Aula 06 – Classes de Complexidade

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação

Introdução

- Complexidade: estudo dos recursos (tempo e espaço) necessários para a computação de uma TM.
- Problemas tratáveis: resolvíveis por um algoritmo eficiente.
- Problemas intratáveis: possuem uma solução teórica, mas esta consome recursos demais.
- **Estes *slides***: Caracterizar a separação dos problemas.
- **Objetivos**: Introduzir as classes de complexidade $\mathcal{P}$, $\mathcal{NP}$ e seus problemas fundamentais.

Referências

Chapter 15 – P, NP, and Cook's Theorem
T. Sudkamp
Chapter 7 – Time Complexity
M. Sipser
Chapter 6 – Complexity Theory
A. Maheshwari

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

Definição 15.2.1 (Sudkamp)

Uma linguagem L é decidível em tempo polinomial (determinístico) se existe uma DTM padrão M que decide L com $tc_M(n) \in O(n^r)$, aonde r é um natural independente de n.

- A classe (conjunto) de linguagens decidíveis em tempo polinomial (determinístico) é denotada por $\mathcal{P}$.
- Outros tipos de DTM poderiam ter sido usados na definição pois preservam as soluções polinomiais.
- Uma linguagem aceita por uma DTM multi-faixa em $O(n^r)$ também é aceita por uma DTM padrão em $O(n^r)$.
- Uma linguagem aceita por uma DTM multi-fita em $O(n^r)$ é aceita por uma DTM padrão em $O(n^{2r})$.
- A robustez da classe $\mathcal{P}$ sob diferentes variação de DTMs justifica a sua escolha como fronteira entre problemas tratáveis e intratáveis.

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

Definição 15.2.2 (Sudkamp)

Uma linguagem L é decidível em tempo polinomial **não-determinístico** se existe uma NTM M que decide L com $tc_M(n) \in O(n^r)$, aonde r é um natural independente de n.

- A classe (conjunto) de linguagens decidíveis em tempo polinomial não-determinístico é denotada por $\mathcal{NP}$.
- A classe $\mathcal{NP}$ é um subconjunto das linguagens recursivas.
- O limite sobre o número de transições garante que todas as possíveis computações de M terminam.
- Como toda DTM também é uma NTM, temos que $\mathcal{P} \subseteq \mathcal{NP}$.
- $\mathcal{NP} \subseteq \mathcal{P}$? $\Rightarrow$ Pergunta mais importante ainda em aberto da computação teórica.

Problemas de Decisão e Classes de Complexidade

Aceite de Palíndromos

Input: string u sobre alfabeto Σ.

Output: sim; se u é um palíndromo
não; caso contrário.

Complexidade: em $\mathcal{P}$? Sim.

- Palíndromos são aceitos em $O(n^2)$ por uma DTM padrão, aonde $n = \mathit{length}(u)$.
- Portanto o problema está na classe $\mathcal{P}$.

Problemas de Decisão e Classes de Complexidade

Caminho em Grafo Direcionado

Input: grafo direcionado $G = (\mathrm{N}, \mathrm{A})$, nós $v_i, v_j \in \mathrm{N}$.

Output: sim; se existe um caminho de v_i para v_j em G
não; caso contrário.

Complexidade: em $\mathcal{P}$? Sim.

- O algoritmo de Dijkstra pode ser usado para descobrir se há um caminho entre dois nós em um grafo direcionado.
- A complexidade de tempo desse algoritmo está em $O(n^2)$, aonde $n = card(\mathrm{N})$.
- Portanto o problema está na classe $\mathcal{P}$.

Problemas de Decisão e Classes de Complexidade

Satisfatibilidade

Input: fórmula Booleana u na forma normal conjuntiva (CNF).

Output: sim; se existe uma atribuição de valores que satisfaz u
não; caso contrário.

Complexidade: em $\mathcal{P}$? Desconhecido.
em $\mathcal{NP}$? Sim.

- Uma NTM para o problema "advinha" uma atribuição de valores e testa se u é verdadeira.
- Esse teste pode ser feito em tempo polinomial (quadrático) em relação ao número de variáveis em u.

Problemas de Decisão e Classes de Complexidade

Problema do Circuito Hamiltoniano

Input: grafo direcionado $G = (N, A)$.

Output: sim; se existe um ciclo que visita todos os vértices exatamente uma vez
não; caso contrário.

Complexidade: em $\mathcal{P}$? Desconhecido.
em $\mathcal{NP}$? Sim.

- Uma NTM para o problema "advinha" uma sequência de $n+1$ vértices e testa se é um ciclo que satisfaz a condição.
- Esse teste pode ser feito em tempo polinomial (linear) em relação ao tamanho da sequência.

Problemas de Decisão e Classes de Complexidade

Problema da Soma do Subconjunto

Input: conjunto S, função $v : S \to \mathbf{N}$, número k.

Output: sim; se existe um subconjunto S' de S cujo valor total é k
não; caso contrário.

Complexidade: em $\mathcal{P}$? Desconhecido.
em $\mathcal{NP}$? Sim.

- Uma NTM para o problema "advinha" um subconjunto S' e testa se a soma dos valores dos elementos de S' é k.
- Esse teste pode ser feito em tempo polinomial (linear) em relação ao tamanho do subconjunto S'.

