Teoria da Computação
Complexidade de Tempo
parte 2

Leonardo Takuno
{leonardo.takuno@gmail.com}

Centro Universitário Senac

Sumário

Sumário

Teoria da complexidade

Definição: Seja $t : N \rightarrow R^{+}$ uma função. Define-se classe de complexidade de tempo, $TIME(t(n))$, a coleção de todas as linguagens que são decidíveis por uma máquina de Turing de tempo $O(t(n))$, formalmente

$$TIME(t(n)) = \{L | L \text{ é decidida por MT de tempo } O(t(n))\}$$

Teoria da complexidade

Exemplo: Considere $A = \{0^k 1^k | k \geq 0\}$.
Mostramos que $A \in TIME(n^2)$ e também que $A \in TIME(n)$.

Observação: Note que a mesma linguagem pode ser um membro de muitas classes de complexidades de tempo dependendo de como estamos planejando o nosso algoritmo.

Teoria da complexidade

$A = \{0^k 1^k | k \geq 0\}$

- Note que a diferença entre os algoritmos para decidir A são diferenças polinomiais, que é, $O(n^2)$ versus $O(n)$.
- Dizemos que os modelos são **polinomialmente equivalentes**: um modelo pode simular o outro com um aumento polinomial.

Sumário

Classe P

Definição: P é a classe de linguagens que são decidíveis em tempo polinomial em uma máquina de Turing Determinística,

$$P = \bigcup_k TIME(n^k), \text{ para } k \geq 0$$

Classe P

A classe P é interessante porque:

- P é invariante para todo modelo computacional que é equivalente à máquina de Turing de uma única fita.
- P, a grosso modo, corresponde a classe de problemas que são realisticamente solúveis por computador.

Classe P

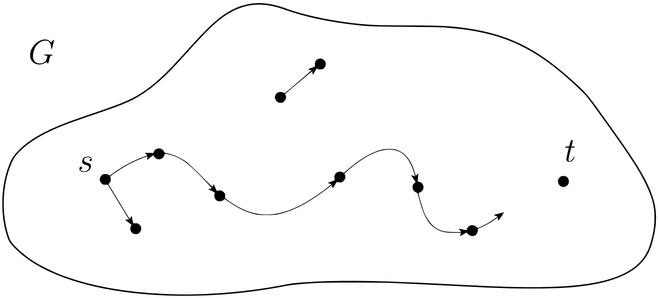
Exemplo: Note que esta definição de nossa linguagem $A = \{0^k 1^k | k \geq 0\}$ é claramente um membro de P sem levar em consideração de qual algoritmo exato usamos para decidí-lo.

$CAM \in P$

Teorema: $CAM \in P$, onde
$CAM = \{\langle G, s, t \rangle |\ G$ é um grafo dirigido que tem um caminho direcionado de s para t $\}$.

Idéia da Prova: Uma busca por força bruta para o caminho não funciona, pois tal algoritmo executará em tempo exponencial no número de nós de G. Entretanto, podemos ser mais espertos e implementar uma busca incremental. Então, provamos esse teorema apresentando um algoritmo em tempo polinomial que decide CAM.

$CAM \in P$



Existe um caminho de s a t ?

$CAM \in P$

Prova: M = "Sobre a entrada $\langle G, s, t \rangle$:

1. Coloque uma marca no nó s.
2. Repita as seguintes instruções até que nenhum nó adicional seja marcado:
3. Faça uma varredura em todas as arestas de G. Se existe uma aresta (a, b) do nó a marcado para um nó b não marcado, então marque o nó b.
4. Se t é marcado, aceite; caso contrário, rejeite."

$CAM \in P$

Análise. A estrutura de repetição no passo 3 leva $O(|E|)$ (onde $|E|$ é o número de arestas) para executar. O loop é executado $O(|V|)$ vezes (onde $|V|$ é o número de vértices). Tempo total é igual à $O(|E| \times |V|)$. Este tempo é polinomial ($O(n^2)$) no tamanho da entrada. Assim $CAM \in P$. $\square$

$PRIM - ES \in P$

Teorema: $PRIM - ES \in P$, onde
$PRIM - ES = \{\langle x, y\rangle | x \text{ e } y \text{ são primos entre si}\}$.

Dois números são primos entre si se 1 é o maior inteiro que divide ambos.
Ex:

- 10 e 21 são primos entre si.
- 10 e 22 não são primos entre si.

$PRIM - ES \in P$

Teorema: $PRIM - ES \in P$, onde
$PRIM - ES = \{\langle x, y\rangle | x \text{ e } y \text{ são primos entre si}\}$.

Idéia da prova:

- Buscar todos os possíveis divisores de ambos os números e aceita se nenhum deles é maior que 1. (Força Bruta - tempo de execução exponencial).
- Algoritmo de Euclides para computar o máximo divisor comum.

