que, assintoticamente, quase todas as funções (uma fração $(1 - o(1))$ delas) necessitam de circuitos maiores que $2^n/n$ para serem computadas.

Portanto, existem muitas (na verdade, a maioria) funções booleanas exponencialmente difíceis mas não conseguimos explicitar uma família natural dessas funções booleanas. Uma justificativa para a dificuldade de explicitarmos um problema difícil para o qual conseguimos provar uma cota inferior superpolinomial, como $2^n/n$, é dada na seção 7.2.

Também, decorre do Exercício 4.4 acima que $\mathsf{P} \subseteq \mathsf{P/poly}$. Atualmente, não é sabido se $\mathsf{NP} \nsubseteq \mathsf{P/poly}$.

Problema 7. $\mathsf{NP} \nsubseteq \mathsf{P/poly}$?

Se esse for o caso, ou seja a resposta para o problema acima é sim, como explicamos no parágrafo acima, concluímos que $\mathsf{P} \neq \mathsf{NP}$, resolvendo o Problema 5 (página 206).

4.2.1 CLASSES PROBABILÍSTICAS

O estudo sistemático da computabilidade e complexidade de modelos probabilísticos, bem como das classes de complexidade, começou com Leeuw *et al.*, 1970 e Gill, 1974. Este último prova algumas propriedades básicas de máquinas de Turing probabilísticas, como a enumerabilidade efetiva, a universalidade e outros resultados. Podemos associar a cada modelo de computação um modelo aumentado que permite elementos aleatórios, por exemplo, podemos modificar a classe de programas RAM introduzindo uma instrução especial chamada RANDOM(N) que atribui para a variável N um bit com igual probabilidade e de forma independente de chamadas anteriores. Como já dissemos antes, dentro das classes de tempo polinomial, os resultados relatados neste texto não dependem de máquina.

Uma **máquina de Turing probabilística** M é definida como uma máquina em que cada transição tem dois movimentos legais δ_0 e δ_1, entretanto, a cada transição durante uma computação só um deles é escolhido como o próximo passo; dado um estado q e um símbolo da fita σ a máquina executa a transição $\delta_i(q, \sigma)$ para uma escolha aleatória $i \in \{0, 1\}$. A resposta da máquina para uma entrada w é uma variável aleatória que depende de w; nesse caso, consideramos a distribuição da variável aleatória $\mathsf{M}(w)$ da máquina probabilística M com entrada fixa $w \in \Sigma^*$ em que a probabilidade é tomada sobre as escolhas aleatórias feitas por M. O espaço de probabilidades é formado por todas as sequências binárias que representam as escolhas internas da máquina numa computação desde a configuração inicial no estado q_0 até alcançar q_{para}. Para uma tal sequência de comprimento m associamos a proba-

bilidade 2^{-m}. Na computação de M com entrada w a probabilidade

$$\mathbb{P}[\, M(w) = z \,]$$

é a soma das probabilidades de todas as sequências de escolhas aleatórias internas que terminam com a palavra $z \in \Sigma^*$ escrita na fita.

Notemos que para uma uma entrada w fixa o número de bits aleatórios usados numa computação com w pode variar, entretanto, para toda sequência binária r que representa as escolhas aleatórias numa computação com w, não existe uma sequência r' que também representa as escolhas aleatórias numa computação e tal que r seja prefixo de r' ou vice-versa.

Uma máquina probabilística M **decide** L **com probabilidade de erro** ϵ **em tempo** T, para $T\colon \mathbb{N} \to \mathbb{N}$, se para toda palavra $w \in \Sigma^*$

$$\mathbb{P}[\, M(w) \neq L(w) \,] \leqslant \varepsilon$$

e $M(w)$ termina a computação em no máximo $T(|w|)$ transições para *qualquer que seja a sequência de bits aleatórios* usados na computação.