Problemas de Decisão e Classes de Complexidade

- Para problemas cuja pertinência em $\mathcal{P}$ é desconhecida, uma solução determinística em geral degenera para uma busca exaustiva.
- Métodos força-bruta em geral são ineficientes (complexidade exponencial em DTM).
- Por isso, um problema cuja pertinência em $\mathcal{P}$ é desconhecida é considerado intratável (Tese de Cobham, 1965).

Problemas de Decisão e Classes de Complexidade

- Como $\mathcal{NP}$ é um subconjunto (próprio) das linguagens recursivas, é de se esperar que existam linguagens cuja complexidade esteja além de $\mathcal{NP}$.
- Esse é o caso para o problema abaixo, cuja solução requer tempo e espaço exponenciais em relação ao tamanho da expressão regular α.

Expressões Regulares com Quadrados

Input: uma expressão regular α sobre um alfabeto Σ.

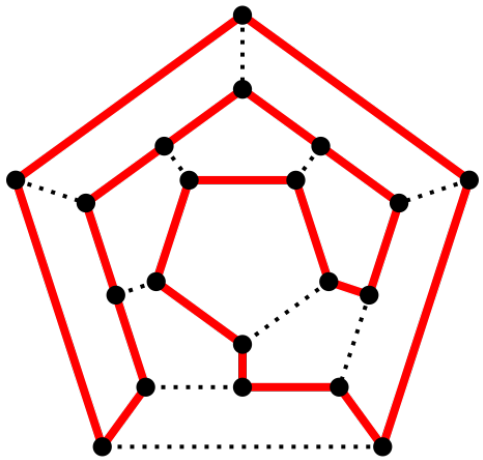
Output: sim; se $\alpha \neq \Sigma^*$
não; caso contrário.

Complexidade: em $\mathcal{P}$? Não.
em $\mathcal{NP}$? Não.

Nota: as operações consideradas na ERs para esse problema são as usuais, além de $u^2 = uu$.

Problema do Circuito Hamiltoniano

- O Problema do Circuito Hamiltoniano – *Hamiltonian Circuit Problem* (HCP) foi proposto por William Hamilton ($\sim$1860).
- Dado um grafo (direcionado ou não), achar um caminho que percorre todos os vértices uma única vez e retorna à origem.



- Vamos tratar do problema de decisão em grafos direcionados.

Problema do Circuito Hamiltoniano

- Seja G um grafo direcionado com n vértices numerados de 1 a n.
- Um Circuito Hamiltoniano (HC) é um caminho $v_0, v_1, \ldots, v_n$ em G que satisfaz:
  1. $v_0 = v_n$
  2. $v_i \neq v_j$ sempre que $i \neq j$ e $0 \leq i, j < n$.
- Isto é, um HC é um caminho que visita cada vértice exatamente uma vez e termina no ponto inicial.
- Um HC também é chamado de tour.
- O HCP é o problema de determinar se um grafo direcionado tem um tour.
- Como um tour contém todos os vértices, assumimos que ele sempre começa e termina no vértice 1.

Problema do Circuito Hamiltoniano

- A solução por DTM para o HCP é uma busca exaustiva pelas sequências de vértices para determinar se uma sequência é um tour.
- As sequências são sistematicamente geradas e testadas até que um tour seja encontrado ou todas as possibilidades sejam examinadas.
- Isso pode ser feito por uma DTM 4-fitas.
  - Fita 1: contém a entrada, a codificação do grafo G.
  - Fita 2: é usada para gerar a sequência de vértices a ser testada.
  - Fita 3: é usada para testar a sequência da fita 2.
  - Fita 4: contém a última sequência, que é a condição de parada.

Problema do Circuito Hamiltoniano

- A performance de pior caso da DTM ocorre quando o grafo não possui nenhum tour.
- Nesse caso, são geradas e testadas n^{n-1} sequências de vértices.
- Portanto, HCP é resolvível por DTM em tempo exponencial.
- Mas não podemos concluir que o problema não pode ser resolvido por DTM em tempo polinomial.
- Só sabemos que até agora nenhum algoritmo polinomial foi descoberto.
- Pode ser que nenhuma solução de fato exista ou só que ninguém foi esperto o suficiente até agora para encontrar uma solução eficiente.

Problema do Circuito Hamiltoniano

- É possível construir uma NTM de 3-fitas que resolve o HCP.
- Basta adaptar a DTM anterior, descartando a fita 4.
- A computação da NTM é dada a seguir.
  1. Para e rejeita a entrada se o grafo tem menos de $n+1$ arcos.
  2. Gera de forma não-determinística uma sequência de $n+1$ vértices na fita 2.
  3. Usa as fitas 1 e 3 para testar se a sequência na fita 2 é um tour.
- A complexidade de tempo da computação da máquina acima é da ordem de $O(k^2 \log_2(k))$ aonde k é o número de arcos do grafo.
- Portanto HCP $\in \mathcal{NP}$.