$PRIM - ES \in P$

$$\boxed{PRIM - ES = \{\langle x, y\rangle | x \text{ e } y \text{ são primos entre si}\}.}$$

Prova: O algoritmo euclidiano E é como segue:

$E =$ "Sobre a entrada $\langle x, y\rangle$, onde x e y são números naturais:

1. Repita até que $y = 0$.
2. Atribua $x \leftarrow x \mod y$.
3. $troca(x, y)$
4. Dê como saída x."

O algoritmo R resolve $PRIM - ES$, usando E como uma sub-rotina.

$R =$ "Sobre a entrada $\langle x, y\rangle$, onde x e y são números naturais:

1. Rode E sobre $\langle x, y\rangle$.
2. Se o resultado for 1, aceite. Caso contrário, rejeite."

Sumário

Classe NP

- Em teoria de complexidade computacional, **NP** é uma das mais fundamentais classe de complexidades.
- A abreviação NP se refere a **tempo polinomial não-determinístico** (*non-deterministic polynomial time*) que denota o conjunto de problemas que são decidíveis em tempo polinomial por uma máquina de Turing não-determinística.

Classe NP

Definição: Definimos a classe de complexidade de tempo $NTIME(t(n))$ como:
$NTIME(t(n)) = \{$ L | L é uma linguagem decidida por uma máquina de Turing não-determinística de tempo $O(t(n))\}$

Definição:

$$NP = \bigcup_k NTIME(n^k), \text{ para } k \geq 0$$

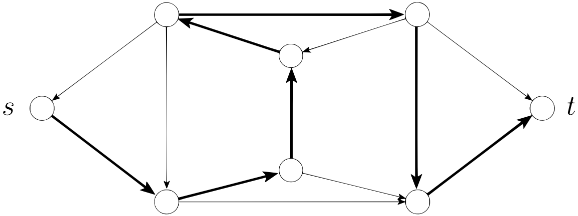
Caminho Hamiltoniano

- Um caminho hamiltoniano em um grafo é um caminho direcionado de s para t que passa por cada nó exatamente uma vez.
- Nenhum algoritmo determinístico de tempo polinomial é conhecido para decidir esta linguagem.

Caminho Hamiltoniano

Teorema:

$CAMHAM = \{\langle G, s, t\rangle |$ G é um grafo direcionado com um caminho hamiltoniano de s para t $\} \in NP$



Caminho Hamiltoniano

Teorema: $CAMHAM \in NP$

Prova: Construímos uma máquina de Turing não-determinística que decida $CAMHAM$ em tempo polinomial.

N1 = "Sobre a entrada $\langle G, s, t \rangle$:

1. Não deterministicamente gere uma permutação de m números p_1, ..., p_m tal que $1 \leq p_i \leq m$ onde m é o número de nós do grafo G.
2. Verifique se $p_1 = s$ e $p_m = t$. Se o teste falhar, *rejeite*.
3. Para cada i entre 1 e m - 1, verifique se (p_i, p_{i+1}) é uma aresta de G. Se alguma não for, *rejeite*. Caso contrário, a lista gerada de números representa o caminho Hamiltoniano, então *aceite*."

Análise: É fácil ver que todos os estágios executam em tempo polinomial. □

Classe NP

Podemos definir a classe NP de maneira alternativa usando verificadores determinísticos de tempo polinomial.

Definição: Um **verificador** para uma linguagem A é uma MT determinística V, onde

$$A = \{w|\ V \text{ aceita } \langle w, c\rangle \text{ para alguma cadeia } c\}$$

Aqui a string c é chamada **certificado**, ou **prova**. Medimos o tempo de um verificador em termos do comprimento de w.

Classe NP

Uma linguagem A é **polinomialmente verificável** se ela tem um verificador de tempo polinomial.

Definição: NP é a classe das linguagens que têm verificadores de tempo polinomial.

Caminho Hamiltoniano (revisitado)

Teorema: $CAMHAM \in NP$

Prova #2: Agora vamos mostrar que existe um verificador de tempo polinomial para o caminho Hamiltoniano. Seja c um caminho Hamiltoniano $\langle p_1 \rightsquigarrow p_m \rangle$, então construímos o verificador V como segue:

V = "Sobre a entrada $\langle\langle G, s, t\rangle, c\rangle$:

1. Verifique que $|p_1 \rightsquigarrow p_m| = m - 1$, senão *rejeite*.
2. Verifique que $p_1 \rightsquigarrow p_m$ não há repetições, se houver *rejeite*.
3. Checar se $p_1 = s$ e $p_m = t$. Se o teste falhar, *rejeite*.
4. Para cada i entre 1 e $m - 1$, checar se (p_i, p_{i+1}) é uma aresta de G. Caso algum não seja, rejeite.
5. Todos os testes passaram, aceite."