Uma alternativa à definição acima é dada a seguir pelo modelo chamado em algumas referências bibliográficas de máquina de Turing probabilística *off-line*: é uma máquina de Turing determinística que tem uma entrada auxiliar escrita numa fita auxiliar só de leitura, além da entrada principal escrita na fita principal. Essa fita auxiliar contém uma sequência B de bits aleatórios, cada posição tem um bit com distribuição uniforme e as ocorrências na sequência são independentes. Cada posição da fita auxiliar é usada (lida) no máximo uma única vez. Notemos que o comprimento da entrada auxiliar pode ser definido como igual ao tempo T de execução da máquina. Nesse caso, consideramos a distribuição da variável aleatória $M(w, B)$ da máquina probabilística M com entrada fixa $w \in \Sigma^*$ e um vetor aleatório $B \in_R \{0,1\}^{T(|w|)}$. Uma máquina probabilística *off-line* M com tempo de execução T **decide** L **com probabilidade de erro** ϵ se para toda palavra $w \in \Sigma^*$

$$\mathop{\mathbb{P}}_{B \in_R \{0,1\}^{T(|w|)}} [M(w, B) \neq L(w)] \leqslant \varepsilon.$$

Os dois modelos são computacionalmente equivalentes (Leeuw *et al.*, 1970). Uma computação da máquina probabilística *off-line* N como definida acima pode ser simulada por uma máquina probabilística M que com entrada w seleciona internamente uma sequência de bits aleatórios b e computa $N(w, b)$. Por outro lado, a computação de uma máquina probabilística M é simulada pela máquina determinística N que realiza transições de $M(w)$ usando uma sequência de bits aleatórios b dados com entrada auxiliar como as escolhas aleatórias internas a M. De fato, mais do que o

que foi dito vale, as definições são equivalentes no contexto do tempo de execução, assim, na sequência, vamos usar livremente o modelo probabilístico que for mais conveniente.

Em vista dessa definição, a classe NP pode ser definida (verifique) como a classe das linguagens L para as quais existe um polinômio p e um algoritmo probabilístico M de tempo polinomial tal que todo $w \in \{0,1\}^*$

1. se $w \in L$ então

$$\mathbb{P}_{B \in_R \{0,1\}^{p(|w|)}}[M(w,B) = L(w)] > 0,$$

   ou seja, se $w \in L$ então existe uma *entrada auxiliar* $B \in \{0,1\}^{p(|w|)}$ que faz M responder *sim*;

2. se $w \notin L$ então

$$\mathbb{P}_{B \in_R \{0,1\}^{p(|w|)}}[M(w,B) = L(w)] = 1$$

   nesse caso, se $w \notin L$, M responde *não* qualquer que seja a *entrada auxiliar*.

Classes de complexidade probabilísticas As seguir, vamos definir as classes de complexidade de tempo polinomial que estão associadas as algoritmos probabilísticos.

A classe RP — *Randomized Polynomial time* — é formada pelas linguagens L decididas por um algoritmo probabilístico M de tempo polinomial tal que para algum polinômio p e para todo $w \in \{0,1\}^*$, a probabilidade de erro é

$$\mathbb{P}_{B \in_R \{0,1\}^{p(|w|)}}[M(w,B) \neq L(w)] \leqslant \begin{cases} 1/3, & \text{se } w \in L \\ 0, & \text{se } w \notin L \end{cases}$$

ou seja, se $w \in L$ então a M erra com probabilidade limitada, e se $w \notin L$ então M não erra. Assim, uma resposta *sim* do algoritmo M está sempre certa, isto é $w \in L$, enquanto que uma resposta *não* pode ser um falso negativo, o que ocorre com probabilidade no máximo 1/3.

O problema de determinar se um gafo G tem um corte pequeno, de tamanho menor que um dado k, está em RP pois o Algoritmo 9, página 75, pode ser executado duas vezes o que limita a probabilidade de erro a 1/4, além disso se o algoritmo acha um corte pequeno responde sim (corretamente), se não acha (o que não significa que não tenha) então responde não (com chance de erro).

A constante 1/3 nos algoritmos que definem RP é arbitrária. Se realizamos k execuções independentes desse algoritmo (com a mesma entrada), então uma resposta 1 é definitiva enquanto que k respostas 0 estão todas erradas com probabilidade

menor que $(1/3)^k$ e se $k = O(n^c)$ para alguma constante positiva c então as execuções terminam em tempo polinomial. O mesmo vale se trocarmos 1/3 por qualquer constante $\varepsilon \in (0,1)$ de modo que se a probabilidade de erro for ε então podemos executar o algoritmo várias vezes (com a mesma entrada) até que a probabilidade de erro fique menor que 1/3, o que certifica a pertinência em RP.

Observemos que se M decide uma linguagem L que pertence a RP então as escolhas de bits aleatórios por $M(w)$ que terminam em $M(w) = 1$ é um certificado c para $w \in L$, portanto, podemos concluir que L pertence a NP.