Redução de Problemas

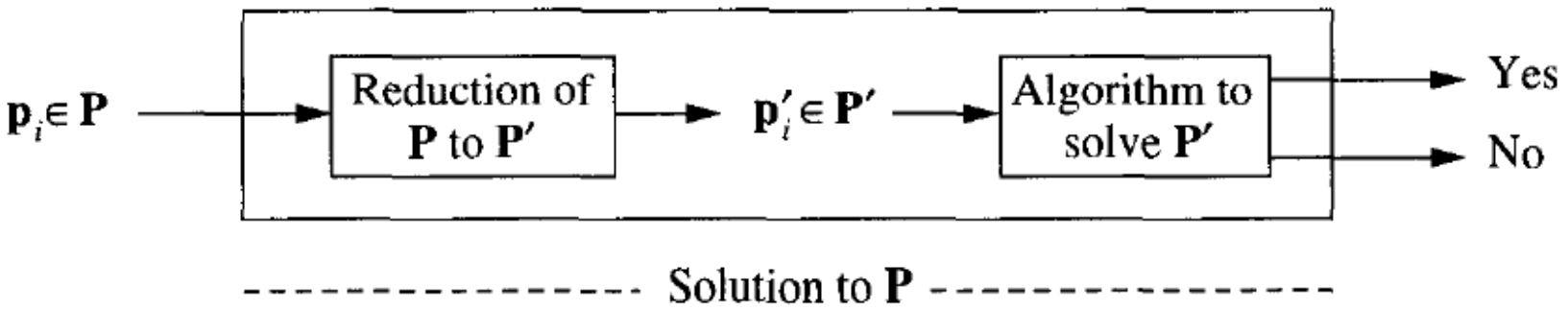
- Redução é uma técnica para solução de problemas usada para evitar “reinventar a roda”.
- Objetivo da redução: transformar instâncias de um novo problema em instâncias de um problema já resolvido.
- Essencialmente, fazer uma redução requer a construção de uma função computável que mapeia instâncias.
- Redução é uma ferramenta importante para estabelecer a decidibilidade de problemas.
- Em particular, é usada para mostrar que certos problemas são indecidíveis.
- Além disso, redução pode ser usada para classificar um problema como tratável ou intratável.

Redução de Problemas

Definição – Redução de Problemas

Um problema de decisão **P** é redutível em muitos-para-um para um problema **P′** se existe uma TM M aonde:

- M recebe como entrada qualquer instância $p_i \in \mathbf{P}$ e produz uma instância associada $p'_i \in \mathbf{P'}$; e
- a resposta para a instância original p_i pode ser obtida a partir da resposta para p'_i.

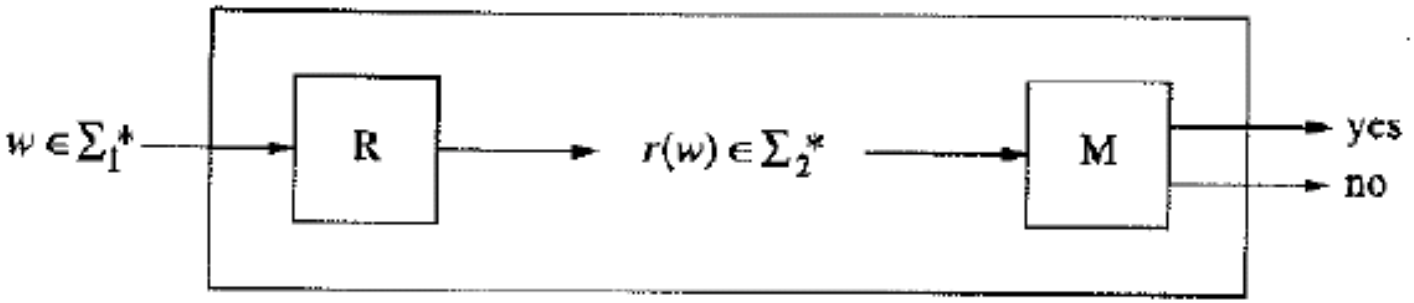


Mapeamento por M não precisa ser injetivo: várias instâncias de **P** podem ser mapeadas na mesma instância de **P′**.

Redução de Problemas

Definição – Redução de Linguagens

Seja L uma linguagem sobre Σ_1 e L′ uma linguagem sobre Σ_2. L é redutível em muitos-para-um para L′ se existe uma função computável $r : \Sigma_1^* \to \Sigma_2^*$ tal que $w \in L$ sss $r(w) \in L'$.



- Importante notar que R não determina a pertinência nem em L nem em L′. R só transforma strings de Σ_1^* para Σ_2^*.
- Pertinência em L′ é determinada por M, e em L pela combinação de R e M.

Redução de Problemas – Exemplo

- A linguagem $\mathrm{L} = \{x^i y^i z^k \mid i \geq 0, k \geq 0\}$ é redutível para $\mathrm{L}' = \{a^i b^i \mid i \geq 0\}$.
- Uma redução de L para L′ pode ser descrita como abaixo.

