteres não-vazios da fita).
Cada **instrução** ou **estado** do programa (exceto a última) é da forma:

$$l : \text{ se } \sigma \text{ então } (\sigma', o, l'),$$

- l e l' são números de instruções;
- σ e σ' são símbolos do alfabeto Σ da fita;
- $o \in \{-1, 0, 1\}$.
- O significado da instrução anterior é:
“Se o símbolo lido na fita de entrada é σ , escreva σ' no seu lugar, mova a cabeça de LG o posições para direita e depois execute a instrução número l' .”

Máquina de Turing

A última instrução do programa é:

t : Aceitar.

A execução finaliza quando a Máquina de Turing alcança a última instrução.
Se a Máquina de Turing está em uma instrução não definida para o símbolo atual σ na fita, a execução trava.

Máquina de Turing

A última instrução do programa é:

t : Aceitar.

A execução finaliza quando a Máquina de Turing alcança a última instrução.

Se a Máquina de Turing está em uma instrução não definida para o símbolo atual σ na fita, a execução trava.

Curiosamente, no seu artigo original em 1937, Turing imaginou não um mecanismo, mas uma pessoa que executa essas regras determinísticas mecanicamente, a quem ele chamou de “computador”.

Máquina de Turing

Máquina de Turing para problemas de decisão:

- Uma cadeia de entrada é dita **aceita** por uma Máquina de Turing em tempo $p(|x|)$ se a última instrução é alcançada depois de no máximo $p(|x|)$ instruções executadas (em particular, $p(n)$ pode ser um polinômio).
- Se a última instrução não é alcançada nesse tempo, ou a execução trava, vamos considerar a cadeia como **rejeitada**.

Máquina de Turing

Máquina de Turing como algoritmo verificador de um problema \mathcal{NP} :

- **Lembrando:** O problema de decisão $A \in \mathcal{NP}$ quando A admite um algoritmo verificador de complexidade polinomial.

Máquina de Turing

Máquina de Turing como algoritmo verificador de um problema \mathcal{NP} :

- **Lembrando:** O problema de decisão $A \in \mathcal{NP}$ quando A admite um algoritmo verificador de complexidade polinomial.
- $A \in \mathcal{NP}$ se existe uma Máquina de Turing M e um polinômio $p(n)$, tais que:
 x é uma instância SIM para A **se e somente se** existe uma cadeia $c(x)$ (um certificado), com $|c(x)| \leq p(|x|)$ (conciso) tal que M **aceita** $x\$c(x)$ usando no máximo $p(|x|)$ instruções.

Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Classes \mathcal{P} , \mathcal{NP} , \mathcal{NP} -difícil, \mathcal{NP} -completo
- 3 Máquina de Turing
- 4 Teorema de Cook**
- 5 Síntese

Teorema de Cook

Teorema de Cook

SAT é \mathcal{NP} -completo.

Teorema de Cook

Teorema de Cook

SAT é \mathcal{NP} -completo.

Esboço da prova:

- **SAT** está em \mathcal{NP} (aula anterior). Precisamos mostrar que **SAT** é \mathcal{NP} -difícil.
- **Notação:** $A \in \mathcal{NP}$ é um problema de decisão, x é uma instância de entrada para A , $c(x)$ é um certificado para x , π é um algoritmo verificador para A (MT) contendo t instruções, $p(n)$ é um polinômio que indica a complexidade de π .
- Vamos mostrar que todo problema $A \in \mathcal{NP}$ se reduz polinomialmente a **SAT**.

Teorema de Cook: prova

Como $A \in \mathcal{NP}$, A possui um algoritmo (Máquina de Turing) que verifica, dada a instância x , e algum certificado $c(x)$, se o certificado é válido para x .

Para cada instância SIM x de A , precisamos definir uma fórmula \mathcal{F} SIM para **SAT**.

Podemos definir as **variáveis booleanas** a seguir:

- $z_{ij\sigma}$ para todo $0 \leq i, j \leq p(|x|)$ e todo $\sigma \in \Sigma$, onde $z_{ij\sigma} = 1$ se e somente se no instante i , a j -ésima posição da cadeia na fita de entrada contém o símbolo σ .
- $y_{ij\ell}$ para todo $0 \leq i \leq p(|x|)$, para todo $0 \leq j \leq p(|x|) + 1$ e todo $1 \leq \ell \leq t$, onde $y_{ij\ell} = 1$ se e somente se no instante i , a cabeça de LG está na j -ésima posição da cadeia na fita de entrada e a ℓ -ésima instrução do programa está sendo executada.

Teorema de Cook: prova

A partir da Máquina de Turing correspondente a π com uma entrada da forma $x\$c(x)$, onde x é a instância, vamos construir uma fórmula booleana \mathcal{F} com as variáveis anteriores da forma:

$$\mathcal{F}(z, y) = U(z, y) \cdot S(z, y) \cdot W(z, y) \cdot E(z, y)$$

Teorema de Cook: prova

- Se $U(z, y)$ for verdadeiro, estará garantido que a cada instante de tempo, cada posição da cadeia de entrada contém um único símbolo, que a cabeça de LG estará sobre uma única posição e que o programa executa uma única instrução.

$$\begin{aligned}
 U(z, y) = & \left(\prod_{\substack{0 \leq i, j \leq p(|x|) \\ \sigma \neq \sigma'}} (\bar{z}_{ij\sigma} + \bar{z}_{ij\sigma'}) \right) \cdot \left(\prod_{\substack{0 \leq i \leq p(|x|) \\ j \neq j' \text{ ou } \ell \neq \ell'}} (\bar{y}_{ij\ell} + \bar{y}_{ij'\ell'}) \right) \cdot \\
 & \left(\prod_{\substack{0 \leq i \leq p(|x|) \\ 1 \leq \ell \leq t}} (\bar{y}_{i0\ell} \cdot \bar{y}_{i, p(|x|)+1, \ell}) \right) \cdot \\
 & \left(\prod_{0 \leq i \leq p(|x|)} \left(\left(\prod_{1 \leq j \leq p(|x|)} \sum_{\sigma \in \Sigma} z_{ij\sigma} \right) \cdot \sum_{\substack{1 \leq j \leq p(|x|) \\ 1 \leq \ell \leq t}} y_{ij\ell} \right) \right)
 \end{aligned}$$

Teorema de Cook: prova

- Se $S(z, y)$ for verdadeiro, estará garantido que a Máquina de Turing está inicializada corretamente. Ou seja: os $|x| + 1$ símbolos mais à esquerda na fita correspondem à codificação de $x\$$, a cabeça de LG está na posição mais à esquerda da fita de entrada, e a primeira instrução do programa a ser executada será a instrução número 1.

$$S(z, y) = \left(\prod_{j=1}^{|x|} z_{0jx(j)} \right) \cdot z_{0,|x|+1,\$} \cdot y_{011}.$$

Teorema de Cook: prova

- Se $W(z, y)$ for verdadeiro, estará garantido que o algoritmo π realiza corretamente as instruções contidas no programa. Ou seja, ao executar a instrução:

ℓ : se σ então (σ', o, ℓ') ,

o símbolo que é gravado na posição corrente é σ' ou permanece inalterado, a cabeça de LG se movimenta para o posições à direita da posição corrente ou fica na mesma posição e a próxima instrução a ser executada é a instrução ℓ' ou a instrução $\ell + 1$. Além disso, deverá ser garantido que, se j é a posição corrente da cabeça de LG, no instante seguinte, todos os símbolos nas demais posições permanecem inalterados.

Teorema de Cook: prova

- $W(z, y)$ é uma **conjunção** de fórmulas $W_{ij\sigma\ell}$.

Para cada $0 \leq i \leq p(|x|)$, $1 \leq j \leq p(|x|)$, $\sigma \in \Sigma$, $1 \leq \ell < t$, considere a ℓ^{a} instrução dada por:

$\ell : \text{ se } \sigma \text{ então } (\sigma', o, \ell')$

- A fórmula $W_{ij\sigma\ell}$ para uma instrução intermediária é:

$$W_{ij\sigma\ell}(z, y) = (\bar{z}_{ij\sigma} + \bar{y}_{ij\ell} + z_{i+1,j,\sigma'}) \cdot (\bar{z}_{ij\sigma} + \bar{y}_{ij\ell} + y_{i+1,j,o,\ell'}) \cdot \prod_{\tau \neq \sigma} ((\bar{z}_{ij\tau} + \bar{y}_{ij\ell} + z_{i+1,j,\tau}) \cdot (\bar{z}_{ij\tau} + \bar{y}_{ij\ell} + y_{i+1,j,\ell+1}))$$

Teorema de Cook: prova

- A fórmula $W_{ij\sigma t}$ para a última instrução é:

$$W_{ij\sigma t}(z, y) = (\bar{z}_{ij\sigma} + \bar{y}_{ijt} + y_{i+1,j,t})$$

Nota: se o algoritmo atinge a instrução t , ele permanece lá.

- Resta garantir que se a cabeça de LG está numa posição diferente de j , esta permanecerá inalterada:

$$\prod_{\substack{0 \leq i \leq p(|x|) \\ \sigma \in \Sigma, j \neq j' \\ 1 \leq \ell \leq t}} (\bar{z}_{ij\sigma} + \bar{y}_{ij'\ell} + z_{i+1,j,\sigma})$$

Teorema de Cook: prova

- Se $E(z, y)$ é verdadeiro, estará garantido que t é a última instrução executada pelo algoritmo π . Ou seja:

$$E(z, y) = \sum_{j=1}^{p(|x|)} y_{p(|x|),j,t}$$

Complexidade da construção de \mathcal{F} : $O(p^3(|x|) \log p(|x|))$.

Teorema de Cook: prova

$\mathcal{F}(z, y)$ tem resposta SIM para **SAT** se e somente se x tem resposta SIM para A .

Teorema de Cook: prova

$\mathcal{F}(z, y)$ tem resposta SIM para **SAT** se e somente se x tem resposta SIM para A .

- $\mathcal{F}(z, y)$ tem resposta SIM para **SAT**
 - \Rightarrow existe uma atribuição de variáveis $z_{ij\sigma}$ e $y_{ij\ell}$ que fazem $\mathcal{F}(z, y)$ ser verdadeira
 - \Rightarrow existe uma entrada $x\$c(x)$ que faz a Máquina de Turing finalizar com Aceitar
 - \Rightarrow existe um certificado $c(x)$ válido
 - $\Rightarrow x$ tem resposta SIM para A .

Teorema de Cook: prova

$\mathcal{F}(z, y)$ tem resposta SIM para **SAT** se e somente se x tem resposta SIM para A .

- $\mathcal{F}(z, y)$ tem resposta SIM para **SAT**
 - \Rightarrow existe uma atribuição de variáveis $z_{ij\sigma}$ e $y_{ij\ell}$ que fazem $\mathcal{F}(z, y)$ ser verdadeira
 - \Rightarrow existe uma entrada $x\$c(x)$ que faz a Máquina de Turing finalizar com Aceitar
 - \Rightarrow existe um certificado $c(x)$ válido
 - $\Rightarrow x$ tem resposta SIM para A .
- x tem resposta SIM para A
 - \Rightarrow existe um certificado $c(x)$ válido
 - \Rightarrow existe uma execução da Máquina de Turing que leva à instrução Aceitar
 - \Rightarrow existe uma atribuição de variáveis $z_{ij\sigma}$ e $y_{ij\ell}$ que fazem $\mathcal{F}(z, y)$ ser verdadeira
 - $\Rightarrow \mathcal{F}(z, y)$ tem resposta SIM para **SAT**

Teorema de Cook

Como utilizaríamos SAT para resolver um problema A ?

Temos um problema $A \in \mathcal{NP}$. Então, há uma Máquina de Turing (MT) que verifica corretamente, dada a entrada x e o certificado $c(x)$, se este certificado é válido.

Existe solução para a instância x (um certificado $c(x)$ que faça o algoritmo verificador, a Máquina de Turing MT, retornar SIM)?

- Posso construir uma fórmula \mathcal{F} a partir do algoritmo MT e da entrada x .
- Posso usar a fórmula \mathcal{F} para determinar se existe alguma **solução** (algum certificado $c(x)$) que resolve A .

Teorema de Cook: consequências

- **SAT** é \mathcal{NP} -completo (e \mathcal{NP} -difícil).
- Todo problema em \mathcal{NP} pode ser reduzido em tempo polinomial a uma instância do problema **SAT**.
- Como **SAT** é um problema \mathcal{NP} -completo, se existe uma redução polinomial de **SAT** para $A \in \mathcal{NP}$, então A é \mathcal{NP} -completo.

Teorema de Cook: consequências

- **SAT** é \mathcal{NP} -completo (e \mathcal{NP} -difícil).
- Todo problema em \mathcal{NP} pode ser reduzido em tempo polinomial a uma instância do problema **SAT**.
- Como **SAT** é um problema \mathcal{NP} -completo, se existe uma redução polinomial de **SAT** para $A \in \mathcal{NP}$, então A é \mathcal{NP} -completo.
- A Questão Fundamental da Computação, $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$, agora é equivalente à questão de saber se existe um algoritmo determinístico polinomial para **SAT**, ou para **qualquer um** dos problemas \mathcal{NP} -completos.

Combinatorial Optimization: Algorithms and Complexity C.H. Papadimitriou e K. Steiglitz, 1998.

The Design and Analysis of Computer Algorithms. A.V. Aho, J.E. Hopcroft e J.D. Ullman, 1974.

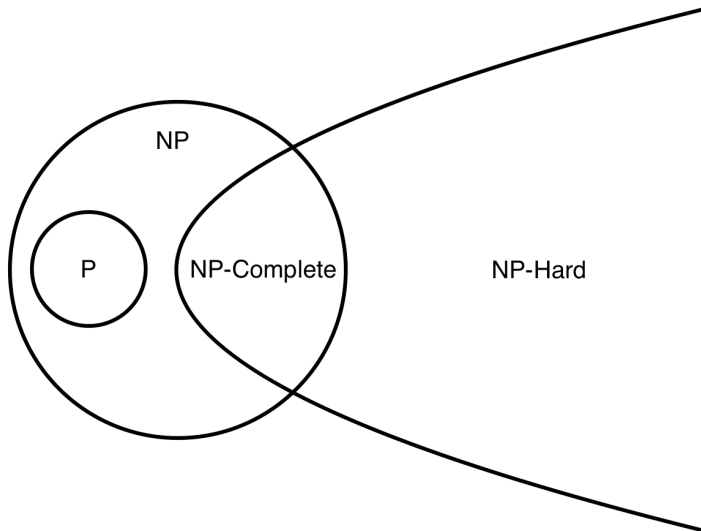
Provas de \mathcal{NP} -completude

- Depois que Cook provou que **SAT** é \mathcal{NP} -completo, Richard Karp (1972) mostrou que outros 24 problemas famosos também são \mathcal{NP} -completos.

Provas de \mathcal{NP} -completude

- Depois que Cook provou que **SAT** é \mathcal{NP} -completo, Richard Karp (1972) mostrou que outros 24 problemas famosos também são \mathcal{NP} -completos.
- **Lembre-se!** Para provar que um problema A é \mathcal{NP} -completo é necessário:
 - 1 Provar que A é \mathcal{NP} ;
 - 2 Provar que A é \mathcal{NP} -difícil. Isso geralmente é feito encontrando uma redução polinomial de algum problema \mathcal{NP} -difícil (quase sempre \mathcal{NP} -completo) para A .

Classes de Problemas: caso $\mathcal{P} \neq \mathcal{NP}$



Resumo

- 1 Revisão do conteúdo e objetivo
- 2 Classes \mathcal{P} , \mathcal{NP} , \mathcal{NP} -difícil, \mathcal{NP} -completo
- 3 Máquina de Turing
- 4 Teorema de Cook
- 5 Síntese

Síntese

- Um problema A é \mathcal{NP} -difícil se todo problema de \mathcal{NP} se reduz polinomialmente a A .
- Um problema A é \mathcal{NP} -completo se $A \in \mathcal{NP}$ e $A \in \mathcal{NP}$ -difícil.
- **SAT** é \mathcal{NP} -completo.
- Para provar que outro problema A é \mathcal{NP} -completo é necessário:
 - Provar que A é \mathcal{NP} , e
 - Encontrar uma redução polinomial de um problema \mathcal{NP} -completo para A .

Material bibliográfico e exercícios

U. Manber. Introduction to Algorithms. – **Cap. 11.**

Exercícios: ver exercícios no final do Capítulo 11.

Bibliografia complementar deste tópico:

Combinatorial Optimization: Algorithms and Complexity

C.H. Papadimitriou e K. Steiglitz, Dover, 1982.

The Design and Analysis of Computer Algorithms (cap. 1)

A.V. Aho, J.E. Hopcroft e J.D. Ullman, Addison-Wesley, 1974.

Dúvidas

Dúvidas?