Proposição 4.6 $RP \subseteq NP$. □

A classe coRP é a classe das linguagens L tais que $\overline{L} \in RP$, isto é, a classe das linguagens L para as quais existe um algoritmo probabilístico M de tempo polinomial tal que uma resposta *não* está sempre certa enquanto que uma resposta *sim* pode ser um falso positivo. Para algum polinômio p e para todo $w \in \{0,1\}^*$,

$$\mathop{\mathbb{P}}_{B \in_R \{0,1\}^{p(|w|)}} [M(w,B) \neq L(w)] \leqslant \begin{cases} 1/3, & \text{se } w \notin L \\ 0, & \text{se } w \in L \end{cases}$$

Nos algoritmos 11 e 13 (pág. 77 e seguintes) para, respectivamente, os testes de igualdade do produto de matrizes e de identidade polinomial, uma resposta *não* está sempre certa enquanto que uma resposta *sim* pode estar errada, ademais duas rodadas independentes de tais algoritmos reduzem a probabilidade de erro para o limiar da definição da classe coRP, logo as linguagens definidas por esses exemplos são linguagens em coRP. O problema de testar se um número é primo também está em coRP como atesta o algoritmo de Miller–Rabin (página 168), que também atesta que o problema de testar se um número é composto está em RP.

Um fato interessante ocorre quando consideramos a possibilidade de uma linguagem $L \in RP \cap coRP$, pois ela pode se beneficiar dos resultados exatos do algoritmo M_{RP}, o que prova que $L \in RP$, e do algoritmo M_{coRP}, o que prova que $L \in coRP$, para especificar um algoritmo probabilístico M' que nunca erra, porém uma escolha ruim de bits aleatórios pode levar o algoritmo a executar por muito tempo: dada uma entrada $w \in \{0,1\}^*$

```
1 enquanto verdadeiro faça
2 |   se M_RP(w) = 1 então responda 1;
3 |   se M_coRP(w) = 0 então responda 0.
```

Algoritmo 39: $RP \cap coRP$

Se M_{RP} responder 1 então por definição garantimos que $w \in L$. Analogamente, se M_{coRP} responder 0 então por definição garantimos que $w \notin L$. Assim nunca obtemos

uma resposta errada. Porém não temos como saber exatamente quantas rodadas vamos esperar até que um certificado apropriado seja encontrado.

Um algoritmo probabilístico tem **tempo esperado** $\mathsf{T}(n)$ se a variável aleatória t_w, que denota o tempo de execução do algoritmo com entrada w, tem valor esperado $\mathbb{E}\, t_w \leqslant \mathsf{T}(|w|)$ para todo w. A esperança é computada sobre os bits aleatórios usados na computação pelo algoritmo. Nos casos em que T é um polinômio dizemos que a máquina é de **tempo esperado polinomial**. Por exemplo, o algoritmo aproximativo para o problema MAX-E3SAT, Algoritmo 19 na página 134, tem tempo esperado polinomial.

No Algoritmo 39 acima, o número esperado de execuções até o passo 2 ter sucesso é no máximo 2 e o mesmo vale para o passo 3, portanto o tempo esperado de M' é no máximo um número constante de simulações de $\mathsf{M}_{\mathsf{RP}}(w)$ e de $\mathsf{M}_{\mathsf{coRP}}(w)$, ou seja, o tempo esperado de execução é polinomial em $|w|$.

A classe ZPP — *Zero-error Probabilistic Polynomial time* — é a classe de complexidade de todas as linguagens para as quais existe um algoritmo probabilístico M de tempo *esperado* polinomial e tal que $\mathbb{P}[\,\mathsf{M}(w,\mathsf{B}) = \mathsf{L}(w)\,] = 1$, para todo $w \in \{0,1\}^*$. Equivalentemente, ZPP é a classe das linguagens para as quais existe um algoritmo probabilístico de tempo polinomial M tal que, para todo w, $\mathsf{M}(w) \in \{0,1,\perp\}$ em que 0 significa $w \notin \mathsf{L}$ e 1 significa $w \in \mathsf{L}$, como é usual, e $\perp$ significa "não sei" e $\mathbb{P}[\,\mathsf{M}(w) = \perp\,] \leqslant 1/2$. Intuitivamente, entendemos essa definição como que em tempo polinomial ou o algoritmo decide ou interrompe a execução.