Redução	**Entrada**	**Condição**
$\mathrm{L} = \{x^i y^i z^k \mid i \geq 0, k \geq 0\}$ para $\mathrm{L}' = \{a^i b^i \mid i \geq 0\}$	$w \in \{x, y, z\}^*$ $\downarrow r$ $v \in \{a, b\}^*$	$w \in \mathrm{L}$ se e somente se $r(w) \in \mathrm{L}'$

Redução de Problemas – Exemplo

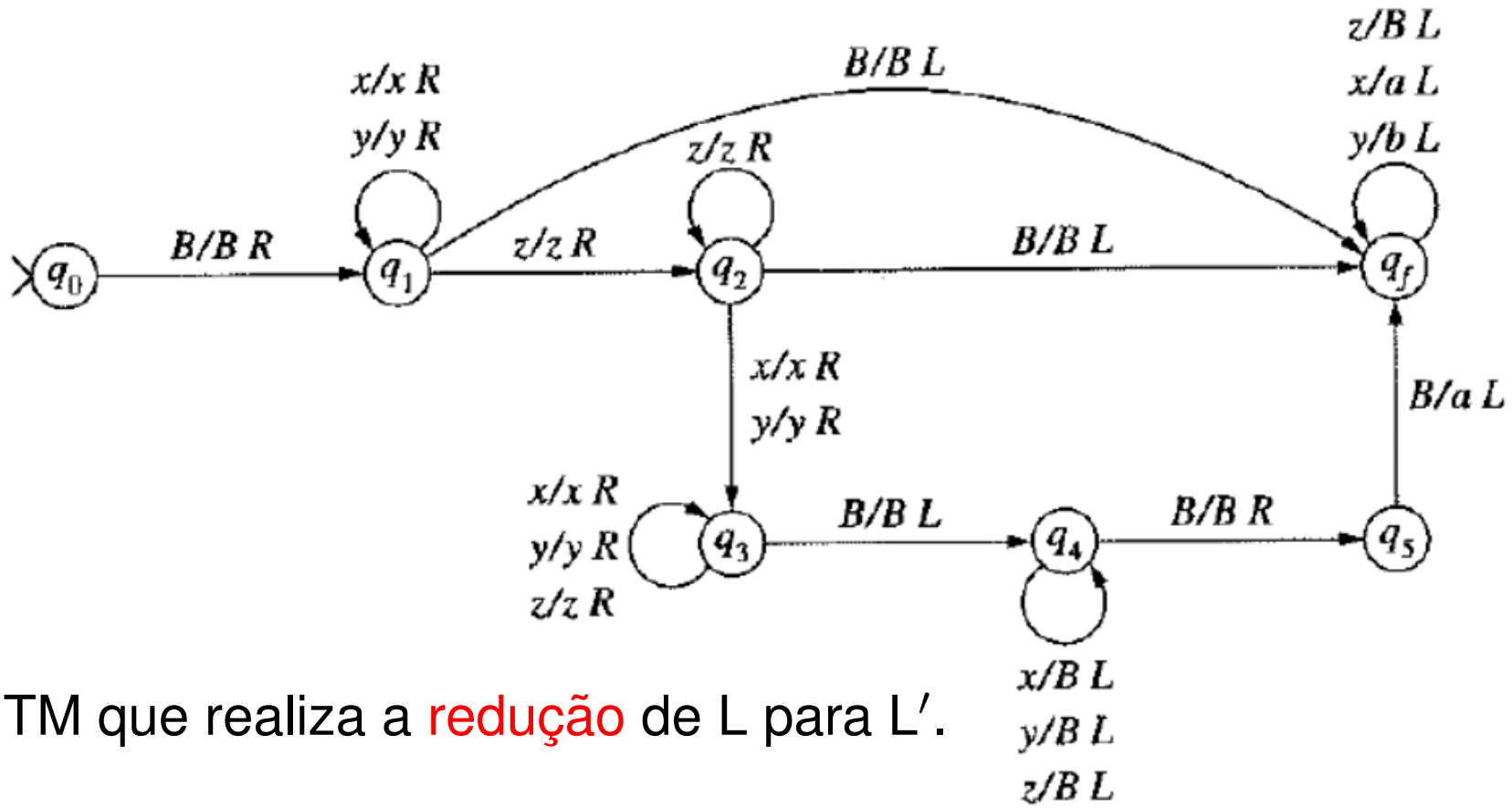
Uma string $w \in \{x, y, z\}^*$ é transformada na string $r(w) \in \{a, b\}^*$ como a seguir:

1. Se w não tem x ou y ocorrendo depois de um z, troque cada x por um a, cada y por um b e apague todos os z's.
2. Se w tem x ou y ocorrendo depois de um z, apague a string toda e escreva um único a na fita.

$w \in \Sigma_1^*$	Em L?	$r(w) \in \Sigma_2^*$	Em L′?
$xxyy$	sim	$aabb$	sim
$xxyyzzz$	sim	$aabb$	sim
$yxxyz$	não	$baab$	não
$xxzyy$	não	a	não
$zyzx$	não	a	não
λ	sim	λ	sim
zzz	sim	λ	sim

Redução de Problemas – Exemplo

Exemplos da tabela do slide anterior mostram porque a transformação é dita muitos-para-um: múltiplas strings de Σ_1^* podem ser mapeadas na mesma string em Σ_2^*.



TM que realiza a redução de L para L′.

