

DCC-IME-USP

Complexidade de Contagem

Parte 1

Thales A. B. Paiva
thalespaiva@gmail.com

5 de novembro de 2015

Sumário

1	Definições Preliminares	2
2	Classes de contagem	3
2.1	A classe $\#\mathbf{P}$	3
2.2	A classe \mathbf{GapP}	4
2.3	Algumas classes de contagem	5
3	$\#\mathbf{P}$-completude	7
3.1	Definição	7
3.2	Reduções parcimoniosas	7
3.3	$\#\mathbf{SAT}$ é $\#\mathbf{P}$ -completo	7
4	Teorema de Valiant	9
5	Poder das Classes de Contagem	14

1 Definições Preliminares

Nesta seção, introduzimos algumas notações e definições. Foram extraídas principalmente de [5].

Fixe $\Sigma = \{0, 1\}$ nosso alfabeto. Então Σ^* é o conjunto de todas as palavras sobre esse alfabeto. Σ^n é o conjunto de todas as strings de tamanho n sobre esse alfabeto.

Definição 1.0.1 (máquina-NP). *Uma **Máquina NP** é uma máquina de Turing não determinística cujo tempo de execução é limitado por um polinômio no tamanho da entrada.*

Definição 1.0.2 (\overline{M}). *Seja M uma máquina-NP, denotamos por \overline{M} a máquina-NP que simula M invertendo a decisão de M sobre uma dada entrada.*

Definição 1.0.3. *Seja M uma máquina-NP, denotamos por $\#M(x)$ o número de caminhos de aceitação para uma entrada x , ou o número de certificados de que x pertence à linguagem reconhecida por M . Fica claro, então, que $\#\overline{M}(x)$ é o número de caminhos de rejeição para uma entrada x .*

Definição 1.0.4. *Seja M uma máquina-NP, chamamos de **Intervalo** (Gap) de $M(x)$, denotado por $\Delta M(x)$, a diferença entre o número de caminhos de aceitação e de rejeição para uma entrada x . Então*

$$\Delta M(x) = \#M(x) - \#\overline{M}(x)$$

Note que, se o tempo de execução de M é limitado pelo polinômio $|x|^k$ para cada entrada x , temos que os valores de $\#M(x)$ e de $\Delta M(x)$ são limitados por $2^{|x|^k}$. Pois lembre que $2^{|x|^k}$ é o número de folhas de uma árvore de altura $|x|^k$.

Definição 1.0.5. *A classe de problemas de função **FP** contém as funções $f : \Sigma^* \rightarrow \mathbb{N}$ que são computáveis em tempo polinomial.*

Note que **FP** é generalização da classe de problemas de decisão **P**.

2 Classes de contagem

Uma classe de contagem é um conjunto de soluções de problemas cuja formulação é feita sobre o número de certificados para problemas em NP. Nesta seção, definimos as duas principais classes de contagem, $\#P$ e \mathbf{GapP} . Jugamos que elas são as principais pois, podemos caracterizar outras classes de problemas a partir delas.

2.1 A classe $\#P$

A classe $\#P$ contém os as funções cujas imagens são dadas pelo número de certificados de problemas em NP.

Podemos definir $\#P$ facilmente em termos de uma máquina-NP M e $\#M$.

Definição 2.1.1. *A classe $\#P$ contém as funções f tais que existe uma máquina-NP M para que toda entrada x em Σ^* , temos $f(x) = \#M(x)$.*

Uma definição alternativa para $\#P$ usando máquinas de Turing polinomiais, como a de [3] é dada abaixo.

Definição 2.1.2 (De [3]). *Uma função $f : \Sigma^* \rightarrow \mathbb{N}$ está em $\#P$ se existe um polinômio $p : \mathbb{N} \rightarrow \mathbb{N}$ e uma máquina de Turing polinomial M tal que, para todo $x \in \Sigma^*$, tem-se que*

$$f(x) = \left| \{y \in \{0, 1\}^{p(|x|)} : M(x, y) = 1\} \right|$$

Como Arora aponta em [3], a principal questão em aberto sobre $\#P$ é se seus problemas podem ser resolvidos eficientemente ($\#P \stackrel{?}{=} \mathbf{FP}$). É direto a definição que:

- $\#P \subset \mathbf{FP} \implies \mathbf{NP} = \mathbf{P}$.
- $\#P \subset \mathbf{PSPACE}$.
- $\#P \not\subset \mathbf{FP}$ (sob a nossa definição de \mathbf{FP}).
- $\mathbf{P} = \mathbf{PSPACE} \implies \mathbf{P} = \#P$.