O Algoritmo 39 acima mostra que

$$\mathsf{RP} \cap \mathsf{coRP} \subseteq \mathsf{ZPP} \tag{4.2}$$

mas nesse caso vale a igualdade dessas classes como demonstra o seguinte resultado.

LEMA 4.7 $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$.

DEMONSTRAÇÃO. De (4.2), só precisamos provar que $\mathsf{ZPP} \subseteq \mathsf{RP} \cap \mathsf{coRP}$.

Seja $\mathsf{L} \in \mathsf{ZPP}$ uma linguagem e M um algoritmo probabilístico de tempo esperado $p(n)$ que decide pertinência em L sem errar. Para provar pertinência em RP, definimos um algoritmo N que computa da seguinte maneira: dada uma entrada $w \in \{0,1\}^*$

simula M com entrada w até no máximo $3p(|w|)$ passos;
se $\mathsf{M}(w) = 1$ **então responda** 1;
senão responda 0.

Algoritmo 40: $\mathsf{N}(w)$

Se $w \notin \mathsf{L}$ então N termina sem aceitar w, portanto $\mathbb{P}[\,\text{erro}\,] = \mathbb{P}[\,\mathsf{N}(w) = 1\,] = 0$. Por outro lado, se $w \in \mathsf{L}$ então o algoritmo N aceita w (corretamente) ou termina

pelo limite dos $3p(|w|)$ passos. No segundo caso $\mathsf{N}(w) = 0$ e a resposta está errada. A probabilidade da resposta errada é $\mathbb{P}[\, t_w > 3p(|w|)\,]$ em que, como acima, t_w é a variável aleatória para o tempo de execução do algoritmo M com entrada w. Pela desigualdade de Markov

$$\mathbb{P}[\, t_w \geqslant 3p(|w|)+1\,] \leqslant \frac{p(|w|)}{3p(|w|)+1} < \frac{1}{3}$$

portanto, se $w \in \mathsf{L}$ então $\mathbb{P}[\,\text{erro}\,] = \mathbb{P}(\,\mathsf{N}(w) = 0\,) < 1/3$ e com isso temos que $\mathsf{L} \in \mathsf{RP}$.

Do maneira análoga, podemos provar que $\mathsf{ZPP} \subseteq \mathsf{coRP}$; definimos um algoritmo N' que com entrada w

> simula M com entrada w até no máximo $3p(|w|)$ passos;
> **se** $\mathsf{M}(w) = 0$ **então responda** 0;
> **senão responda** 1.

Algoritmo 41: $\mathsf{N}'(w)$

Assim, se $w \in \mathsf{L}$ então $\mathbb{P}[\,\text{erro}\,] = \mathbb{P}[\,\mathsf{N}(w) = 0\,] = 0$. Se $w \notin \mathsf{L}$ então o algoritmo pode aceitar erroneamente e $\mathbb{P}[\,\text{erro}\,] = \mathbb{P}[\, t_w > 3p(|w|)\,] < 1/3$. Portanto, $\mathsf{ZPP} \subseteq \mathsf{RP} \cap \mathsf{coRP}$. □

BPP A classe BPP — *Bounded-error Probabilistic Polynomial time* — é a classe das linguagens decididas por algoritmos probabilísticos de tempo polinomial com probabilidade de erro 1/3, ou seja, $\mathsf{L} \in \mathsf{BPP}$ se existe um polinômio p e um algoritmo probabilístico M de tempo polinomial tal que para todo $w \in \{0,1\}^*$

$$\mathop{\mathbb{P}}_{\mathsf{B} \in_R \{0,1\}^{p(|w|)}} [\mathsf{M}(w, \mathsf{B}) \neq \mathsf{L}(w)] \leqslant 1/3.$$

A probabilidade de erro 1/3 na definição não tem nada de especial, qualquer constante $\varepsilon \in (0, 1/2)$ serviria para definir a mesma classe de linguagens, como veremos no Lema 4.8 abaixo a probabilidade de erro pode ser feita menor que qualquer constante. Se M é uma máquina probabilística de tempo polinomial que aceita a linguagem L com probabilidade de erro ε, então podemos escrever uma máquina N probabilística e de tempo polinomial que aceita a mesma linguagem L com probabilidade de erro 1/3.

Lema 4.8 *Para toda constante $0 < \varepsilon < 1/2$, todo polinômio p, toda linguagem $\mathsf{L} \subseteq \{0,1\}^*$ e todo algoritmo probabilístico* M *de tempo polinomial que decide* L *com probabilidade de erro ε, existe um algoritmo probabilístico* N *de tempo polinomial e que decide* L *com probabilidade de erro $2^{-p(n)}$ nas entradas de tamanho n, para todo n.*

Demonstração. Sejam ε, p, L e M como no enunciado. Definimos

$$\delta := 4\varepsilon(1-\varepsilon) \;\text{ e }\; k(n) := \left\lceil \frac{p(n)}{\log_2(1/\delta)} \right\rceil$$

e notemos que $\delta < 1$ pois $\varepsilon < 1/2$.

Consideremos o algoritmo N que com entrada $w \in \{0,1\}^*$ simula $M(w)$ um número ímpar de vezes e decide pela resposta dada na maioria das simulações:

```
1 para i de 1 até 2k+1 faça
2 |   m_i ← M(w);
3 se ∑_i m_i > k então responda 1;
4 senão responda 0.
```

Certamente, N é uma algoritmo probabilístico de tempo polinomial. Seja $S = S(w) \in \{0,1\}^{2k+1}$ uma sequência de respostas dadas na linha 2 em uma execução de N com entrada w. Sejam $c = c(S)$ e $e = e(S)$ o número de respostas certas e respostas erradas, respectivamente, em S, onde resposta certa quer dizer que M decidiu corretamente a pertinência de w em L.

O algoritmo N responde errado se para a sequência S correspondente a uma execução de N ocorre $c < e$. Queremos limitar a probabilidade desse evento. Como M tem probabilidade de erro limitada por $\varepsilon < 1/2$, a probabilidade de uma sequência S que faz N responder errado é no máximo $\varepsilon^e(1-\varepsilon)^c \leqslant \varepsilon^{k+1}(1-\varepsilon)^k$ pois $e \geqslant k+1$ e $\varepsilon < 1-\varepsilon$. Assim, a probabilidade de erro é

$$\mathbb{P}[N(w) \neq L(w)] \leqslant \sum_S \varepsilon^{e(S)}(1-\varepsilon)^{c(S)} \leqslant 2^{2k+1}\varepsilon^{k+1}(1-\varepsilon)^k = 2\varepsilon(2^2\varepsilon(1-\varepsilon))^k < (4\varepsilon(1-\varepsilon))^k = \delta^k$$

em que a soma é sobre toda sequência S de respostas de M que faz N responder errado. Pela escolha de k e de $\log_2(1/\delta) = 1/\log_\delta(1/2)$ temos $\delta^k \leqslant 2^{-p(|w|)}$. □

Da definição das classes deduzimos que

$$\mathsf{RP}, \mathsf{coRP}, \mathsf{ZPP} \subseteq \mathsf{BPP}.$$

Exemplo 4.9 (PIT∈BPP). Seja $p \in \mathbb{F}[x_1, \ldots, x_n]$ um polinômio dado por um circuito aritmético, como definido na página 204. O PIT é o problema da identidade polinomial: decidir se o polinômio p definido pelo circuito $\langle p \rangle$ é identicamente nulo.

Comentamos na seção 1.7.3 que é um problema importante na teoria da computação sabermos se é possível resolvê-lo em tempo polinomial no tamanho do circuito. Também comentamos que, a princípio, temos dois problemas computacionais: dado um polinômio p, EZE – *Evaluates to Zero Everywhere* – é o problema de decidir se, como função, p vale zero em todo elemento do corpo e PIT que é decidir se p na forma canônica tem todos os coeficientes nulos. EZE está em coNP, pois está em coRP como mostra o algoritmo 13. De fato é coNP-difícil, pois é possível escrever uma fórmula 3-CNF como um polinômio sobre $\mathbb{F}_2$.

Um circuito aritmético de tamanho m tem profundidade no máximo m, portanto realiza no máximo m multiplicações, logo define um polinômio de grau no máximo 2^m. Nessa situação, o Algoritmo 13, página 82, sorteia n valores em $\{1,\ldots,2^{m+2}\}$, avalia o valor de p com esses valores simulando o circuito aritmético em tempo polinomial em m e resolve PIT com probabilidade de erro 1/4. Porém, a sequência sorteada $(x_1,\ldots,x_n)$ tem tamanho $O(nm)$ bits enquanto que x^{2^m} calculado em 2^{m+2} resulta num número com $O(m2^m)$ bits, logo só para escrevê-lo o tempo consumido seria exponencialmente grande, o que não resulta num algoritmo de tempo polinomial para o problema.

Um modo de contornarmos esse problema é calcularmos as operações aritméticas nas portas do circuito módulo um inteiro positivo k apropriado, resultando, no final do cômputo, $p(x_1,\ldots,x_n) \bmod k$. Com isso o tempo de simulação do circuito continua polinomial e os números que ocorrem nas operações têm tamanho controlado, mas aumenta a probabilidade de erro pois podemos ter $p(x_1,\ldots,x_n) \neq 0$ e $p(x_1,\ldots,x_n) \bmod k = 0$, caso k divida $p(x_1,\ldots,x_n)$. Agora, devemos estimar essa probabilidade de erro e fazê-la pequena usando rodadas independentes de escolhas para k.

Tomemos $k \in \{1,2,\ldots,2^{2m}\}$. A quantidade de números primos nesse conjunto é, pelo Teorema dos Números Primos, maior que

$$\frac{2^{2m}}{2m+2}$$

se $m > 2$ (Rosser, 1941).

Assumamos que $p := p(x_1,\ldots,x_n) \neq 0$. A quantidade de fatores primos distintos em p é $(m+2)2^m$, pois se $p = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_t^{\alpha_t}$ então $p \geqslant 2^t$, portanto $t \leqslant \log_2 p \leqslant \log_2 (2^{m+2})^{2^m} = (m+2)2^m$. Assim, a quantidade de primos em $\{1,2,\ldots,2^{2m}\}$ que não é um dos t fatores primos de p é maior que

$$\frac{2^{2m}}{2m+2} - (m+2)2^m > \frac{2^{2m}}{8m}$$

para todo $m > 7$, de modo que a probabilidade com que uma escolha aleatória em $\{1,2,\ldots,2^{2m}\}$ resulte num número *bom*, isto é um primo que não divide m, é maior que $1/8m$. Em r sorteios para o valor de k, basta um deles resultar num número *bom* para podermos responder que $p(x_1,\ldots,x_n) \neq 0$. A probabilidade com que nenhum dentre $r := 16m$ sorteios resulte num número *bom* é no máximo

$$\left(1 - \frac{1}{8m}\right)^r = \left(\left(1 - \frac{1}{8m}\right)^{-8m}\right)^{-2} < 0{,}15$$

para todo inteiro $m > 0$, pois $(1 - 1/8m)^{-8m} \geqslant \mathrm{e}$ (veja (s.8)). Portanto, se $\langle p \rangle$ é nulo o algoritmo sempre descobre, caso contrário o polinômio não nulo é declarado nulo com probabilidade 0,15, logo PIT $\in$ BPP. $\Diamond$

Notemos que $\mathsf{P} \subseteq \mathsf{BPP}$ pois para todo algoritmo de tempo polinomial M podemos escrever um algoritmo probabilístico de tempo polinomial M' que simula M e ignora os bits aleatórios. Não sabemos se a recíproca vale ou não

Problema 8. $\mathsf{BPP} \subseteq \mathsf{P}$?

Também não sabemos situar BPP com relação a NP

Problema 9. $\mathsf{BPP} \subseteq \mathsf{NP}$?

não sabemos se $\mathsf{BPP} \subseteq \mathsf{NP}$, se $\mathsf{NP} \subseteq \mathsf{BPP}$ ou se nenhuma dessas duas relações valem. Também, não conhecemos nenhuma linguagem que é completa para BPP. Agora, é um exercício fácil, que deixamos para o leitor, verificar que $\mathsf{BPP} = \mathsf{coBPP}$.

Problema 10. Alguma(s) das inclusões $\mathsf{P} \subseteq \mathsf{ZPP} \subseteq \mathsf{RP} \subseteq \mathsf{BPP}$ são próprias?

Os algoritmos que reconhecem linguagens em BPP são chamados, em algumas referências bibliográficas, de **Atlantic city**, os de RP e coRP são conhecidos como **Monte-Carlo** e os de ZPP são os **Las Vegas**.

Finalizamos essa seção enunciado o seguinte resultado sem prova.

Teorema. (Impagliazzo e Wigderson, 1999) *Se existe uma linguagem* L *decidida em tempo* $2^{O(n)}$ *que nas palavras de tamanho* n *requer circuito de tamanho* 2^{cn}, *para algum* $c > 0$ *e todo* n, *então* $\mathsf{P} = \mathsf{BPP}$.

Exercício 4.10. Prove que se $L \in \mathsf{BPP}$, então existe um algoritmo probabilístico que com entrada w decide se $w \in L$ com probabilidade de erro menor que $1/(3m)$ e em tempo polinomial em $|w|$, em que $m = m(|w|)$ é o número (polinomial) de bits aleatórios usados na computação (dica: ajuste a quantidade de rodadas independentes de simulações no algoritmo acima).

$\mathsf{BPP} \subseteq \mathsf{P/poly}$ O próximo resultado mostra que toda linguagem decidida por algoritmo probabilístico de tempo polinomial também é decidida por circuitos de tamanho polinomial. Como circuito é um modelo não-uniforme de computação e como há muitas sequências binárias aleatórias que testemunham a favor da decisão correta, podemos escolher uma tal sequência para cada n e todo $w \in \{0,1\}^n$ e projetar os circuitos com as sequências escolhidas. Essa ideia está formalizada no teorema a seguir. A inclusão é própria porque há problemas não decidíveis por algoritmos que são decididos por família de circuitos de tamanho polinomial, como já dissemos acima (Exercício 4.20, página 238).

Teorema 4.11 (Adleman, 1978) $\mathsf{BPP} \subsetneq \mathsf{P/poly}$.

Demonstração. Sejam $L \in \mathsf{BPP}$ e M um algoritmo probabilístico de tempo polinomial que decide L com probabilidade de erro exponencialmente pequena

$$\mathop{\mathbb{P}}_{B \in_R \{0,1\}^{p(n)}}[M(w,B) \neq L(w)] < 2^{-n}$$

nas entradas de tamanho n, para algum polinômio p. A existência desse algoritmo é garantido pelo Lema 4.8 acima.

Fixado um inteiro positivo n, podemos afirmar que existe uma sequência binária b que é um certificado de pertinência em L para toda entrada w de tamanho n. De fato,

$$\mathop{\mathbb{P}}_{B \in_R \{0,1\}^{p(n)}}\left[\bigcup_{w \in \{0,1\}^n} M(w,B) \neq L(w)\right] < \sum_{w \in \{0,1\}^n} 2^{-n} = 1$$

logo, com probabilidade maior que 0 para pelo menos um $B \in \{0,1\}^{p(n)}$ fixo o algoritmo M responde corretamente para todo $w \in \{0,1\}^n$, ou seja, existe (ao menos) uma sequência $b_n \in \{0,1\}^{p(n)}$, com a qual M não erra nas entradas de tamanho n. O algoritmo (determinístico) M' dado por

$$M'(w) := M(w, b_n)$$

não erra em entradas de tamanho n.

A partir de M' construímos, para cada n, um circuito booleano C_n de tamanho polinomial, conforme construção do Exercício 4.4 (página 204), tal que $C_n(w) = M'(w)$, para todo w de tamanho n. Dessa forma, a família de circuitos $(C_n)_{n>0}$ decide L, portanto $L \in \mathsf{P/poly}$. □

4.2.2 BPP ESTÁ NA HIERARQUIA POLINOMIAL

A hierarquia polinomial é uma hierarquia formada por classes de problemas com complexidade polinomial, as *classes do nível i* são denotadas por Σ_i e Π_i, para todo $i \in \mathbb{N}$. Essas classes generalizam P e NP de um certo modo natural. Uma motivação para essa hierarquia pode ser lida no capítulo 17 de Papadimitriou (1994), assim como outros resultados que estão fora do escopo deste texto pois precisam de vários pré-requisitos da Complexidade Computacional. Nesta seção, falaremos brevemente sobre a hierarquia polinomial, sua relação com as classes probabilísticas e daremos alguns outros resultados sem prova para que o leitor possa ter alguma referência sobre a importância de PH e sua relação com outras classes de complexidade.

Por enquanto vamos nos concentrar nos primeiros níveis da hierarquia e em seguida mostrar uma relação com a classe BPP. Começamos declarando que

$$\Sigma_0 = \Pi_0 := \mathsf{P} \quad \text{e} \quad \Sigma_1 := \mathsf{NP} \quad \text{e} \quad \Pi_1 := \mathsf{coNP}$$