Redução em Tempo Polinomial

- Seja r a redução de L para L′, computada pela máquina R.
- Se L′ é decidida pela máquina M então L é decidida por uma máquina que:
  1. executa R sobre a string de entrada w; e
  2. executa M sobre $r(w)$.
- A string $r(w)$ é aceita por M sss $w \in$ L.
- A complexidade de tempo da solução composta deve considerar o tempo gasto tanto por R quanto por M.
- Problemas com solução eficiente = complexidade de tempo polinomial.
- $\Rightarrow$ Restrição de tempo sobre a redução.

Redução em Tempo Polinomial

Definição 15.6.1 (Sudkamp)

Sejam L e L′ linguagens sobre os alfabetos Σ_1 e Σ_2, respectivamente. Diz-se que L é redutível em tempo polinomial para L′ se existe uma função $r : \Sigma_1^* \to \Sigma_2^*$, computável em tempo polinomial (determinístico), tal que $w \in L$ sss $r(w) \in L'$.

- Reduções em tempo polinomial são importantes porque o limite no número de transições de R restringe o tamanho da string $r(w)$ que entra em M.
- Isto garante que a combinação de uma redução polinomial R com uma máquina polinomial M produz outra máquina polinomial R;M.
- Isso é formalizado no teorema a seguir.

Redução em Tempo Polinomial

Teorema 15.6.2 (Sudkamp)

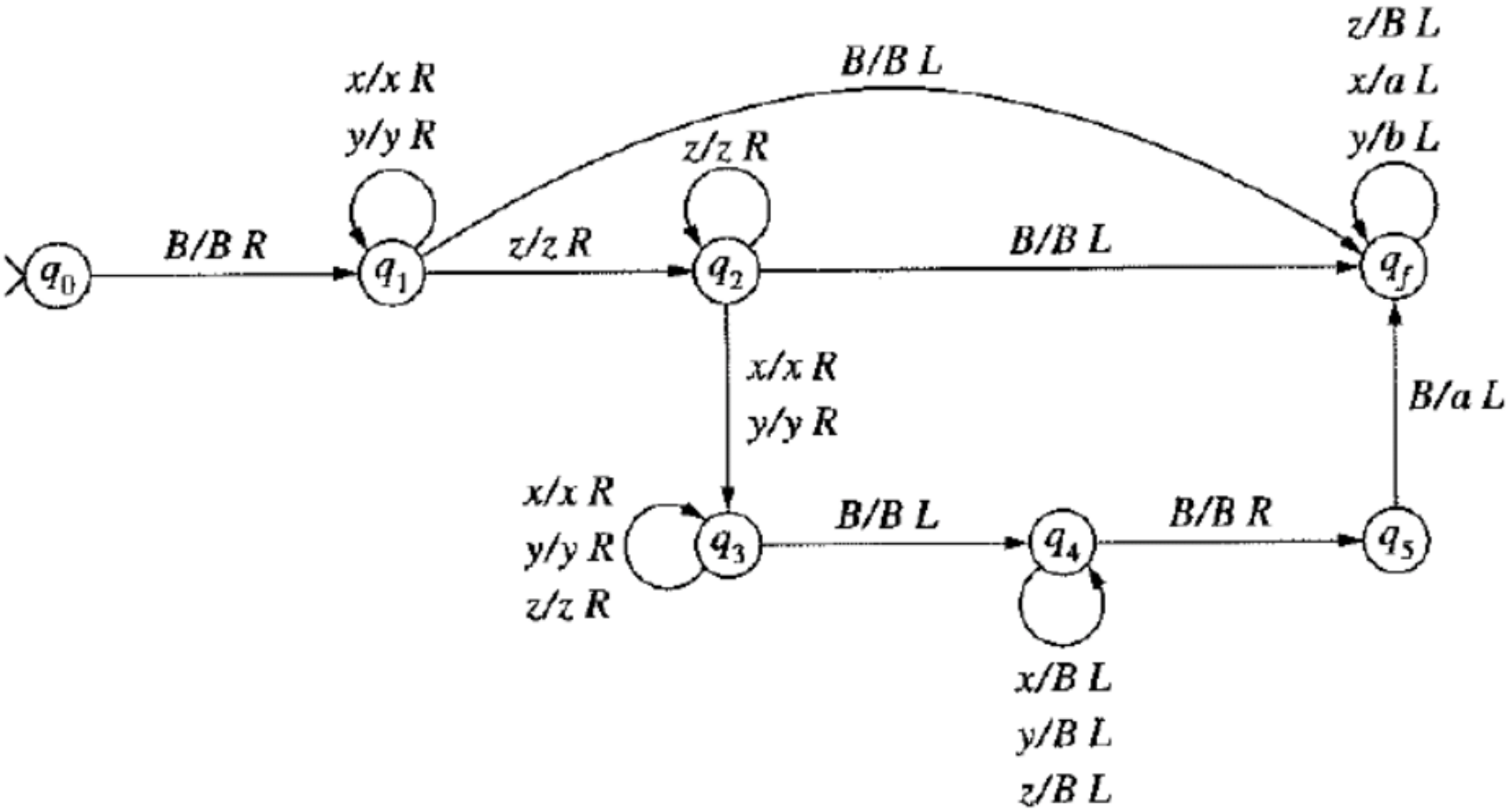
Se L é redutível a L′ em tempo polinomial e $L' \in \mathcal{P}$, então $L \in \mathcal{P}$.