Classe NP

> **Teorema**: Uma linguagem está em NP sse ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial.

Idéia da prova: Mostramos como converter um verificador de tempo polinomial para uma MTN de tempo polinomial equivalente e vice e versa.

Classe NP

Teorema: Uma linguagem está em NP sse ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial.

Idéia da prova: A MTN simula o verificador advinhando o certificado. O verificador simula a MTN usando o ramo de computação de aceitação como certificado.

Classe NP

Teorema: Uma linguagem está em NP sse ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial.

Prova: Para a direção de frente, suponha que $A \in NP$ e mostre que A é decidida por uma MTN de tempo polinomial N. Seja V o verificador de tempo polinomial para A que existe pela definição de NP. Assuma que V seja uma MT que roda em tempo n^k e construa N da seguinte maneira:

N $=$ "Sobre a entrada w de comprimento n:

1. Não-deterministicamente selecione uma cadeia c de comprimento no máximo n^k
2. Rode V sobre $\langle w, c \rangle$.
3. Se V aceita, *aceite*. Caso contrário *rejeite*.

Classe NP

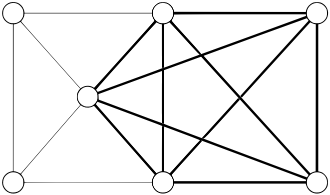
Teorema: Uma linguagem está em NP sse ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial.

Para provar a outra direção do teorema, assuma que A seja decidida por uma MTN de tempo polinomial N e construa um verificador de tempo polinomial V da seguinte maneira.
V = "Sobre a entrada $\langle w, c \rangle$, onde w e c são cadeias:

1. Simule N sobre a entrada w, tratar cada símbolo de c como uma descrição da escolha não-determinística a fazer a cada passo.
2. Se esse ramo da computação de N aceita, *aceite*; caso contrário, *rejeite* ." $\square$

Clique $\in NP$

Um clique em um grafo não-direcionado é um subgrafo, no qual todo par de nós está conectado por uma arestas. Um k-clique é um clique que contém k nós.



5-clique

CLIQUE $= \{\langle G, k\rangle |$ G é um grafo não-direcionado com um *k-clique*$\}$

Clique $\in NP$

Teorema: Clique está em NP

Prova #1: Construir um verificador V para Clique
V = "Sobre a entrada $\langle\langle G, k\rangle, c\rangle$

1. Teste se c é um conjunto de k nós em G.
2. Teste se G contém todas as arestas conectando nós em c.
3. Se ambos os testes retornam positivo, *aceite*; caso contrário, *rejeite*."

Aqui o estágio 1 e 2 executam em $O(n^2)$, logo o verificador determinístico executa em tempo polinomial

Clique $\in NP$

Teorema: Clique está em NP

Prova $\#2$**:** Construir um decisor não determinístico de tempo polinomial

N = "Sobre a entrada $\langle G, k \rangle$

1. Não deterministicamente selecione um conjunto Q de k nós de G.
2. Teste se G contém todas as arestas conectando nós em Q.
3. Se sim, *aceite*; caso contrário, *rejeite*."

O estágio 2 roda em $O(n^2)$ com $n = |\langle G, k \rangle|$. Portanto, a MT executa em tempo polinomial não determinístico.

Soma de subconjuntos

$SOMA - SUBC = \{\langle S, t\rangle |\ S = \{x_1, \cdots, x_k\}$ e para algum $\{y_1, \cdots, y_l\} \subseteq \{x_1, \cdots, x_k\}$, temos $\Sigma y_i = t\}$

Teorema: $SOMA - SUBC$ está em NP

Prova #1: O que se segue é um verificador V para $SOMA - SUBC$

V = "Sobre a entrada $\langle\langle S, t\rangle, c\rangle$

1. Teste se c é uma coleção de números que somam t.
2. Teste se S contém todos os números de c.
3. Se ambos os testes retornem positivo, **aceite**; caso contrário, **rejeite**".

Soma de subconjuntos

$SOMA - SUBC = \{\langle S, t\rangle |\ S = \{x_1, \cdots, x_k\}$ e para algum $\{y_1, \cdots, y_l\} \subseteq \{x_1, \cdots, x_k\}$, temos $\Sigma y_i = t\}$

Teorema: $SOMA - SUBC$ está em NP

Prova #2**:** Construímos uma MTN de tempo polinomial $SOMA - SUBC$

N = "Sobre a entrada $\langle S, t\rangle$

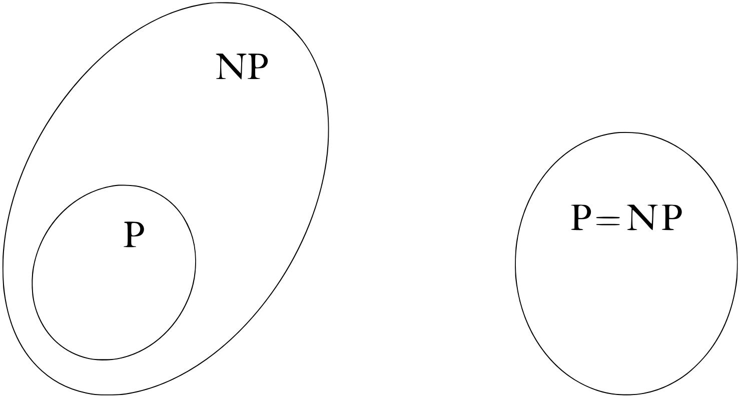
1. Não-deterministicamente selecione um subconjunto c dos números em S.
2. Teste se c é uma coleção de números que somam t.
3. Se o teste der positivo, **aceite**; caso contrário, **rejeite**".

Problemas que não estão em NP

$\overline{CAMHAM}$, $\overline{CLIQUE}$, $\overline{SOMA-SUBC}$ não são membros de NP.

- Verificar que algo não está presente parece mais difícil que verificar que está presente.
- coNP: linguagens que são complemento das linguagens em NP.

P vs NP



P vs NP

Como MT é considerado um caso de MTN, então temos

$$P \subset NP$$

Ainda é uma questão aberta se $P = NP$, já que atualmente sabemos que algoritmos para problemas NP usam tempo exponencial

$$NP \subset EXPTIME = \bigcup_k TIME(2^{n^k})$$

(Lembre-se: simular uma máquina de Turing não-determinística em uma MT determinística precisamos de tempo exponencial)

Sumário

NP-Completude

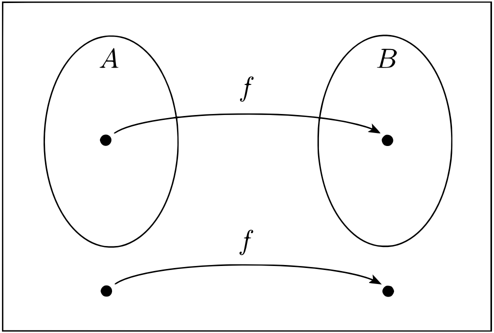
Definição: Uma função $f : \Sigma^* \rightarrow \Sigma^*$ é uma **função computável em tempo polinomial** se existe alguma máquina de Turing de tempo polinomial M que pára com exatamente $f(w)$ na sua fita, quando iniciada sobre qualquer entrada w.

Definição: Uma linguagem A é **redutível por mapeamento em tempo polinomial**, ou simplesmente **redutível em tempo polinomial**, à linguagem B, em símbolos $A \leq_p B$, se existe uma função computável em tempo polinomial $f : \Sigma^* \rightarrow \Sigma^*$, onde para toda w,

$$w \in A \Leftrightarrow f(w) \in B$$

A função f é chamada redução de tempo polinomial de A para B.

$A \leq_p B$



NP-Completude

Teorema: Se $A \leq_p B$ e $B \in P$, então $A \in P$

Prova: Seja M uma algoritmo de tempo polinomial que decide B e f a função de redução de tempo polinomial de A para B.
Descrevemos um algoritmo polinomial N que decide A da seguinte forma:
N = "Sobre a entrada w:

1. Compute $f(w)$
2. Rode M sobre a entrada $f(w)$ e dê como saída o que quer que M dê como saída.'

Claramente, se $w \in A$ então $f(w) \in B$ pois f é uma redução. É fácil ver que N executa em tempo polinomial. □

NP-Completude

Definição: Uma linguagem B é NP-Completa se satisfazem duas condições:

1. $B \in NP$, e
2. todo $A \in NP$ é redutível em tempo polinomial a B.

NP-Completude

$\boxed{\textbf{Teorema}\text{: Se } B \text{ for NP-Completa e } B \in P\text{, então } P = NP.}$

Prova: Esse teorema segue diretamente da definição da redutibilidade de tempo polinomial.

NP-Completude

Teorema: Se B for NP-Completa e $B \in P$, então $P = NP$.

Esse teorema enfatiza a importância dos problemas NP-Completos. No caso de soluções determinísticas de tempo polinomial serem encontradas para um problema NP-Completo, então a classe de complexidade NP igualaria a classe de complexidade P.

NP-Completude

Teorema: Se B for NP-Completa e $B \leq_p C$ para $C \in NP$, então C é NP-Completa.

Prova: Seja g_i uma redução de tempo polinomial de qualquer linguagem $A_i \in NP$ e f uma redução de tempo polinomial de B para C. Sabemos que g_i deve existir para toda linguagem $A_i \in NP$ pois B é NP-Completa. Isto nos fornece uma redução polinomial $f \circ g_i$ de qualquer linguagem A_i para C. $\square$