Porém, não sabemos se $\mathbf{NP} = \mathbf{P} \stackrel{?}{\implies} \#P = \mathbf{FP}$.

Duas propriedades esperadas sobre $\#P$ são o fechamento sob adição e sob multiplicação.

Propriedade 2.1.1. *$\#P$ é fechada sob a adição. Ou seja, para todas funções f_1 e f_2 em $\#P$, existe uma função f também em $\#P$ tal que, para todo x em Σ^* , temos que*

$$f(x) = f_1(x) + f_2(x)$$

Demonstração. Tome M_1 e M_2 máquinas NP tais que

$$\#M_1(x) = f_1(x) \text{ e } \#M_2(x) = f_2(x), \text{ para todo } x \text{ em } \Sigma^*.$$

Construa uma máquina-NP M cuja primeira ramificação seja simular a máquina M_1 ou a máquina M_2 . Claro que

$$\begin{aligned}\#M(x) &= \#M_1(x) + \#M_2(x) \\ &= f_1(x) + f_2(x) \\ &= f(x).\end{aligned}$$

□

Propriedade 2.1.2. $\#P$ é fechada sob a multiplicação.

Demonstração. Segue da propriedade anterior. □

Apesar dessas propriedades, é óbvio que $\#P$ não é fechada sob a subtração. Para resolver esse problema e facilitar o estudo de $\#P$, introduzimos a classe **GapP**.

2.2 A classe **GapP**

A classe **GapP**, definida por Fenner em [4], generaliza a classe $\#P$ sem perda de poder computacional. Sua definição, analoga à de $\#P$ para ΔM , é dada a seguir.

Definição 2.2.1. A classe **GapP** contém as funções f tais que existe uma máquina-NP M para que toda entrada x em Σ^* , temos $f(x) = \Delta M(x)$.

É fácil ver que **GapP** é fechado sob negação.

Propriedade 2.2.1. Se $f \in \mathbf{GapP}$, então $-f \in \mathbf{GapP}$.

Demonstração. Tome M uma máquina-NP tal que $f(x) = \Delta M(x)$, para todo x em Σ^* . Então $-f(x) = \Delta \bar{M}(x)$. □

GapP é uma classe ao menos tão poderosa quando a classe $\#P$. Ou seja, que toda função em $\#P$ está em **GapP**.

Teorema 2.2.1. Se $f \in \#P$, então $f \in \mathbf{GapP}$.

Demonstração. Seja M uma máquina-NP tal que, para todo x em $\#P$, $f(x) = \#M(x)$. Queremos construir uma máquina-NP N tal que $f(x) = \Delta N(x)$. Considere a seguinte construção de uma máquina-NP N :

- N simula M com entrada x em Σ^* .
- Se M aceita x , então N aceita x .
- Se M rejeita x , então N ramifica em duas configurações, uma de aceitação, outra de rejeição.

Temos então que:

$$\begin{aligned}\#N(x) &= \#M(x) + \#\overline{M}(x), \text{ e que} \\ \#\overline{N}(x) &= \#\overline{M}(x).\end{aligned}$$

Assim:

$$\begin{aligned}\Delta N(x) &= \#N(x) - \#\overline{N}(x) \\ &= (\#M(x) + \#\overline{M}(x)) - (\#\overline{M}(x)) \\ &= \#M(x) \\ &= f(x).\end{aligned}$$

□

Mais à frente, veremos que **GapP** é exatamente o fechamento de **#P** sob a subtração e que **#P** e **GapP** têm poder de computação equivalente.

2.3 Algumas classes de contagem

Agora podemos definir um pouco melhor o termo **classe de contagem** como uma classe de funções que têm uma caracterização simples em termo de **#P** e **GapP**, (como Fortnow em [5]).

É útil caracterizar classes em termos de **GapP** e de **#P** para que se possa usar as propriedades de fechamento dessas classes, que veremos em próximas seções. Algumas caracterizações são:

Definição 2.3.1. A classe **NP** contém as linguagens L tais que existe uma função f em **#P** para que todo x em Σ^* :

- Se $x \in L$, então $f(x) > 0$.
- Se $x \notin L$, então $f(x) = 0$.

Essa caracterização não é muito útil mas é ilustrativa. Uma outra classe com caracterização simples é a **UP** (*Unique P*).

Definição 2.3.2. A classe **UP** contém todas as linguagens L tais que existe uma função f em **#P** para que todo x em Σ^* :

- Se $x \in L$, então $f(x) = 1$.
- Se $x \notin L$, então $f(x) = 0$.

Definição 2.3.3. A classe $\oplus P$ contém todas as linguagens L tais que existe uma função f em **#P** para que todo x em Σ^* :

- Se $x \in L$, então $f(x)$ é par.

- Se $x \notin L$, então $f(x)$ é ímpar.

Uma caracterização bem útil é a da classe **PP**. Definida por Gill, como a classe de problemas de decisão tais que existe uma máquina de Turing polinomial probabilística tal que, x pertence a **PP** se e somente se $\mathbb{P}(M(x) = 1) > \frac{1}{2}$. **PP** é mais forte do que **BPP**, inclusive **BPP** \subset **PP**, pois o quão próximo de $\frac{1}{2}$ estão as probabilidades de erro e acerto podem depender da entrada. Isso faria com que o algoritmo devesse ser rodado por tempo exponencial para aumentar a probabilidade de acerto, e não por tempo polinomial através do limite de Chernoff.

Essa classe é particularmente importante no estudo de classes de contagem, pois veremos mais para frente que **PP** é tão poderosa quanto **#P**.

Definição 2.3.4. A classe **PP** contém todas as linguagens L tais que existe uma função f em **GapP** para que todo x em Σ^* :

- Se $x \in L$, então $f(x) > 0$.
- Se $x \notin L$, então $f(x) \leq 0$.

3 #P-completeness

Nesta seção curta, definiremos #P-completeness e mostramos alguns exemplos de funções #P-completas.

3.1 Definição

Uma função de #P é #P-completa se um algoritmo polinomial para f implica que $\#P \subset FP$. Para a definição formal, podemos usar classes com oráculos.

Lembre que uma classe FP com oráculo para uma função f , denotado por FP^f é o conjunto de funções eficientemente computáveis dado que se sabe calcular o valor de f em um passo computacional.

Definição 3.1.1. *Uma função f de #P é #P-completa se cada g em #P pertence a FP^f .*

Problemas NP-completos costumam ter formulações de contagem #P-completas. Exemplos são #SAT, #3SAT, #CLIQUE, e #HAM.

3.2 Reduções parcimoniosas

Em complexidade de contagem estamos interessados em reduções de A para B que preservem o número de soluções.

Definição 3.2.1. *Dadas duas linguagens A e B de NP. Uma redução parcimoniosa é uma redução $h : A \rightarrow B$ tal que h é calculada em tempo polinomial e o número de certificados para $x \in A$ é o mesmo de $f(x) \in B$.*

Como pode ser difícil (ou impossível) encontrar reduções parcimoniosas, podemos relaxar um pouco o tipo de redução que precisamos. Basta que, para nossa redução de A para B , seja possível reconstruir o número de soluções de A através do número de soluções para B .

Definição 3.2.2 (Erik Demaine). *Dadas duas linguagens A e B de NP. Uma redução c -moniosa é uma redução $h : A \rightarrow B$ tal que h é calculada em tempo polinomial e o número de certificados para $x \in A$ é o mesmo de $f(x) \in B$ multiplicado por um inteiro c .*

3.3 #SAT é #P-completo

Demonstração. Basta ver que a redução de Cook-Levin para mostrar que SAT é NP-completa é, em particular, uma redução de Levin. Esta redução consiste de três funções f, g, h de uma linguagem $L \in NP$ para SAT tal que:

1. $x \in L \Leftrightarrow f(x) \in \text{SAT}$.
2. y é certificado de que $x \in L \Leftrightarrow g(y)$ é certificado de que $f(x) \in \text{SAT}$.

3. $h(z)$ é certificado de que $x \in L \Leftrightarrow z$ é certificado de que $f(x) \in \mathbf{SAT}$.

Então, o número de certificados para cada $x \in L$, é o mesmo de certificados para $f(x) \in \mathbf{SAT}$. □

Note que isso implica que $\#\mathbf{3SAT}$ é $\#\mathbf{P}$ -completa, pois a redução de \mathbf{SAT} para $\mathbf{3SAT}$ também é parcimoniosa. Usaremos esse fato para provar o Teorema de Valiant.

4 Teorema de Valiant

Valiant mostrou que calcular o número de emparelhamentos perfeitos num grafo bipartido é $\#P$ -completo. Isso é surpreendente pois o problema de decisão associado está em P .

Para demonstrar esse teorema, primeiro lembramos que calcular o número de emparelhamentos perfeitos num grafo bipartido é o mesmo que encontrar o número de coberturas por ciclos de um grafo direcionado. Ainda, se considerarmos A como uma matriz de adjacência de um digrafo G , calcular o permanente de A é calcular o número de coberturas por ciclos de G . Finalmente, reduzimos o problema $\#3SAT$, que é $\#P$ -completo, ao cálculo do número de coberturas por ciclos de um digrafo.

Teorema 4.0.1 (Valiant 1979). **PERMANENTE** é $\#P$ -completo.

Demonstração. Queremos reduzir $\#3SAT$ a **PERMANENTE**. Para isso, dada uma fórmula ϕ na 3-FNC, devemos construir um digrafo G cujas coberturas por ciclos correspondam a valorações das variáveis de ϕ que a satisfazem.

Seja ϕ uma fórmula na forma normal conjuntiva com exatamente 3 variáveis por cláusula sobre as variáveis em $X = \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$. Assim, temos que

$$\phi = \bigwedge_{i=1}^m C_i = C_1 \wedge C_2 \wedge \dots \wedge C_m,$$

onde cada cláusula C_i é da forma $C_i = (y_{i1} \vee y_{i2} \vee y_{i3})$, com cada y_{ij} em X .

Iremos considerar três dispositivos para construir o digrafo G relacionado. Cada dispositivo representa alguma característica ou restrição de uma fórmula na 3-FNC em um digrafo. Os dispositivos e suas descrições resumidas são dadas a seguir.

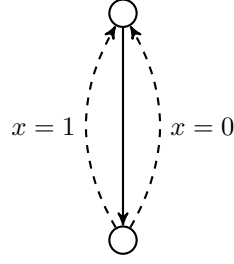
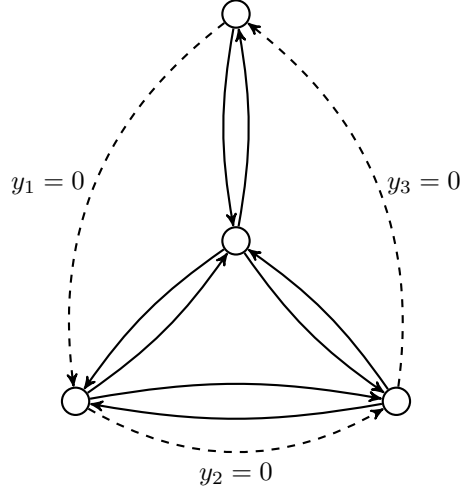
Dispositivo de Valoração D_V :

Cada variável gera um dispositivo de valoração. Sua valoração está associada univocamente a como os dois vértices deste dispositivo são cobertos por uma cobertura por ciclos de G . A representação gráfica de D_v para uma variável x pode ser vista na Figura 4.0.1, em que 0 e 1 representam respectivamente os valores verdadeiro e falso. Dois dos arcos estão pontilhados pois poderão ser acoplados a outros dispositivos na construção de G .

Dispositivo de Cláusula D_C :

Semelhante ao dispositivo anterior, cada cláusula gera um dispositivo de cláusula. Este dispositivo captura a restrição de que ao menos uma das expressões componentes de uma cláusula deve ser verdadeiro. Assim, para uma cláusula C , cada ciclo em seu dispositivo associado corresponde a uma valoração que faz C ser satisfeita.

Nesse dispositivo, três arcos externos representam as valorações que fazem cada expressão de uma cláusula $C = (y_1 \vee y_2 \vee y_3)$ ser falsa. Podemos ver na Figura 4.0.2 que não pode haver cobertura por ciclos que percorra os três arcos externos. Também vemos

Figura 4.0.1: Dispositivo de valoração para variável x Figura 4.0.2: Dispositivo de cláusula para a cláusula $C = (y_1 \vee y_2 \vee y_3)$

que para qualquer aresta externa ou par de arestas externas, há apenas uma cobertura por ciclos do dispositivo. Aqui, os arcos externos estão tracejados pois serão conectados a dispositivos de valoração através de dispositivos de valoração exclusiva.

Dispositivo de Valoração Exclusiva D_{\oplus} :

Este dispositivo é o que garante a consistência entre os dispositivos de valoração e de cláusulas. Queremos garantir que uma cobertura por ciclos de G não possua arestas que se contradizem. Ou seja, garantir que as coberturas dos dispositivos de valoração para cada variável de uma cláusula C não entrem em conflito com as coberturas de C .

Como exemplo, suponha que a fórmula ϕ seja composta por apenas duas cláusulas e dada por

$$\phi = C_1 \wedge C_2 = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee x_3 \vee \overline{x}_4).$$

Um esquema do digrafo para ϕ é dado na Figura 4.0.3. Nesta figura, os D_{\oplus} representam os dispositivos de valoração exclusiva que devem ser acoplados às arestas contraditórias.

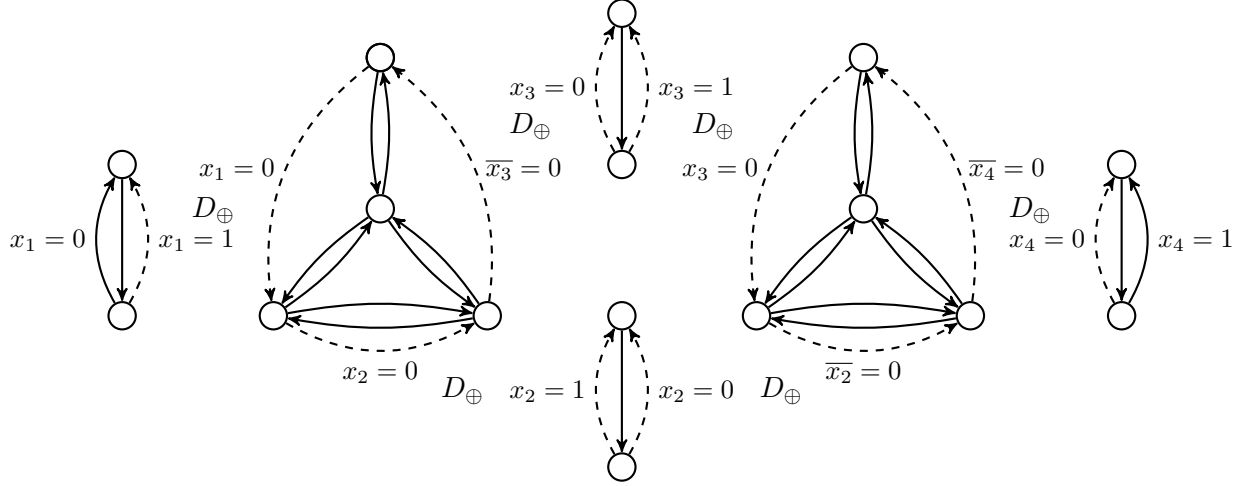


Figura 4.0.3: D_{\oplus} para forçar o uso exclusivo de arestas contraditórias

Gostaríamos que D_{\oplus} garantisse que exatamente uma das arestas contraditórias fosse atravessada. Porém, se isso fosse possível, esta redução seria parcimoniosa, o que implicaria $\mathbf{P} = \mathbf{NP}$. Pois como sabemos dizer se $\text{perm}(A) \geq 0$ eficientemente, saberíamos dizer se uma instância de **3SAT** tem ao menos uma solução.

Para desconsiderar coberturas por ciclos de G que implicariam valorações contraditórias, usaremos faremos de D_{\oplus} um digrafo com pesos nas arestas. O permanente, então, passa a ser a soma dos pesos de todas as coberturas por ciclos de G .

Assim, este digrafo deve ser tal que coberturas que correspondem a valorações contraditórias tenham peso 0. Coberturas que correspondem a valorações válidas têm peso constante. Queremos então, as observar as seguintes propriedades na matriz de adjacência $M_{D_{\oplus}}$ deste dispositivo:

1. Seu permanente é 0. .
2. O permanente de $M_{D_{\oplus}}$ sem a primeira linha e sem a última coluna deve ser 0. O mesmo para a última linha e última coluna.
3. O permanente de $M_{D_{\oplus}}$ sem a primeira e última linha e sem a primeira e última colunas deve ser 0.
4. O permanente da matriz sem a primeira linha e última coluna deve ser o mesmo que o da matriz sem a primeira coluna e última linha. Ainda, esse valor deve ser uma constante maior que 0.

Veja que o dispositivo mostrado na Figura 4.0.4 e sua matriz de adjacências $M_{D_{\oplus}}$ dada a seguir satisfazem as exigências. Note ainda que a exigência 4 é satisfeita para a constante $k = 4$.

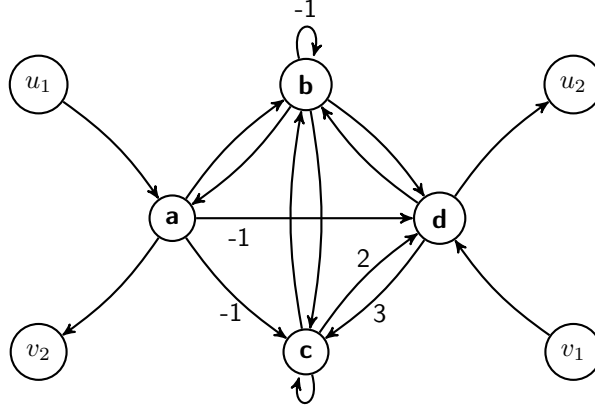


Figura 4.0.4: O dispositivo de valoração exclusiva D_{\oplus} sobre arcos (u_1, u_2) e (v_1, v_2)

$$M_{D_{\oplus}} = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 3 & 0 \end{bmatrix}$$

Nossa construção usa três dispositivos D_{\oplus} por cláusula, totalizando $3m$ dispositivos. Cada valoração que satisfaz ϕ corresponde a uma cobertura por ciclos de peso 4^{3m} . Assim, se $\#\phi$ denotar o número de soluções de ϕ :

$$\begin{aligned} perm(G) &= \#\phi 4^{3m} \\ \implies \#\phi &= \frac{perm(G)}{4^{3m}}. \end{aligned}$$

Como não estamos mais usando apenas entradas 0 e 1 na matriz de G , provamos que a generalização de **PERMANENT** é $\#\mathbf{P}$ -completa. Então, precisamos reduzir esta solução ao problema inicial.

Para simular arestas de peso 2 e 3, use os dispositivos da Figura 4.0.5

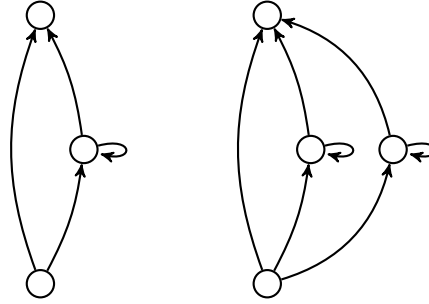
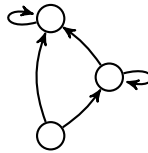


Figura 4.0.5: Dispositivo para substituir arestas de peso 2 e 3

Então resta apenas lidar com a entrada -1 da matriz. Para isso, consideramos o problema **PERMANENT MOD N** que obviamente se reduz a **PERMANENTGERAL**.

Assim, podemos calcular o permanente de $G \bmod N$, para N grande, digamos $N = 2^{8(3m)} + 1$. Então podemos substituir a aresta -1 pelo dispositivo da Figura 4.0.6.

E finalmente, o permanente de $G \bmod N$ é $(\#\phi)4^{3m}$ e, por causa do $\bmod N$, temos a correspondência com emparelhamentos perfeitos. \square

Figura 4.0.6: Dispositivo para substituir arestas de peso 2^n por n concatenações

5 Poder das Classes de Contagem

Nesta seção, estudamos as relações entre as classes de contagem e algumas classes conhecidas.

Teorema 5.0.1. *Toda função f em \mathbf{FP} também está em \mathbf{GapP} . Em particular, se $f(x)$ é não negativo para toda string x , então f também está em $\#\mathbf{P}$.*

Demonstração. Seja f uma função em \mathbf{FP} . Construa uma máquina-NP M tal que, dada uma entrada x :

1. Calcula $n = |f(x)|$.
2. Chuta n caminhos de computação.
3. Se $f(x) > 0$, então aceita todos os caminhos.
4. Senão, rejeita todos.

Claro que, para toda palavra x , $\Delta M = f(x)$ e, se $f(x) \geq 0$, então $\#M(x) = f(x)$. \square

Mostramos a seguir que \mathbf{GapP} e $\#\mathbf{P}$ têm o mesmo poder de computação.

Teorema 5.0.2. *Para toda função f , as seguintes afirmações são equivalentes:*

1. f pertence a \mathbf{GapP} .
2. f é dada pela diferença entre duas funções de $\#\mathbf{P}$.
3. f é dada pela diferença entre uma função de $\#\mathbf{P}$ e outra de \mathbf{FP} .
4. f é dada pela diferença entre uma função de \mathbf{FP} e outra de $\#\mathbf{P}$.

Demonstração.

$1 \Rightarrow 2$ É direto da definição pois

$$f \in \mathbf{GapP} \Rightarrow f = \Delta M \Rightarrow \#M - \#\overline{M}$$

$2 \Rightarrow 3$ Sejam f e g funções de $\#\mathbf{P}$. Então para algum par de máquinas-NP, M_1 e M_2 , temos que $f = \#M_1$ e $g = \#M_2$. Suponha que M_2 tem, para algum polinômio q no tamanho da entrada, $2^{q(|x|)}$ caminhos de computação (caso contrário, basta adicionar o número de caminhos restantes). Construa uma máquina N tal que:

1. N se ramifica em dois ramos, L e R .
2. Simula M_1 no ramo L e M_2 no ramo R .

Então, para qualquer x :

$$\begin{aligned} f(x) - g(x) &= \#M_1(x) - \#M_2(x) \\ &= \#M_1(x) + \#M_2(x) - 2^{q(|x|)} \\ &= \#N(x) - 2^{q(|x|)} \end{aligned}$$

E claro que $\#N \in \#\mathbf{P}$ e $2^{q(|x|)} \in \mathbf{FP}$.

Pelo fechamento sob a negação, $3 \Leftrightarrow 4$.

$3 \Rightarrow 1$ Sejam f função de $\#P$ e g função de FP . Seja M a máquina-NP tal que $\Delta M = f$. Construa N tal que, para cada palavra x , $N(x)$ é dada por $M(x)$ com mais $g(x)$ caminhos de rejeição. Então:

$$\begin{aligned}\#N(x) &= \#M(x) \\ \#\overline{N}(x) &= \#\overline{M}(x) + g(x)\end{aligned}$$

E assim:

$$\begin{aligned}\Delta N(x) &= \#N(x) - \#\overline{N}(x) \\ &= \#M(x) - \#\overline{M}(x) - g(x) \\ &= f(x) - g(x)\end{aligned}$$

□

Uma consequência deste teorema, é que $GapP \subset FP^{\#P[1]}$. Lembre que $FP^{\#P[k]}$ contém as funções computáveis em tempo polinomial dado que se pode fazer até k consulta a um oráculo de $\#P$.

Teorema 5.0.3. *Seja f função de FP e g função de $\#P$ (ou $GapP$). Então $g \circ f$ está em $\#P$ (ou $GapP$).*

Demonstração. Seja M máquina-NP tal que $\#M = g$. Construa a máquina-NP N que, para uma entrada x calcula $f(x)$ e simula $M(f(x))$. Claro que $\#N(x) = \#M(f(x)) = g(f(x))$.

□

Babai e Fortnow mostraram em [7] que qualquer função em $GapP$ pode ser expressada como polinômios construídos de maneira simples. Não provaremos este resultado.

Definição 5.0.1. *Um programa de aritmética retardada com substituição binária (RAB) é uma sequência $\{p_1, p_2, \dots\}$ de instruções tais que, para todo k , vale uma das condições seguintes:*

1. $p_k = 0$ ou $p_k = 1$;
2. $p_k = x_i$ para algum $i \leq k$;
3. $p_k = 1 - x_i$ para algum $i \leq k$;
4. $p_k = p_i - p_j$ para algum par $i, j < k$;
5. $p_k = p_i p_j$ para algum par i, j tais que $i + j \leq k$;
6. $p_k = p_j(x_i = 0)$ ou $p_k = p_j(x_i = 1)$, para algum par $i, j < k$. (substituição binária)

Dizemos que um RAB é uniforme se existe uma máquina de Turing polinomial que, com entrada 1^n imprime as primeiras n instruções.

Teorema 5.0.4. Para toda função f , as afirmações seguintes são equivalentes:

1. $f \in \mathbf{GapP}$
2. Existe um RAB uniforme se existe um polinômio $q(n)$ tal que $p_{q(n)}$ tem variáveis livres x_1, \dots, x_n e tal que, para cada palavra $x \in \Sigma^*$, tem-se que $f(x) = p_{q(n)}(x)$.

Vamos mostrar agora que \mathbf{GapP} e \mathbf{PP} têm mesmo poder computacional. Mas antes, mostramos uma caracterização alternativa para \mathbf{PP} .

Teorema 5.0.5. A classe \mathbf{PP} contém todas as linguagens L para que existe uma função f em \mathbf{GapP} tal que, para toda palavra x

- Se $x \in L$, então $f(x) > 0$
- Se $x \notin L$, então $f(x) < 0$

Demonstração. Sabemos, da definição de \mathbf{PP} , que se $x \in \mathbf{PP}$, então existe uma função g de \mathbf{GapP} tal que:

- Se $x \in L$, então $g(x) > 0$
- Se $x \notin L$, então $g(x) \leq 0$

Considere a função f tal que $f(x) = 2g(x) - 1$. Sabemos que $f \in \mathbf{GapP}$ e, f é tal que:

- Se $g(x) > 0$, então $f(x) > 0$
- Se $g(x) \leq 0$, então $f(x) < 0$

□

Dessa caracterização, é direto que \mathbf{PP} é fechada sob complemento.

Mostramos agora que \mathbf{PP} e \mathbf{GapP} têm mesmo poder computacional.

Teorema 5.0.6. $P^{PP} = P^{\mathbf{GapP}}$

Demonstração. Como \mathbf{GapP} é tão ou mais forte que \mathbf{PP} , precisamos apenas mostrar que toda função g de \mathbf{GapP} está em $\mathbf{FP}^{\mathbf{PP}}$. Considere a linguagem:

$$L = \{\langle x, k \rangle : g(x) > k\}.$$

$L \in \mathbf{PP}$, comprovado pelo fato de a função $f(x, k) = g(x) - k$ estar em \mathbf{GapP} . Então, dada uma palavra x , podemos calcular $g(x)$ fazendo uma busca binária com ajuda de um oráculo para L .

□

Finalmente, enunciamos, sem demonstração, o teorema de Toda sobre a relação das classes de contagem e a hierarquia polinomial.

Teorema 5.0.7 (Toda).

$$PH \subseteq P^{GapP[1]}$$

E suas consequências imediatas são:

1. $PH \subseteq P^{\#P[1]}$
2. $PH \subseteq P^{PP}$

Referências

- [1] Valiant, Leslie G. "The complexity of enumeration and reliability problems." *SIAM Journal on Computing* 8.3 (1979): 410-421.
- [2] Valiant, Leslie G. "The complexity of computing the permanent." *Theoretical computer science* 8.2 (1979): 189-201.
- [3] Arora, Sanjeev, and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [4] Fenner, Stephen A., Lance J. Fortnow, and Stuart A. Kurtz. "Gap-definable counting classes." *Journal of Computer and System Sciences* 48.1 (1994): 116-148.
- [5] Fortnow, Lance. "Counting complexity." *Complexity theory retrospective II* (1997): 81-107.
- [6] Papadimitriou, Christos H. *Computational complexity*. John Wiley and Sons Ltd., 2003.
- [7] Babai, László, and Lance Fortnow. "Arithmetization: A new method in structural complexity theory." *Computational Complexity* 1.1 (1991): 41-66.