- As complexidades de tempo tc_R e tc_M combinam para produzir um limite superior para a complexidade da máquina R;M.
- Se a string w para R tem tamanho n, então o tamanho de $r(w)$ não tem como exceder $tc_R(n) \in O(n^s)$.
- Seja $k = length(r(w))$. A computação de M realiza no máximo $tc_M(k) \in O(k^t)$ transições.
- Considerando pior caso aonde $k = n^s$, a computação de R;M tem complexidade de tempo

$$tc_{R;M}(n) \in O(n^{st}).$$

Redução em Tempo Polinomial

Relembrando: a máquina R abaixo realiza a redução de $L = \{x^i y^i z^k \mid i \geq 0, k \geq 0\}$ para $L' = \{a^i b^i \mid i \geq 0\}$.



Redução em Tempo Polinomial

- O pior caso para a computação de R ocorre quando um símbolo x ou y aparece depois de um z.
- Neste caso, a máquina anda até o final da string e depois retorna apagando todos os símbolos.
- No total, são feitas $2(n+2)$ transições e portanto

$$tc_{\mathrm{R}}(n) = 2n + 4.$$

Redução em Tempo Polinomial

- Uma máquina M pode ser construída para decidir L′ com complexidade de tempo

$$tc_{\mathrm{M}}(n) \in O(n^2).$$

- Construção similar à DTM que aceita palíndromos.
- Com isso, a solução para o problema de pertinência em L tem complexidade de tempo

$$tc_{\mathrm{R;M}}(n) = 2n + 4 + O(n^2) \in O(n^2).$$

- Resultado acima corresponde ao esperado na prova do Teorema 15.6.2.

Redução em Tempo Polinomial

- Redução de problemas provê uma base para a comparação da dificuldade relativa de dois problemas.
- Vamos considerar que dois problemas têm dificuldade equivalente se a complexidade de tempo das soluções diferem apenas por um termo polinomial.
- Ênfase aqui é distinguir entre problemas tratáveis e intratáveis.
- Nesse aspecto, diferenças polinomiais na complexidade dos algoritmos não são significantes.
- Claro que do ponto de vista prático um algoritmo $O(n^2)$ é preferível a um $O(n^3)$, por exemplo.

Redução em Tempo Polinomial

- Se L é redutível a L′ em tempo polinomial, então L′ pode ser vista como sendo um problema tão difícil quanto L.
- Uma solução para L′ automaticamente provê uma solução para L.
- Além disso, se L′ é tratável então L também é.
- A relação entre reduções e a dificuldade relativa das linguagens pode ser estendida para classes (conjuntos) de linguagens.

Redução em Tempo Polinomial

Definição 15.6.3 (Sudkamp)

Seja $\mathcal{C}$ uma classe (conjunto) de linguagens. Uma linguagem L′ é difícil para a classe $\mathcal{C}$ se toda linguagem em $\mathcal{C}$ é redutível a L′ em tempo polinomial.

Se L′ é difícil para a classe $\mathcal{C}$ e L′ é decidível em tempo polinomial, então todas as linguagens em $\mathcal{C}$ são decidíveis em tempo polinomial, e $\mathcal{C} \subseteq \mathcal{P}$.

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

- Uma linguagem decidível em tempo polinomial determinístico (i.e., aceita por uma DTM) está em $\mathcal{P}$.
- Uma linguagem decidível em tempo polinomial não-determinístico (i.e., aceita por uma NTM) está em $\mathcal{NP}$.
- A construção de uma DTM a partir de uma NTM causa um crescimento exponencial na complexidade de tempo.
- Diferença fundamental das TMs para o HCP:
  - DTM: gera todas as sequências de vértices e testa.
  - NTM: "advinha" uma sequência de vértices e testa.
- Interpretação intuitiva da pergunta $\mathcal{P} = \mathcal{NP}$?:
  - Construir uma solução para um problema é inerentemente mais difícil do que verificar se uma única possibilidade satisfaz as condições do problema?
- Considerando que a resposta parece ser sim para um grande número de problemas, acredita-se que $\mathcal{P} \neq \mathcal{NP}$.

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

- Suponha que alguém descubra uma solução polinomial determinística para o HCP.
- Se isso acontecesse, então poderíamos concluir que HCP $\in \mathcal{P}$.
- Isso seria suficiente para responder a pergunta $\mathcal{P} = \mathcal{NP}$?
- Não. Descobriu-se apenas uma solução eficiente para um problema que antes era intratável.
- E os demais problemas em $\mathcal{NP}$? Essa classe possui vários problemas.
- $\Rightarrow$ É inviável tentar descobrir uma solução tratável para cada um dos problemas em $\mathcal{NP}$ individualmente.

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

- É necessário um método que resolve a questão da existência de uma solução eficiente para todas as linguagens em $\mathcal{NP}$ de uma única vez.
- A noção de uma linguagem ser difícil para a classe $\mathcal{NP}$ permite transformar a pergunta sobre a existência de uma solução eficiente para todos os problemas em $\mathcal{NP}$ em uma pergunta de um único problema.

Definição 15.7.1 (Sudkamp)

- Uma linguagem L′ é dita NP-hard se para toda L $\in \mathcal{NP}$, L é redutível à L′ em tempo polinomial.
- Uma linguagem NP-hard que também está em $\mathcal{NP}$ é dita NP-complete.

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

- Podemos considerar uma linguagem NP-complete como uma linguagem universal da classe $\mathcal{NP}$.
- A descoberta de uma DTM que decide uma linguagem NP-complete em tempo polinomial, levaria à decisão de todas as linguagens em $\mathcal{NP}$ em tempo polinomial determinístico.
- Se isso acontecer, teremos que $\mathcal{P} = \mathcal{NP}$.

Teorema 15.7.2 (Sudkamp)

Se existe uma linguagem NP-complete que está em $\mathcal{P}$, então $\mathcal{P} = \mathcal{NP}$.

- Suponha L′ uma linguagem NP-complete aceita por DTM em tempo polinomial (L′ $\in \mathcal{P}$).
- L′ é NP-hard (pré-condição para ser NP-complete) e portanto para qualquer L $\in \mathcal{NP}$, existe uma redução em tempo polinomial de L para L′.
- Como L′ $\in \mathcal{P}$, pelo Teorema 15.6.2 temos que L $\in \mathcal{P}$, para qualquer L $\in \mathcal{NP}$, e portanto, $\mathcal{P} = \mathcal{NP}$.

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

- A classe de linguagens (problemas de decisão) NP-complete é denotada por $\mathcal{NPC}$.
- A classe $\mathcal{NPC}$ é claramente uma classe importante de problemas.
- Mas qual seria um problema universal que está em $\mathcal{NPC}$?
- $\Rightarrow$ SAT.

Problema da Satisfatibilidade (SAT)

- SAT foi o primeiro problema NP-complete.
- Provado por Cook em 1971.
- SAT recebe como entrada uma fórmula em lógica proposicional em CNF.
- CNF (*Conjuntive Normal Form*): uma fórmula que tem a forma

$$u_1 \wedge u_2 \wedge \cdots \wedge u_n$$

  aonde cada cláusula u_i é uma disjunção ($\vee$) de literais.
- Um literal é uma proposição atômica ou sua negação.
- SAT é um problema de decisão que busca determinar se existe um conjunto de valores Booleanos (atribuição) para as proposições que torna a fórmula verdadeira.
- Se a resposta for sim então a fórmula é satisfatível.

Problema da Satisfatibilidade (SAT)

- Uma solução determinística para SAT pode gerar e analisar cada possível atribuição de valores para as variáveis Booleanas (proposições).
- O número de atribuições possíveis é 2^n, onde n é o número de variáveis.
- Claramente, a complexidade dessa solução é exponencial.
- Podemos então afirmar que SAT $\notin \mathcal{P}$?
- Não. Só podemos dizer que o método força bruta não é tratável.
- Com o exposto acima, podemos dizer que SAT $\in \mathcal{NP}$?
- Não. Isso requer mostrar que existe uma solução por NTM com complexidade de tempo polinomial.

Problema da Satisfatibilidade (SAT)

- O trabalho de gerar todas as atribuições é exponencial.
- Por outro lado, verificar se uma dada atribuição torna a fórmula satisfatível tem uma complexidade que cresce polinomialmente com relação ao número de variáveis e o comprimento da fórmula.
- Essa observação é o suficiente para se projetar uma NTM que resolve SAT em tempo polinomial.

Teorema 15.8.2 (Sudkamp)

O Problema da Satisfatibilidade está em $\mathcal{NP}$.

Problema da Satisfatibilidade (SAT)

- Representação de fórmulas como strings de entrada para a NTM?
- Seja u uma fórmula com variáveis $x_1, \ldots, x_n$.
- Uma variável é codificada pela representação binária do seu subscrito.
- A codificação de um literal consiste na codificação da variável seguida de $\#1$ se o literal é positivo, ou de $\#0$ se é negativo.
- O número seguindo a codificação da variável especifica o valor Booleano que satisfaz o literal.

Problema da Satisfatibilidade (SAT)

- Exemplo: a fórmula CNF

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

é codificada como

$$1\#1 \vee 10\#0 \wedge 1\#0 \vee 11\#1 \quad .$$

- A representação de uma instância de SAT é uma string sobre o alfabeto $\Sigma = \{0, 1, \wedge, \vee, \#\}$.
- A linguagem L_{SAT} é formada por todas as strings sobre Σ que representam fórmulas CNF satisfatíveis.

Problema da Satisfatibilidade (SAT)

- Construímos uma NTM M de 2-fitas para resolver SAT.
- Construção segue o método padrão de NTM: *guess-and-check*.
- Inicialmente a fita 1 contém a fórmula a ser analisada e a fita 2 está vazia.
- Primeiramente, M testa a string de entrada e rejeita se não estiver no formato esperado. Isso pode ser feito com complexidade linear.
- A seguir, M "advinha" não-deterministicamente uma atribuição de valores para as variáveis da fórmula. Também requer complexidade linear.
- M verifica se a atribuição torna a fórmula verdadeira. Isso pode ser realizado em tempo $O(k \cdot n^2)$, com n o número de variáveis e k o comprimento da fórmula.
- Como a NTM M resolve SAT em tempo polinomial, concluímos que SAT $\in \mathcal{NP}$.

Problema da Satisfatibilidade (SAT)

- Para concluir a prova de que SAT é NP-complete, é necessário mostrar que L_{SAT} é NP-hard.
- É preciso mostrar que toda linguagem em $\mathcal{NP}$ é redutível em tempo polinomial para L_{SAT}.
- Isso pode parecer impossível, uma vez que $\mathcal{NP}$ contém infinitas linguagens.
- Inclusive, muitas dessas linguagens nem mesmo possuem um alfabeto em comum.
- Como fazer?
- Explorar o fato de que todas as linguagens em $\mathcal{NP}$ são aceitas por NTM em tempo polinomial.
- Em vez de se concentrar nas linguagens, a prova se baseia nas propriedades das NTMs que as aceitam.
- Provê um método geral de redução para L_{SAT} que funciona para qualquer linguagem em $\mathcal{NP}$.

Problema da Satisfatibilidade (SAT)

Teorema 15.8.3 (Sudkamp) – Teorema de Cook

O Problema da Satisfatibilidade é NP-hard.

- Seja L uma linguagem decidida por uma NTM M cuja complexidade de tempo é limitada por um polinômio $p(n)$.
- A redução de L para SAT transforma as computações de M com string de entrada u em uma fórmula CNF $f(u)$ tal que $u \in L(M)$ sss $f(u)$ é satisfatível.
- Essa redução deve ser determinística e ter complexidade polinomial.

Problema da Satisfatibilidade (SAT)

- A ideia geral da redução é criar uma fórmula $f(u)$ formada por uma série de cláusulas que capturam toda a sequência de computação de M para a entrada u.
- Assim, se M aceita u, então $f(u)$ é satisfatível; e se M rejeita u, então $f(u)$ é insatisfatível.
- A fórmula $f(u)$ é composta por três tipos de variáveis.

Variable		Interpretation (When Satisfied)
$Q_{i,k}$	$0 \le i \le m$ $0 \le k \le p(n)$	M is in state q_i at time k
$P_{j,k}$	$0 \le j \le p(n)$ $0 \le k \le p(n)$	M is scanning position j at time k
$S_{j,r,k}$	$0 \le j \le p(n)$ $0 \le r \le t$ $0 \le k \le p(n)$	Tape position j contains symbol a_r at time k

Problema da Satisfatibilidade (SAT)

- As cláusulas da fórmula podem ser de oito tipos distintos.
- Algumas das cláusulas descrevem as condições necessárias para a existência de M, como abaixo.

	Clause	Conditions	Interpretation
i)	State		
	$\bigvee_{i=0}^{m} Q_{i,k}$	$0 \leq k \leq p(n)$	For each time k, M is in at least one state.
	$\sim Q_{i,k} \vee \sim Q_{i',k}$	$0 \leq i < i' \leq m$ $0 \leq k \leq p(n)$	M is in at most one state (not two different states at the same time).

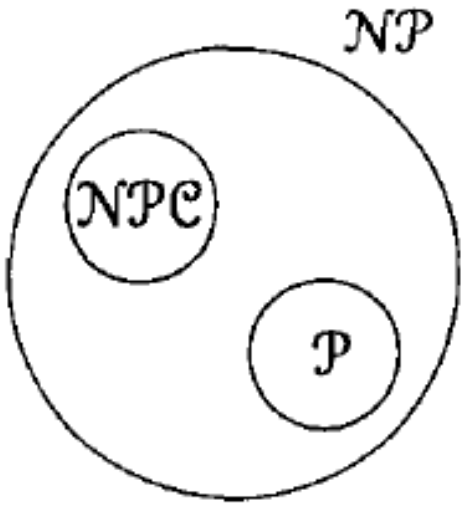
- Outros tipos de cláusulas cuidam da consistência da fita e dos efeitos de uma transição.
- A prova completa se encontra no livro do Sudkamp (8 páginas!). É bastante longa mas possível de ser entendida.

Problema da Satisfatibilidade (SAT)

- A construção de $f(u)$ a partir de M e u consiste na agregação de várias cláusulas.
- O número de cláusulas em $f(u)$ é uma função:
  1. do número de estados m de M e do número de símbolos t de Σ;
  2. do tamanho n da string de entrada u; e
  3. do limite $p(n)$ sobre o tamanho da computação de M.
- O valores m e t são obtidos de M e são independentes da string de entrada.
- Portanto, o número de cláusulas é polinomial em $p(n)$.
- Com isso, a fórmula $f(u)$ pode ser construída em um número de passos que cresce polinomialmente em relação ao tamanho da string de entrada.
- Assim, a redução está em $O(n^r)$. Vale notar que r em geral é um natural bastante grande. (De qualquer forma, a redução continua sendo polinomial.)

Relações Entre Classes de Complexidade

O diagrama abaixo ilustra como ficam as relações entre as classes de complexidade estudadas, se $\mathcal{P} \neq \mathcal{NP}$:



No caso improvável de $\mathcal{P} = \mathcal{NP}$, as duas classes se colapsam em uma só mas $\mathcal{NPC}$ continua sendo um subconjunto próprio de $\mathcal{NP}(= \mathcal{P})$ pois a linguagem vazia $\emptyset$ e seu complemento não são NP-complete.

Aula 06 – Classes de Complexidade

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação