Algoritmos Probabilísticos

Jair Donadelli

"Numbers that fool the Fermat test are called Carmichael numbers, and little is known about them other than that they are extremely rare. There are 255 Carmichael numbers below 100,000,000. The smallest few are 561, 1105, 1729, 2465, 2821, and 6601. In testing primality of very large numbers chosen at random, the chance of stumbling upon a value that fools the Fermat test is less than the chance that cosmic radiation will cause the computer to make an error in carrying out a "correct" algorithm. Considering an algorithm to be inadequate for the first reason but not for the second illustrates the difference between mathematics and engineering." Abelson & Sussman

[*Rosencrantz and Guildenstern are riding horses down a path - they pause*]
R: Umm, uh...
[*Guildenstern rides away, and Rosencrantz follows. Rosencrantz spots a gold coin on the ground*]
R: Whoa - whoa, whoa.
[*Gets off horse and starts flipping the coin*] **R**: Hmmm. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads. Heads.
[*Guildenstern grabs the coin, checks both sides, then tosses it back to Rosencrantz*]
R: Heads.
[*Guildenstern pulls a coin out of his own pocket and flips it*]
R: Bet? Heads I win?
[*Guildenstern looks at coin and tosses it to Rosencrantz*]
R: Again? Heads.
[...]
R: Heads
G: A weaker man might be moved to re-examine his faith, if in nothing else at least in the law of probability.
R: Heads
G: Consider. One, probability is a factor which operates within natural forces. Two, probability is not operating as a factor. Three, we are now held within um...sub or supernatural forces. Discuss!
R: What?

[...]
R: Heads, getting a bit of a bore, isn't it?
[...]
R: 78 in a row. A new record, I imagine.
G: Is that what you imagine? A new record?
R: Well...
G: No questions? Not a flicker of doubt?
R: I could be wrong.
G: No fear?
R: Fear?
G: Fear!

R: Seventy nine.

[...]

G: I don't suppose either of us was more than a couple of gold pieces up or down. I hope that doesn't sound surprising because its very unsurprisingness is something I am trying to keep hold of. The equanimity of your average tosser of coins depends upon a law, or rather a tendency, or let us say a probability, or at any rate a mathematically calculable chance, which ensures that he will not upset himself by losing too much nor upset his opponent by winning too often. This made for a kind of harmony and a kind of confidence. It related the fortuitous and the ordained into a reassuring union which we recognized as nature. The sun came up about as often as it went down, in the long run, and a coin showed heads about as often as it showed tails.

Tom Stoppard, *Rosenkrantz and Guildenstern are dead* (1996).

SUMÁRIO

Apresentação

Algoritmos aleatorizados são usados há muito tempo em disciplinas da Estatística e Física, para amostragem (*sampling*) e simulação (métodos de Monte Carlo[1]) e foram introduzidos na Computação na década de 70 com o objetivo de projetar algoritmos eficientes; os resultados pioneiros foram os algoritmos de Miller (1975), Rabin (1980a) e Solovay and Strassen (1977) para testar primalidade. Estes algoritmos utilizam uma fonte de bits aleatórios que gera bits com probabilidade uniforme em tempo constante. Desde então, o número de aplicações desses algoritmos vem crescendo. Atualmente são conhecidas soluções aleatorizadas para problemas que não admitem solução determinística, como compartilhamento de recursos em sistemas distribuídos. A aleatorização é figura central em alguma áreas, como a Criptografia e gerou um sub-área rica da Complexidade Computacional com resultados fascinantes.

O objetivo deste texto é exibir alguns desses resultados em Computação.

Um dos atrativos dos algopritmos aleatorizados reside em que vários problemas computacionais admitem algoritmo aleatorizado simples e eficiente enquanto que algoritmos determinísticos de tempo subexponencial não são conhecidos. Por exemplo

Dado	: p, q polinômios de grau $\leqslant d$ sobre $\mathbb{Q}[x_1, x_2, \ldots, x_n]$.
Devolve:	decidir se $p \equiv q$.

Problema da identidade entre polinômios.

onde $\equiv$ significa que quando os polinômios são escritos como combinação linear de monômios os coeficientes são os mesmos; esse, por exemplo, é o caso quando precisamos decidir se

$$\sum_{\sigma \in \mathbb{S}_n} \text{sinal}\left(\prod_{1 \leqslant i < j \leqslant n} (\sigma(j) - \sigma(i))\right) \prod_{i=1}^{n} x_i{}^{\sigma(i)-1} \equiv \prod_{1 \leqslant i < j \leqslant n} (x_j - x_i) \; ? \tag{1}$$

onde $\mathbb{S}_n$ é o grupo das permutações de $\{1, 2, \ldots, n\}$ (A identidade da equação (1) é verdadeira, o lado esquerdo é o determinante da matriz de Vandermonde.)

Esse é um problema para o qual não se conhece algoritmo determinístico de tempo subexponencial no tamanho da entrada e uma solução probabilística de

[1] O estudo sistemático desse método ganhou força com a invenção do computador, sendo que as primeiras simulações foram feitas pelos fdp do Projeto Manhattan que construiu a primeira bomba atômica.

tempo polinomial é, para d e para um subconjunto finito $S \subseteq \mathbb{Q}$ fixos,

Algoritmo 2: Identidade entre polinômios.

Dado : $f \in \mathbb{Q}[x_1, \ldots, x_n]$ de grau no máximo d.

Devolve: *sim* se $f \equiv 0$ e *não* caso contrário.

1 escolha $(x_1, x_2, \ldots, x_n)$ em S^n uniformemente;
2 **se** $f(x_1, x_2, \ldots, x_n) = 0$ **então devolva** *sim*;
3 **senão devolva** *não*.

Note que se $f \equiv 0$ então o algoritmo sempre responde *sim*. Por outro lado, mesmo se $f \not\equiv 0$ pode ocorrer que resposta seja *sim*. Quão freqüente é isso? A resposta é não muito, no máximo 1 em cada 3 vezes: *Seja* $f \in \mathbb{Q}[x_1, \ldots, x_n]$ *um polinômio de grau* $d > 0$, $f \not\equiv 0$. *Dado* $(x_1, \ldots, x_n) \in \{1, 2, \ldots, 3d\}^n$ *escolhido com distribuição uniforme, temos que a probabilidade de* $f(x_1, \ldots, x_n) = 0$ *é no máximo* $1/3$. Isso é suficientemente confiável? Se não for, basta notarmos que se repetirmos k vezes o algoritmo acima e em todas obtemos a resposta *sim* então a probabilidade da resposta estar errada é no máximo $(1/3)^k$; a repetição 15 vezes resulta em chance de erro de $0,00000007$, aproximadamente[2]

Essa característica de aliar confiança e eficiência é determinante no estabelecimento da técnica probabilística.

Outro exemplo é o problema de decidir se um número é primo. O teste de primalidade de Miller–Rabin é um algoritmo que recebe um inteiro e responde *primo* ou *composto*. A resposta *composto* sempre é certa. A resposta *primo*, por outro lado, pode estar errada, mas a probabilidade desse erro é controlada. Pode-se tornar a probabilidade da resposta errada arbitrariamente pequena às custas do aumento no tempo de execução. A probabilidade de erro é tão pequena e o algoritmo tão eficiente[3] que mesmo após a descoberta de um algoritmo determinístico de tempo polinomial para o problema 30 anos depois[4], esse teste desenvolvido na década de 70 continua sendo amplamente utilizado.

Algoritmos aleatorizados, como nos exemplos acima, com tempo de execução que depende somente da entrada e não dos sorteios e que podem responder errado, são conhecidos na literatura como algoritmos *Monte Carlo*. A complexidade de tempo usada para medir o desempenho é a de pior caso.

[2] Isso é dez vezes menos provável do que ter sido atingido por um raio no Brasil em 2003.

[3] A complexidade de pior caso para decidir primalidade de n em k rodadas é $O(k \log^3 n)$ e a probabilidade de erro é $< (1/4)^k$.

[4] Agrawal, Kayal e Saxena publicaram em 2004 um algoritmo determinístico com complexidade de pior caso $O(\log^{12} n)$ Agrawal, Kayal, and Saxena (2004).

Outra modalidade de algoritmos aleatorizados são os chamados de *Las Vegas*. Esses algoritmos tem probabilidade pequena de executarem por muito tempo, ou seja, a complexidade de tempo depende dos sorteios feitos durante a execução. Em outras palavras, a complexidade é uma variável aleatória e, idealmente, o algoritmo é eficiente em média com alta probabilidade. Nos algoritmos Las Vegas, a resposta sempre está correta.

Um exemplo de algoritmo dessa classe é o *Quicksort* aleatorizado que com probabilidade maior que $1 - 1/n^{-6}$ ordena n elementos em tempo $O(n \log n)$.

Algoritmo 3: *Quicksort* aleatorizado

Dado : uma seqüência de números $S = (x_1, x_2, \ldots, x_n)$.

Devolve: os elementos de S em ordem crescente.

```
1 se |S| = 1 então devolva S
2 senão
3     escolha um pivô x ∈ S uniformemente;
4     para cada y ∈ S, y ≠ x faça
5         se y < x então insira y em S1
6         senão insira y em S2;
7     ordene, recursivamente, S1 e S2;
8     devolva (S1, x, S2).
```

Do ponto de vista teórico, o algoritmo para identidade polinomial pode ser desaleatorizado e o modo mais trivial de fazer isso é rodar o algoritmo para todas os possíveis números aleatórios. Obviamente isso resulta num algoritmo exponencial. Não se sabe se desaleatorização de algoritmos eficientes pode ser feita de modo eficiente (veja Kabanets and Impagliazzo (2004)). Em outras palavras, não se sabe responder

$$\text{P=BPP?}$$

onde **P** é a classe do problemas computacionais que podem ser decididos por um algoritmo determinístico de tempo polinomial e **BPP** é a classe dos problemas computacionais que podem ser decididos por algoritmo aleatorizado de tempo polinomial com probabilidade de erro menor que $1/3$.

Um exemplo de desaleatorização eficiente é o algoritmo AKS para teste de primalidade Agrawal et al. (2004), ele é resultado de uma desaleatorização do algoritmo Agrawal–Biswas Agrawal and Biswas (2003), ou seja, o AKS é a desaleatorização

de um teste probabilístico de identidade polinomial, entretanto, essa desaleatorização só funciona no caso específico, uma técnica geral de desaleatorização de identidades polinomiais teria grande impacto na Teoria da Computação. Por outro lado, se existe f que pode ser computado por um algoritmo de tempo de pior caso $2^{O(n)}$ e tal que todo circuito booleano que computa f tem pelo menos $2^{\Omega(n)}$ portas lógicas então P=BPP, ou seja, se existem funções que são genuinamente difíceis de computar então desaleatorização é fácil Kabanets and Impagliazzo (2004).

Notação assintótica

Sejam f_n e g_n seqüências de números reais, onde $f_n > 0$ para todo n suficientemente grande. Usaremos as seguintes notações para o comportamento assintótico dessas seqüências:

- $g_n = O(f_n)$, quando $n \to \infty$, se existem constantes positivas $c \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $|g_n| \leqslant cf_n$, para todo $n \geqslant n_0$;
- $g_n = \Omega(f_n)$, quando $n \to \infty$, se existem constantes positivas $C \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $g_n \geqslant Cf_n$, para todo $n \geqslant n_0$;
- $g_n = \Theta(f_n)$, quando $n \to \infty$, se existem constantes positivas c, $C \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $Cf_n \leqslant g_n \leqslant cf_n$, para todo $n \geqslant n_0$;
- $g_n = o(f_n)$ se $g_n/f_n \to 0$, quando $n \to \infty$.

Fatos e notações utilizados

- Se $A \subseteq B$ então $\overline{A}$ denota o conjunto $B \setminus A$.
- se $|x| < 1$ então $\sum_{n\geqslant 0} x^{-n} = (1-x)^{-1}$.
- se $|x| < 1$ então $\sum_{n\geqslant k} x^{-n} = x^k(1-x)^{-1}$.
- se $|x| < 1$ então $\sum_{n\geqslant 1} nx^{-n} = x(1-x)^{-2}$.
- $\sum_{n\geqslant 1} n^{-2} = \pi/6$.
- $\left(\frac{n}{k}\right)^k \leqslant \binom{n}{k} \leqslant \left(e\frac{n}{k}\right)^k$
- $(1-x) \leqslant e^{-x}$.
- GRAFOS

CAPÍTULO 1

Introdução à probabilidade discreta

Conteúdo

1.1 Probabilidade discreta

Neste texto consideraremos triplas $(\Omega, \mathcal{E}, \mathbb{P})$ que definem um **espaço de probabilidade discreto**, ou seja,

(i) Ω é um conjunto enumerável, chamado **espaço amostral**;

(ii) $\mathcal{E} = 2^{\Omega}$ é a σ-álgebra de eventos,

(iii) $\mathbb{P}\colon \mathcal{E} \to \mathbb{R}$ é tal que

(a) $0 \leqslant \mathbb{P}(E) \leqslant 1$ para todo **evento** $E \in \mathcal{E}$;

(b) $\mathbb{P}(\Omega) = 1$;

(c) para qualquer seqüência enumerável $(E_i)_{i\geqslant 1}$ de eventos disjuntos, mutuamente, vale

$$\mathbb{P}\left(\bigcup_{i\geqslant 1} E_i\right) = \sum_{i\geqslant 1} \mathbb{P}(E_i). \tag{1.1}$$

Observação 1. O lado esquerdo da equação (1.1) não depende de uma enumeração particular dos conjuntos e um teorema do cálculo (veja Bartle (1976)) garante o mesmo para o lado direito, ou seja, que se uma série de termos não-negativos converge então qualquer rearranjo dos termos resulta numa série que converge para o mesmo valor.

Como $\mathcal{E} = 2^{\Omega} = \{E\colon E \subseteq \Omega\}$ está fixo por todo o texto, escrevemos somente $(\Omega, \mathbb{P})$. No caso de espaços discretos a função de probabilidade $\mathbb{P}$ fica definida pelos valores de $\mathbb{P}(\omega)$ para todo $\omega \in \Omega$ pois decorre de (1.1) que

$$P(E) = \sum_{a\in E} P(a)$$

para todo $A \subseteq \Omega$. Com isso, podemos definir $\mathbb{P}\colon \Omega \to [0,1]$ e, usualmente, não escrevemos as chaves nos eventos unitários, por exemplo, no exemplo 2 a seguir usamos $\mathbb{P}(\text{coroa})$ para $\mathbb{P}(\{\text{coroa}\})$.

Exemplo 2 (lançamento de moeda equilibrada). Quando nos referimos ao lançamento de uma moeda equilibrada, estamos considerando o espaço de probabilidade dado por $\Omega = \{\text{cara}, \text{coroa}\}$, com $\mathcal{E} = \{\varnothing, \{\text{cara}\}, \{\text{coroa}\}, \Omega\}$ e $\mathbb{P}(\varnothing) = 0$, $\mathbb{P}(\text{cara}) = \mathbb{P}(\text{coroa}) = 1/2, \mathbb{P}(\Omega) = 1$.

Exemplo 3. Suponha que uma moeda equilibrada é lançada até sair cara; o espaço amostral que representa esse experimento é o conjunto das seqüencias $\omega_n = (c_1, c_2, \ldots, c_n)$ tal que para cada $n \geqslant 1$ temos $c_n = \text{cara}$ e $c_i = \text{coroa}$ para $1 \leqslant i < n$, logo $\Omega = \{\omega_n\colon n \geqslant 1\}$. Esse conjunto é claramente enumerável, a enumeração é dada pelo número de lançamentos; uma função de probabilidade é

$$\mathbb{P}(\omega_n) = \left(\frac{1}{2}\right)^n$$

que de fato define um espaço de probabilidade pois $0 < \mathbb{P}(\omega_n) < 1$ para todo ω_n e

$$\sum_{\omega\in\Omega} \mathbb{P}(\omega) = \sum_{i\geqslant 1} 2^{-i} = 1.$$

Se $a \in \Omega$, com Ω finito, e $\mathbb{P}(a) = 1/|\Omega|$ então escrevemos

$$a \in_R \Omega$$

e dizemos que a é uma **escolha aleatoria** em Ω. Nos algoritmos, nós vamos assumir a possibilidade de se fazer escolhas aleatórias, ou seja, assumir que os algoritmos dispõem de uma fonte de bits aleatórios com a escolha de cada bit com custo constante, isto é, em tempo $O(1)$ tem-se $a \in_R \{0,1\}$. Nos algoritmos usaremos a instrução

$$a \leftarrow_R \{0,1\}$$

para indicar que a variável a recebe o resultado de uma escolha aleatória em $\{0,1\}$.

Seja $\mathcal{P}$ a descrição de un evento $P \subseteq \Omega$, então as vezes escreveremos

$$\mathbb{P}_{\omega\in\Omega}[\ \mathcal{P}\ ] = \mathbb{P}\big(\{\omega \in \Omega\colon \omega \text{ satisfaz a propriedade } \mathcal{P}\}\big) \tag{1.2}$$

para $\mathbb{P}(P)$ e evidenciar o espaço amostral, por exemplo, quando consideramos a probabilidade de um determinado algoritmo responder errado, escrevemos $\mathbb{P}_{\omega\in_R\{0,1\}^*}[\text{erro}]$ para a probabilidade de $P = \{\omega \in \Omega\colon \omega$ é uma seqüência de bits que faz com que o algoritmo responda errado$\}$.

Exercício 4. Prove que $\mathbb{P}(E_1 \cup E_2) = \mathbb{P}(E_1) + \mathbb{P}(E_2) - \mathbb{P}(E_1 \cap E_2)$, para quaisquer eventos E_1 e E_2 num espaço de probabilidade discreto.

Exercício 5. Prove que para qualquer seqüência enumerável $(E_i)_{i\geqslant 1}$ de eventos num espaço discreto vale

$$\mathbb{P}\left(\bigcup_{i\geqslant 1} E_i\right) \leqslant \sum_{i>1} \mathbb{P}(E_i).$$

1.1.1 Identidade polinomial

Consideremos o problema de decidir se $f \equiv 0$, onde $f \in \mathbb{Q}[x]$ é um polinômio não-nulo de grau no máximo $d > 0$. O espaço amostral da escolha aleatória na linha 1 do algoritmo 2 é S com probabilidade $1/|S|$. Agora, consideremos o evento

$$E = E(f) = \{r \in S\colon r \text{ é raiz de } f\}.$$

A probabilidade do algoritmo 2 falhar é igual a $\mathbb{P}(E) = |E|/|S|$, portanto,

$$\mathbb{P}_{\omega\in_R S}[\ \text{erro}\ ] = \mathbb{P}(E) \leqslant \text{grau}(f)/|S|$$

pois f tem no máximo $\text{grau}(f)$ raízes em $\mathbb{Q}$, pelo teorema fundamental da álgebra.

Lema 6. *Seja* $f \in \mathbb{Q}[x]$ *um polinômio de grau no máximo* d, $f \not\equiv 0$. *Então*

$$\mathbb{P}_{x \in_R S}[\, f(x) = 0 \,] \leqslant d/|S|.$$

Corolário 7. *Se* $S = \{1, 2, \ldots, 3d\}$ *então o algoritmo 2, página 10, erra com probabilidade no máximo* $1/3$ *no caso de polinômios de uma variável.* □

1.2 Probabilidade condicional e Independência

Num espaço $(\Omega, \mathbb{P})$, a **probabilidade condicional** do evento E dado o evento A com $\mathbb{P}(A) > 0$ é definida por

$$\mathbb{P}(E|A) = \frac{\mathbb{P}(A \cap E)}{\mathbb{P}(A)}. \tag{1.3}$$

Exemplo 8. Dois dados, um azul e outro verde, são lançados e cada uma das seis faces são eqüiprováveis nos dois dados (dizemos que os dados são equilibrados). Qual é a probabilidade do dado verde ter resultado 6 dado que a soma dos resultados foi 8? O espaço amostral tem 36 elementos eqüiprováveis, formado pelos pares ordenados $(\text{azul}, \text{verde})$ dos resultados possíveis. O evento "a soma é 8" é $A = \{(2,6), (3,5), (4,4), (5,3), (6,2)\}$ logo a probabilidade é $1/5$. De outro modo, $E = \{(x,6) : 1 \leqslant x \leqslant 6,\ x \in \mathbb{N}\}$ e

$$\mathbb{P}(E|A) = \frac{\mathbb{P}(A \cap E)}{\mathbb{P}(A)} = \frac{\mathbb{P}\left(\{(2,6)\}\right)}{\mathbb{P}(A)} = \frac{1/36}{5/36} = \frac{1}{5}.$$

Exercício 9. Prove que $(A, \mathbb{P}_A)$, onde $\mathbb{P}_A(E) = \mathbb{P}(E|A)$ é um espaço de probabilidade.

Exemplo 10. Numa cômoda há três gavetas e em cada gaveta um par de meias. Na primeira gaveta há um par de meias brancas, na segunda um par de meias pretas e na gaveta que resta um par com um pé de cada cor, preta e branca.

Uma gaveta é escolhida aleatoriamente e, sem olhar para o interior da gaveta, um pé de meia é escolhido aleatoriamente e a gaveta é fechada. O pé de meia retirado é branco. Qual a probabilidade de o outro pé que ficou sozinho na gaveta ser preto?

Consideramos o espaço amostral formado pelos pares $(\text{gaveta}, \text{cor da meia retirada}) \in \{1,2,3\} \times \{\text{branca}, \text{preta}\}$ com as probabilidades de cada par dadas por

	branca	preta
1	1/3	0
2	0	1/3
3	1/6	1/6

Se A é o evento "retirou uma meia branca", $A = \{(1, \text{branca}), (3, \text{branca})\}$ e E o evento "ficou uma meia preta", $E = \{(2, \text{preta}), (3, \text{branca})\}$, então

$$\mathbb{P}(E|A) = \frac{\mathbb{P}(A \cap E)}{\mathbb{P}(A)} = \frac{1/6}{1/3 + 1/6} = \frac{1}{3}.$$

Exercício 11 (**Lei das probabilidades totais**). Prove que se $E_1, \ldots, E_m$ é uma partição de Ω então para todo evento B

$$\mathbb{P}(B) = \sum_{i=1}^{m} \mathbb{P}(B \cap E_i) = \sum_{i=1}^{m} \mathbb{P}(B|E_i)\mathbb{P}(E_i). \tag{1.4}$$

Exemplo 12 (Monty Hall). Esse exemplo é baseado num programa de um canal de televisão norte-americano chamado "*let's make a deal*" apresentado por Monty Hall, daí vem o nome do problema. Esse problema é bastante conhecido e já causou muita controvérsia.

Em um programa de auditório, um convidado deve escolher uma dentre três portas. Atrás de uma das portas há um carro e atrás de cada uma das outras duas há um bode. O convidado ganhará como prêmio o objeto que estiver atrás da porta que ele escolher.

O protocolo do jogo é o seguinte: o convidado escolhe, provisoriamente, uma das três portas. Neste momento o apresentador do programa, que sabe o que há atrás de cada porta, abre uma das outras duas portas sempre revelando um dos bodes. O convidado agora tem a opção de ficar com a primeira porta que ele escolheu ou trocar pela outra porta fechada.

Que estratégia o convidado deve adotar pra ter mais chance de ganhar o carro?

Se o convidado fica com a porta que escolheu inicialmente, então a probabilidade de ganhar um carro é $1/3$, que é a probabilidade dele ter escolhido a porta certa logo de início.

Vamos supor que o convidado troca de porta. Denotamos por E o evento "ganha o carro" e por A o evento "a primeira escolha era a correta", isto é, era a porta que escondia um carro. Claramente, $\mathbb{P}(A) = 1/3$.

Agora, se a primeira escolha era a correta então o convidado não ganha o carro, ou seja, $\mathbb{P}(E|A) = 0$, caso contrário o convidado ganha o carro, ou seja, $\mathbb{P}(E|\overline{A}) = 1$. Com isso temos

$$\mathbb{P}(E) = \mathbb{P}(E \cap A) + \mathbb{P}(E \cap \overline{A}) = \mathbb{P}(E|A)\mathbb{P}(A) + \mathbb{P}(E|\overline{A})\mathbb{P}(\overline{A}) = \frac{2}{3}$$

portanto, é melhor trocar de porta. Tente você mesmo.

Exercício 13. Sejam $A_1, A_2, \ldots, A_n$ eventos num espaço de probabilidade. Prove que

$$\mathbb{P}\left(\bigcap_{i=1}^{n} A_i\right) = \mathbb{P}\left(A_n \middle| \bigcap_{i=1}^{n-1} A_i\right) \mathbb{P}\left(A_{n-1} \middle| \bigcap_{i=1}^{n-2} A_i\right) \cdots \mathbb{P}\left(A_2 \middle| A_1\right) \mathbb{P}\left(A_1\right).$$

Dizemos que os eventos A e B são **independentes** se

$$\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B). \tag{1.5}$$

Uma coleção de eventos $E_1, \ldots, E_n$ é k**-a-**k **independente** se para todo subconjunto de índices $\{i_1, i_2, \cdots, i_k\} \subseteq \{1, 2, \ldots, n\}$

$$\mathbb{P}\left(\bigcap_{\ell=1}^{k} E_{i_\ell}\right) = \prod_{\ell=1}^{k} \mathbb{P}(E_{i_\ell}) \tag{1.6}$$

e, essa coleção é **mutuamente independente** se se é k-a-k independente para todo k, onde $2 \leqslant k \leqslant n$.

Exemplo 14. Considere três lançamentos de uma moeda equilibrada, E_{12} é o evento "o resultado da primeira e da segunda coincidem", E_{13} o evento "o resultado da primeira e da terceira coincidem" e E_{23} o evento "o resultado da segunda e da terceira coincidem". Os eventos são 2-a-2 independentes, por exemplo, $E_{12} \cap E_{13} = \{(\mathrm{cara}, \mathrm{cara}, \mathrm{cara}), (\mathrm{coroa}, \mathrm{coroa}, \mathrm{coroa})\}$ com probabilidade $1/4$ e $\mathbb{P}(E_{12}) = \mathbb{P}(E_{13}) = \mathbb{P}(E_{23}) = 1/2$, entretanto esses eventos não são mutuamente independentes pois $\mathbb{P}(E_{12} \cap E_{13} \cap E_{23}) = 1/4$.

Exemplo 15 (k**-corte-mínimo**, Karger (1993)). Seja $G = (V, E)$ um grafo. Sem perda de generalidade podemos supor que $V = \{1, 2, \ldots, n\}$. Um subconjunto de arestas de G da forma

$$E(A, \overline{A}) = \left\{\{u, v\} \in E(G) \colon u \in A \text{ e } v \in \overline{A}\right\} \tag{1.7}$$

é chamado de **corte definido por** A em G. Um **corte mínimo** em G é um corte com

$$\operatorname{mincut}(G) = \min_{\varnothing \neq A \subsetneq V} \left|E(A, \overline{A})\right|$$

arestas. O problema que estamos interessados é determinar se $\operatorname{mincut}(G) \leqslant k$, dados um grafo G e um inteiro positivo k.

Um *multigrafo* é um par (V, E) onde V é um conjunto finito cujos elementos são chamados de vértices e E é um multiconjunto finito onde cada elemento é chamado de aresta e é formado por dois vértices distintos. Seja M um multigrafo. Em M

definimos por *contração da aresta* $e \in E(M)$ a operação que resulta no multigrafo com os extremos de e identificados e os laços sobre esses extremos removidos, o multigrafo resultante é denotado por M/e.

A idéia do algoritmo que apresentaremos a seguir para determinar se $\text{mincut}(G) \leqslant k$ é repetir na seqüência de operações

- sortear aresta,
- contrair a aresta sorteada,

até que restem 2 vértices. As arestas que ligam esses 2 vértices são arestas de um corte no grafo original. A figura 1.1 representa uma seqüência de três contrações de arestas, a aresta que sofre a contração está em negrito.

Figura 1.1: Exemplo de contração de arestas.

Se no próximo passo identificarmos os vértices $1,4,5$ com o $2,3$ então temos o corte definido por $\{6\}$ em G (fig. 1.2 (a)), que tem duas arestas e é um corte mínimo. Por outro lado, se identificarmos $2,3$ e 6 então temos o corte definido por $\{2,3,6\}$ e que tem 4 arestas (fig. 1.2 (b)).

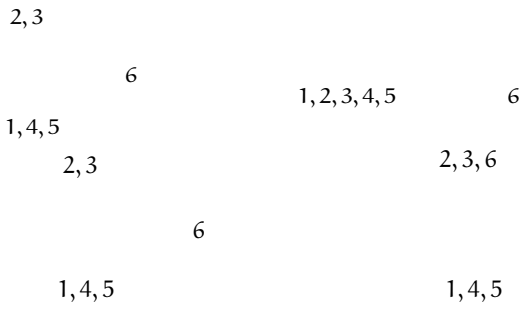

Figura 1.2: Duas possíveis continuações para as contrações da figura 1.1.

O seguinte algoritmo recebe (G,k) e se $\text{mincut}(G) > k$ então ele responde *não* com probabilidade 1, pois o resultado das contrações define um corte em G (exercício 18) e todos os cortes têm mais que k arestas. Entretanto, se $\text{mincut}(G) \leqslant k$ pode ocorrer do algoritmo não descobrir um corte com menos que k arestas e

responder *não*.

Algoritmo 4: Corte mínimo

Dado : um grafo G de ordem $n \geqslant 3$ e $k \in \mathbb{N}$.

Devolve: *sim* se há corte com $\leqslant k$ arestas, *não* caso contrário.

1 **repita**
2 $i \leftarrow 0$;
3 $G_o \leftarrow G$;
4 **repita**
5 escolha $e \in E(G_i)$ aleatoriamente;
6 $G_{i+1} \leftarrow G_i/e$;
7 $i \leftarrow i+1$;
8 **até que** $i = n-2$;
9 **se** $|E(G_{n-2})| \leqslant k$ **então devolva** *sim;*
10 **até que** *complete* $\binom{n}{2}$ *rodadas* ;
11 **devolva** *não*.

Lema 16. *Fixado um corte mínimo* C *num grafo* G *com pelo menos três vértices, a probabilidade do algoritmo* 4 *determinar* C *no laço da linha* 4 *é pelo menos* $1/\binom{n}{2}$.

Demonstração. Seja $G = (V, E)$ um grafo e $C = E(A, \overline{A})$ um corte mínimo em G definido por $A \subset V$ com m arestas. Notemos que de $\text{mincut}(G) \geqslant m$ o grau mínimo de um vértice em G é pelo menos m e que, portanto, G tem pelo menos $mn/2$ arestas.

O algoritmo devolve C se nas $n-2$ iterações somente contrai arestas com ambos os extremos em A ou ambos extremos em $\overline{A}$. O espaço amostral das $n-2$ rodadas do **repita** da linha 4 é $\Omega = \prod_i E(G_i)$. Seja B_i o evento "no i-ésimo sorteio da linha 5, $e \notin C$", isto é, $B_i = \{(e_1, \ldots, e_{n-2}) \in \Omega : e_i \notin C\}$.

A probabilidade de escolher uma aresta de C no primeiro sorteio é $|C|/|E(G_0)| \leqslant m/(nm/2)$, logo

$$\mathbb{P}(B_1) \geqslant 1 - \frac{m}{mn/2} = 1 - \frac{2}{n}.$$

Agora, a probabilidade de escolher uma aresta de C no segundo sorteio dado uma aresta de C não foi escolhida no primeiro é $|C|/|E(G_1)| \leqslant m/(m(n-1)/2)$ pois o multigrafo tem $n-1$ vértice e grau mínimo pelo menos m, logo

$$\mathbb{P}(B_2|B_1) \geqslant 1 - \frac{m}{m(n-1)/2}$$

e, genericamente,

$$\mathbb{P}\left(B_i \middle| \bigcap_{j=1}^{i-1} B_j\right) \geqslant 1 - \frac{2}{n-i+1}.$$

A probabilidade de nenhuma aresta sorteada ser de C é $\mathbb{P}\left(\bigcap_{i=1}^{n-2} B_i\right)$ e pelo exercício 13, página 18, temos que

$$\mathbb{P}\left(\bigcap_{i=1}^{n-2} B_i\right) \geqslant \prod_{i=1}^{n-2}\left(\frac{n-i-1}{n-i+1}\right) = \frac{2}{n(n-1)}$$

e o lema segue da definição de coeficiente binomial. □

Teorema 17. *Supondo que as rodadas do laço da linha 1 sejam independentes temos* $\mathbb{P}[\text{ erro }] < 1/2$.

Demonstração. Se G tem um corte com no máximo k arestas, então a probabilidade do algoritmo não encontrar um tal corte em nenhuma das iterações do laço na linha 1 é no máximo

$$\left(1 - \frac{2}{n(n-1)}\right)^{\binom{n}{2}} \leqslant \exp\left(-\frac{2}{n(n-1)}\right)^{\binom{n}{2}} = \frac{1}{e} < \frac{1}{2}.$$

□

Exercício 18. Prove que após um número qualquer de contrações de arestas, um corte no multigrafo resultante é um corte no grafo original. Conclua que a seqüência de operações (i) sortear aresta, (ii) contrair a aresta sorteada até que restem 2 vértices num grafo G termina com um corte de G.

Exercício 19. Seja G um multigrafo tal que $\text{mincut}(G) \geqslant m$. Prove que o grau mínimo de um vértice em G é m e que G tem pelo menos $mn/2$ arestas.

1.2.1 Espaço produto

Se $(\Omega_i, \mathbb{P}_i)$, para $1 \leqslant i \leqslant n$, são espaços de probabilidade então o **espaço produto** é o espaço $(\prod_{i=1}^{n} \Omega_i, \mathbb{P})$ com

$$\mathbb{P}(\omega_1, \omega_2, \ldots, \omega_n) = \prod_{i=1}^{n} \mathbb{P}_i(\omega_i)$$

para todo $(\omega_1, \omega_2, \ldots, \omega_n) \in \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$.

Exercício 20. Prove que escolher $(d_1, d_2, \ldots, d_n)$ em $\{0,1\}^n$ aleatoriamente é equivalente a escolher aleatoriamente d_i em $\{0,1\}$ para cada i, $1 \leqslant i \leqslant n$, com as escolhas mutuamente independentes.

Exemplo 21 (**Gerador de números aleatórios**). Dado um inteiro positivo M, queremos gerar um número inteiro entre 0 e $M-1$ aleatoriamente a partir de uma fonte de bits aleatórios.

Algoritmo 5: Gerador de números aleatórios

Dado : inteiro positivo $M > 0$.

Devolve: um inteiro escolhido uniformemente em $\{0, 1, \ldots, M-1\}$.

```
1 seja k o número de bits de M;
2 repita
3     para cada i ∈ {0, ..., k − 1} faça
4     d_i ←_R {0, 1};
5     N ← Σ_i d_i 2^i;
6 até que N < M ;
7 devolva N.
```

O resultado dos sorteios na linha 4, a seqüência $d_{k-1}d_{k-2}\ldots d_0$, é qualquer número em $\Omega = \{0, 1, \ldots, 2^k - 1\}$ na representação binária; a probabilidade de ocorrer uma seqüência particular de bits é $(1/2)^k$. Seja $A = \{0, 1, \ldots, M-1\}$, então a probabilidade do algoritmo responder t, para algum $t \in A$ é

$$\mathbb{P}(\{t\}|A) = \frac{\mathbb{P}(\{t\})}{\mathbb{P}(A)} = \frac{(1/2)^k}{M/2^k} = \frac{1}{M}.$$

Observação 22. Note que não há, a princípio, garantia que o algoritmo acima termine. Voltaremos a esse fato adiante (veja o exercício 54), mas deixamos registrado que o algoritmo pára com probabilidade 1.

Exercício 23. Mostre que gerar um número n de k bits e devolver n mod M, o que evitaria a execução de mais que uma iteração do **repita**, não resulta em uma resposta em $\{0, \ldots, M-1\}$ com probabilidade $1/M$.

Observação 24. Para obtermos $n \in_R \{a, \ldots, b\}$ basta executar o algoritmo 5 acima com $M = b - a$ e devolver $N + a$. Com isso, introduzimos a instrução

$$x \leftarrow_R \{a, \ldots, b\}$$

que significa que a variável x recebe o valor retornado pelo algoritmo que gera um número de $\{a, \ldots, b\}$ com probabilidade $1/(b-a+1)$.

1.2.2 Identidade polinomial revisitada

Lembremos que se p e q são polinômios então escrevemos $p \equiv q$ se os coeficiente do polinômios são iguais aos escrevermo-os como combinação linear de monômios.

Por exemplo, o determinante da matriz de Vandermonde, equação (1), é o polinômio $\prod_{1\leqslant i<j\leqslant n}(x_i - x_j)$ que como combinação linear de monômios resulta numa expressão da forma

$$\sum z_1 z_2 \cdots z_{\binom{n-1}{2}} z_{\binom{n}{2}}$$

onde $z \in \{x, y\}$, com a soma de 2^n parcelas totalizado uma expressão de tamanho $O(n^2 2^n)$, contra a expressão original de tamanho $O(n^2)$, ou seja, somente a escrita da expansão já toma tempo exponencial.

Teorema 25 (Schwartz-Zippel). *Seja* $f \in \mathbb{Q}[x_1, \ldots, x_n]$ *um polinômio de grau* d, $d > 0$ *e* $f \not\equiv 0$ *e* $S \subset \mathbb{Q}$ *finito. Então, temos*

$$\mathbb{P}_{(x_1,\ldots,x_n)\in_R S^n}\left[\, f(x_1, \ldots, x_n) = 0 \,\right] \leqslant \frac{d}{|S|}.$$

Demonstração. Fixamos $S \subset \mathbb{Q}$ finito e consideremos todos elementos de S equiprováveis. Definimos, para cada $n \in \mathbb{N}$, o espaço produto $\Omega_n = S^n$ com $\mathbb{P}_n(\omega_1, \ldots, \omega_n) = (1/|S|)^n$ para todo $(\omega_1, \omega_2, \ldots, \omega_n) \in \Omega_n$. Seja $f \in \mathbb{Q}[x_1, \ldots, x_n]$ um polinômio não-nulo de grau no máximo d, $d > 0$ e definimos evento $E_n(f) \subseteq \Omega_n$

$$E_n(f) = \left\{(r_1, \ldots, r_n) \in \Omega_n \colon (r_1, \ldots, r_n) \text{ é raiz de } f\right\}$$

A prova é por indução em n. Para $n = 1$ vimos na seção 1.1.1 que

$$\mathbb{P}_1(E_1(f)) = \mathbb{P}_{x_1\in_R S}\left[\, f(x_1) = 0 \,\right] \leqslant \frac{d}{|S|}. \tag{1.8}$$

Para $n > 1$, seja k o maior grau de x_n em f e escreva

$$f(x_1, \ldots, x_n) = g_0 {x_n}^0 + g_1 {x_n}^1 + \cdots + g_k {x_n}^k \tag{1.9}$$

com $g_i \in \mathbb{Q}[x_1, \ldots, x_{n-1}]$ de grau $\leqslant d - i$ e $0 < k \leqslant d$. Assuma que

$$\mathbb{P}(E_{n-1}(g_k)) = \mathbb{P}_{(x_1,\ldots,x_{n-1})\in_R S^{n-1}}\left[\, g_k(x_1, \ldots, x_{n-1}) = 0 \,\right] \leqslant \frac{d-k}{|S|}. \tag{1.10}$$

e vamos provar que $\mathbb{P}_n(E_n(f)) \leqslant d/|S|$. Defina o evento $E' \subset \Omega_n$ por

$$E' = \{(a_1, a_2, \ldots, a_n) \colon g_k(a_1, \ldots, a_{n-1}) = 0\} = E_{n-1}(g_k) \times S.$$

e temos

$$\mathbb{P}_n(E_n(f)) = \mathbb{P}\big(E_n(f) \cap E'\big) + \mathbb{P}\big(E_n(f) \cap \overline{E'}\big).$$

Agora, $\mathbb{P}\big(E_n(f) \cap E'\big) \leqslant \mathbb{P}_n(E') = \mathbb{P}_{n-1}(E_{n-1}(g_k))\mathbb{P}_1(S) = \mathbb{P}_{n-1}(E_{n-1}(g_k)) \leqslant (d - k)/|S|$, pela hipótese (1.10).

Ainda, para cada $(a_1, \ldots, a_{n-1}) \in \Omega_{n-1}$ tal que $g_k(a_1, \ldots, a_{n-1}) \neq 0$ temos, por (1.9), que f é um polinômio de grau k em $\mathbb{Q}[x_n]$ logo

$$\mathbb{P}\big(E_n(f) \cap \overline{E'}\big) = \mathbb{P}\big(\{(x_1, x_2, \ldots, x_n) \colon f(x_1, x_2, \ldots, x_n) = 0 \text{ e } g_k(x_1, x_2, \ldots, x_{n-1}) \neq 0\big) \leqslant \frac{k}{|S|}.$$

Assim,

$$\mathbb{P}(E_n(f)) \leqslant \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}.$$

□

Corolário 26. *Se $S = \{1, 2, \ldots, 3d\}$ então o algoritmo 2 erra com probabilidade no máximo $1/3$.* □

Finalmente, podemos usar rodadas independentes para reduzir a probabilidade de erro.

Algoritmo 6: Teste de identidade entre polinômios.

Dado : $d > 0$ e $f \in \mathbb{Q}[x_1, \ldots, x_n]$ de grau (total) no máximo d e o número de rodadas k.

Devolve: *sim* se $f \equiv 0$ ou *não*, caso contrário.

repita
  para cada $i \in \{0, \ldots, k-1\}$ **faça** $x_i \leftarrow_R \{1, 2, \ldots, 3d\}$;
  se $f(x_1, x_2, \ldots, x_n) \neq 0$ **então devolva** *não*;
até que *complete* k *rodadas* ;
devolva *sim*.

Pelos resultados acima

$$\begin{aligned} \mathbb{P}\,[\, f(x_1, \ldots, x_n) = 0 | f \equiv 0 \,] &= 1 \\ \mathbb{P}\,[\, f(x_1, \ldots, x_n) = 0 | f \not\equiv 0 \,] &\leqslant 1/3; \end{aligned}$$

se $f \not\equiv 0$ então o algoritmo responde *sim* somente se nas k iterações (independentes) do **repita** foi sorteada uma raiz de f, ou seja, ocorreu o evento $E_n(f)$. Como as iterações são mutuamente independentes

$$\mathbb{P}\,[\,\text{erro}\,] = \mathbb{P}\big(E_n(f) \times \cdots \times E_n(f)\big) = \prod_{i=1}^{k} \mathbb{P}\big(E_n(f)\big) \leqslant \left(\frac{1}{3}\right)^k.$$

1.2.3 Igualdade do produto de matrizes

Dadas as matrizes $\mathbf{A}, \mathbf{B}, \mathbf{C}$ quadradas de dimensão n sobre $\mathbb{Q}$ queremos decidir se vale a igualdade $\mathbf{AB} = \mathbf{C}$. Para isso, fixamos $S \subset \mathbb{Q}$ finito e construímos o

algoritmo

Algoritmo 7: Teste de produto de matrizes.

Dado : matrizes $\mathbf{A}, \mathbf{B}, \mathbf{C}$ quadradas de ordem n.

Devolve: *sim* se $\mathbf{AB} = \mathbf{C}$ ou *não*, caso contrário.

$v \in_R S^n$;

se $\mathbf{AB}v = \mathbf{C}v$ **então devolva** *sim*;

senão devolva *não*

que sorteia um vetor $v = (r_1, \ldots, r_n) \in_R S^n$ testa se $\mathbf{AB}v = \mathbf{C}v$; se valer a igualdade então responde *sim*, senão responde *não*. Assim, se de fato $\mathbf{AB} = \mathbf{C}$ então o algoritmo responde certo, mas se $\mathbf{AB} \neq \mathbf{C}$ então pode ser que uma escolha ruim de v leve o algoritmo a responder *sim*, um falso positivo.

Para analisar a probabilidade de erro vamos introduzir uma estratégia chamada *princípio da decisão adiada*. Seja $\mathbf{D} = \mathbf{AB} - \mathbf{C}$, escreva $\mathbf{D} = (d_{i,j})$ e assuma que $\mathbf{D} \neq \mathbf{0}$. Queremos estimar

$$\mathbb{P}[\text{ erro }] = \mathbb{P}_{v \in_R S^n}[\ \mathbf{D}v = \mathbf{0}\ ].$$

De $\mathbf{D} \neq \mathbf{0}$, existem ℓ e c tais que $d_{\ell,c} \neq 0$. Como $\sum_{j=1}^{n} d_{\ell,j} r_j = 0$ temos

$$r_c = \frac{-\sum_{j=1}^{n} d_{\ell,j} r_j}{d_{\ell,c}}$$

e, se consideramos que cada coordenada foi sorteada independentemente, podemos assumir que r_i, para todo $i \neq c$, foi sorteado antes de r_c (a decisão de r_c foi adiada), então o lado direito da igualdade acima fica determinado e a probabilidade de sortear r_c que satisfaça a igualdade é no máximo $1/|S|$. Portanto,

$$\mathbb{P}[\text{ erro }] \leqslant \frac{1}{|S|}.$$

Pondo em outra forma, sejam

$$E = \{v \in S^n : \mathbf{D}v = \mathbf{0}\} \quad \subset \quad E' = \left\{(r_1, r_2, \ldots, r_n) \in S^n : r_c = \frac{-\sum_j d_{\ell,j} r_j}{d_{\ell,c}}\right\},$$

onde $\mathbf{D}$, ℓ e c são como acima. Podemos particionar o espaço amostral S^n definindo para cada $(n-1)$-upla de elementos de S, $(r'_1, \ldots, r'_{n-1}) \in \{0,1\}^{n-1}$, o evento

$$A_{r'_1,\ldots,r'_{n-1}} = \{(r'_1, \ldots, r'_{c-1}, s, r'_c, \ldots, r'_{n-1}) : s \in S\}$$

e assim, para cada $v = (r_1, \ldots, r_n) \in S^n$ existe um, e só um, $(r'_1, \ldots, r'_{n-1}) \in S^{n-1}$ tal que $v \in A_{r'_1,\ldots,r'_{n-1}}$. Pela lei das probabilidades totais e de $E \subset E'$

$$\begin{aligned}\mathbb{P}(E) &= \sum_{(r'_1,\ldots,r'_{n-1}) \in S^{n-1}} \mathbb{P}(E \cap A_{r'_1,\ldots,r'_{n-1}}) \leqslant \sum_{(r'_1,\ldots,r'_{n-1}) \in S^{n-1}} \mathbb{P}(E' \cap A_{r'_1,\ldots,r'_{n-1}}) \\ &= \sum_{(r'_1,\ldots,r'_{n-1}) \in S^{n-1}} \mathbb{P}(E' | A_{r'_1,\ldots,r'_{n-1}}) \mathbb{P}(A_{r'_1,\ldots,r'_{n-1}})\end{aligned}$$

da independência das coordenadas temos

$$\mathbb{P}(E'|A_{r'_1,\ldots,r'_{n-1}}) = \mathbb{P}\left(\left\{s \in S\colon s = \frac{-\sum_j d_{\ell,j} r'_j}{d_{\ell,c}}\right\}\right) \leqslant \frac{1}{|S|}$$

assim

$$\mathbb{P}(E) \leqslant \sum_{(r'_1,\ldots,r'_{n-1})\in S^{n-1}} \frac{1}{|S|}\mathbb{P}(A_{r'_1,\ldots,r'_{n-1}}) = \frac{1}{|S|}.$$

Exercício 27. Um baralho de 52 cartas é embaralhado de modo que a disposição final é qualquer uma as 52! possibilidades com igual probabilidade, em seguida é dividido em 13 montes de 4 cartas cada, todo monte tem um rótulo tomado em A,2,3,...,9,10,J,Q,K arbitrariamente. No primeiro movimento abrimos uma carta do monte K, e o resultado (ignoramos o naipe) indica o próximo monte donde abriremos uma carta, e assim por diante. O jogo acaba quando uma jogada indica um monte vazio. Vencemos o jogo quando abrimos todas as cartas do baralho.

Use o princípio da decisão adiada para provar que a probabilidade de vencermos o jogo é $\leqslant 1/13$.

Exercício 28. Um baralho de 52 cartas é embaralhado de modo que a disposição final é qualquer uma dentre as 52! possibilidades com igual probabilidade. Denote por E o evento "a carta do topo é de espadas" e F é o evento "a quarta carta a partir do topo é espada". Use o princípio da decisão adiada para provar que $\mathbb{P}(E) = \mathbb{P}(F)$.

Exercício 29. Lançamos dez dados equilibrados. Supondo os resultados independentes, use o princípio da decisão adiada para determinar a probabilidade da soma dos resultados ser divisível por 6.

1.3 Teorema de Bayes

Exercício 30 (**Teorema de Bayes**). Prove que se $E_1, \ldots, E_m$ é uma partição do espaço amostral, então

$$\mathbb{P}(E_j|A) = \frac{\mathbb{P}(A|E_j)\mathbb{P}(E_j)}{\sum_{i=1}^{m} \mathbb{P}(A|E_i)\mathbb{P}(E_i)}. \tag{1.11}$$

Exemplo 31. Suponha que *probabilite* é uma doença que afeta 1% da população, que existe um teste para detectar *probabilite* e que não é perfeito: há 3% de falsos positivos e 2% de falsos negativos. Dado que o teste para um determinado paciente deu positivo, qual é a probabilidade que ele tenha de fato a doença?

A lenda diz que a maioria dos médicos respondem 97% sem pestanejar, já que há 3% de falsos positivos. Certamente, essa justificativa está errada, vamos determinar essa probabilidade usando o teorema de Bayes.

Seja A o evento "o exame deu positivo" e B o evento "tem *probabilite*". Por Bayes

$$\mathbb{P}(B|A) = \frac{\mathbb{P}(A|B)\mathbb{P}(B)}{\mathbb{P}(A|B)\mathbb{P}(B) + \mathbb{P}(A|\bar{B})\mathbb{P}(\bar{B})} = \frac{0,98 \cdot 0,01}{0,98 \cdot 0,01 + 0,03 \cdot 0,99}$$

ou seja, $\mathbb{P}(B|A) = 0,24810126582278$, a chance de ter a doença dado que o teste foi positivo é menor que 25%.

Agora, suponha que o paciente foi ao médico por que estava com dor de cabeça e o teste deu positivo. É sabido que todos que tem *probabilite* apresentam dor de cabeça, enquanto que 10% da população apresenta dor de cabeça; também é sabido que o evento C, "ter dor de cabeça", não afeta a precisão do teste. Com esse fato, a probabilidade de estar doente dado que o teste deu positivo e o paciente tem dor de cabeça é

$$\begin{aligned}
\mathbb{P}(B|A \cap C) &= \frac{\mathbb{P}(A \cap C|B)\mathbb{P}(B)}{\mathbb{P}(A \cap C|B)\mathbb{P}(B) + \mathbb{P}(A \cap C|\bar{B})\mathbb{P}(\bar{B})} \\
&= \frac{\mathbb{P}(A|B)\mathbb{P}(C|B)\mathbb{P}(B)}{\mathbb{P}(A|B)\mathbb{P}(C|B)\mathbb{P}(B) + \mathbb{P}(A|\bar{B})\mathbb{P}(C|\bar{B})\mathbb{P}(\bar{B})} \\
&= \frac{0,98 \cdot 1 \cdot 0,01}{0,98 \cdot 1 \cdot 0,01 + 0,03 \cdot 0,1 \cdot 0,99} = 0,76742364917776
\end{aligned}$$

ou seja, mais que 75% de chance de ter a doença. Estima-se que 90% da população terá algum tipo de dor de cabeça durante a vida. Se uma pessoa com dor de cabeça ter um teste positivo para *probabilite*, qual a probabilidade dessa pessoa estar doente?

A seguir veremos duas aplicações em computação.

1.3.1 Sigilo perfeito

Um *sistema de codificação* é uma quina $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{D}, \mathcal{E})$ de conjuntos, onde $\mathcal{P}$ é o espaço dos textos comuns (*plaintext*); $\mathcal{C}$ é o espaço dos textos cifrados (*chipertext*); $\mathcal{K}$ é o espaço das chaves; $\mathcal{E}$ é o espaço das funções de codificação $E_k\colon \mathcal{P} \to \mathcal{C}$, onde $k \in \mathcal{K}$; $\mathcal{D}$ é o espaço das funções de decodificação $D_k\colon \mathcal{C} \to \mathcal{P}$, onde $k \in \mathcal{K}$; tais que para cada $e \in \mathcal{K}$ existe $d \in \mathcal{K}$ tal que $D_d(E_e(p)) = p$.

Exemplo 32 (Cifra de César). Essa cifra identifica o alfabeto com o $\mathbb{Z}_{26}$, assim $\mathcal{K} = \mathbb{Z}_{26}$ e $\mathcal{P}, \mathcal{C} = \{a, b, \ldots, z\}$. Para $e, d \in \mathbb{Z}_{26}$ temos as funções

$$E_e(x) = (x + e) \bmod 26 \qquad \text{e} \qquad D_d(x) = (x - d) \bmod 26.$$

Agora, estenda o sistema para que os textos sejam seqüências de de letras do alfabeto. Por exemplo, para a chave $e = d = 3$

Normal: essaauladasono
Cifrado: hvvddzodgdvrqr

Exemplo 33 (*one-time pad*). Gilbert Sandford Vernam inventou o seguinte sistema de codificação: dada uma chave k, escolhida previamente

$$E_k(x) = x \,\mathbf{xor}\, k \qquad e \qquad D_k(y) = y \,\mathbf{xor}\, k. \tag{1.12}$$

O fato da chave ser re-usada tornava o sistema inseguro. Pouco tempo depois Joseph Mauborgne propôs que a chave fosse aleatória e usada uma única vez (*one-time pad*). Claude Shannon provou que o *one-time pad* é uma codificação "inquebrável".

A idéia de *sigilo perfeito* pode ser interpretada da seguinte: escolhemos um texto $P \in \mathcal{P}$ de acordo com a distribuição $\mathbb{P}_{\mathcal{P}}$ (baseada, por exemplo, na freqüência com que as letras aparecem nas palavras em uma determinada língua) e o ciframos. Entregamos ao adversário o texto cifrado C e perguntamos qual a probabilidade do texto que foi cifrado? Se o adversário não pode fazer melhor do que responder $\mathbb{P}_{\mathcal{P}}(P)$, ou seja, o conhecimento de C não torna uns textos mais prováveis e outros menos prováveis do que eles são, significa que ele não ganhou informação com o conhecimento de C e temos sigilo perfeito.

Fixamos um sistema com $\mathcal{P}, \mathcal{K}, \mathcal{C}$ finitos e uma função de probabilidade $\mathbb{P}_{\mathcal{P}} \colon \mathcal{P} \to [0,1]$ tal que $\mathbb{P}_{\mathcal{P}}(P) > 0$ $(\forall P \in \mathcal{P})$. Tomamos o espaço produto $(\mathcal{K} \times \mathcal{P}, \mathbb{P})$ com probabilidade $\mathbb{P}_{\mathcal{K}}$ nas chaves e $\mathbb{P}_{\mathcal{P}}$ nos textos e definimos as funções sobre o espaço amostral

$$\begin{array}{ccc} k \colon \mathcal{K} \times \mathcal{P} \to \mathcal{K} & p \colon \mathcal{K} \times \mathcal{P} \to \mathcal{P} & c \colon \mathcal{K} \times \mathcal{P} \to \mathcal{C} \\ (K,P) \mapsto K & (K,P) \mapsto P & (K,P) \mapsto E_K(P) \end{array} \tag{1.13}$$

dessa forma $\mathbb{P}[\, p = P \,] = \mathbb{P}\big(\{(K,P) \colon k \in \mathcal{K}\}\big)$ e $\mathbb{P}[\, c = C \,] = \mathbb{P}\big(\{(K,P) \colon E_K(P) = C\}\big)$. Notemos que $\mathbb{P}[\, p = P \,] = \mathbb{P}_{\mathcal{P}}(P)$.

Um sistema de codificação nas condições acima tem **sigilo perfeito** se para todo $P \in \mathcal{P}$ e todo $C \in \mathcal{C}$

$$\mathbb{P}\left[\, p = P \middle| c = C \,\right] = \mathbb{P}[\, p = P \,]. \tag{1.14}$$

Exemplo 34. O seguinte sistema de codificação não tem sigilo perfeito.

Tome $\mathcal{P} = \{\alpha, \beta\}$ com $\mathbb{P}_{\mathcal{P}}(\alpha) = 1/4$ e $\mathbb{P}_{\mathcal{P}}(\beta) = 3/4$, os textos cifrados $\mathcal{C} = \{a, b\}$ e chaves $\mathcal{K} = \{0, 1\}$ com $\mathbb{P}_{\mathcal{K}}(0) = 1/4$ e $\mathbb{P}_{\mathcal{K}}(1) = 3/4$. A cifra $E_K(P)$ é dada por

$_P\backslash^K$	0	1
α	a	b
β	b	a

resulta em $\mathbb{P}[\ c = a\ ] = 5/8$ e $\mathbb{P}[\ c = b\ ] = 3/8$. As probabilidades condicionais no lado esquerdo de (1.14) são dadas abaixo

	$[p = \alpha \vert c = a]$	$[p = \beta \vert c = a]$	$[p = \alpha \vert c = b]$	$[p = \beta \vert c = b]$
$\mathbb{P}[p = P \vert c = C]$	1/10	9/10	1/2	1/2

Esse sistema de codificação não tem sigilo perfeito pois se o adversário recebe $c = a$ então ele tem quase certeza de que o texto cifrado foi β.

Exemplo 35. Um exemplo de sistema com sigilo perfeito é o *One-time pad* com $\mathcal{P} = \mathcal{K} = \mathcal{C} = \{0, 1\}^2$. A seguinte tabela tem entradas $\mathbb{P}[\ c = C \vert p = P\ ]$; fixado P as possíveis cifras são $P \,\mathbf{xor}\, K$ onde $K \in_R \mathcal{K}$, portanto, independente de P as cifras são eqüiprováveis.

		C	00	01	10	11
$\mathbb{P}_{\mathcal{P}}(P)$	P					
1/6	00		1/4	1/4	1/4	1/4
1/3	01		1/4	1/4	1/4	1/4
1/4	10		1/4	1/4	1/4	1/4
1/4	11		1/4	1/4	1/4	1/4

A tabela abaixo mostra $\mathbb{P}[\ p = P \vert c = C\ ]$,

		C	00	01	10	11
$\mathbb{P}_{\mathcal{P}}(P)$	P					
1/6	00		1/6	1/6	1/6	1/6
1/3	01		1/3	1/3	1/3	1/3
1/4	10		1/4	1/4	1/4	1/4
1/4	11		1/4	1/4	1/4	1/4

Nesse exemplo temos sigilo perfeito pra essa distribuição particular $\mathbb{P}_{\mathcal{P}}$.

Lema 36. *Para o* one-time pad *com as chaves equiprováveis*

$$\mathbb{P}[\ c = C \vert p = P\ ] = \frac{1}{2^n}$$

para qualquer $P \in \mathcal{P}$*, qualquer* $C \in \{0,1\}^n$ *e qualquer distribuição de probabilidades sobre* $\mathcal{P}$*.*

Demonstração. Fixado P, temos $C = P \operatorname{\mathbf{xor}} K$ se e só se $K = P \operatorname{\mathbf{xor}} C$. Como K é escolhida aleatoriamente a probabilidade é $(1/2)^n$. □

Teorema 37. *O* one-time pad *tem sigilo perfeito.*

Demonstração. Usando o teorema de Bayes

$$\mathbb{P}[\, p = P | c = C \,] = \frac{\mathbb{P}[\, c = C | p = P \,]\,\mathbb{P}[\, p = P \,]}{\sum_{X \in \mathcal{P}} \mathbb{P}[\, c = C | p = X \,]\,\mathbb{P}[\, p = X \,]} = \frac{2^{-n}\mathbb{P}[\, p = P \,]}{\sum_{X \in \mathcal{P}} 2^{-n}\mathbb{P}[\, p = X \,]} = \mathbb{P}_{\mathcal{P}}(P)$$

□

Observação 38. O *one-time pad* não é o único sistema que possui sigilo perfeito, mas foi o primeiro a ser descoberto.

Exercício 39. Prove ou refute: a cifra de César, exemplo 32, tem sigilo perfeito se em $\mathcal{K}$ as chaves são equiprováveis.

Exercício 40 (**Teorema de Shannon**). Dados um sistema com $\mathcal{P}, \mathcal{K}, \mathcal{C}$ finitos e $|K| = |C|$ e dada uma distribuição de probabilidade $\mathbb{P}_{\mathcal{P}} \colon \mathcal{P} \to [0,1]$ tal que $\mathbb{P}_{\mathcal{P}}(P) > 0$, para todo $P \in \mathcal{P}$, esse sistema tem sigilo perfeito se e somente se as chaves são equiprováveis e se para todo $P \in \mathcal{P}$ e todo $C \in \mathcal{C}$ existe um único $K \in \mathcal{K}$ tal que $E_K(P) = C$.

1.3.2 *Spam*

Agora mostraremos uma aplicação do teorema de Bayes na classificação de mensagens eletrônicas que são *spam*. O espaço amostral é dado por $\{0,1\}^n$ de modo que cada coordenada indica se uma mensagem tem (1) ou não tem (0) uma determinada característica e a primeira coordenada, especificamente, é 1 se a mensagem é *spam* e 0 caso, contrário. Assim o evento S dado por $\{1\} \times \{0,1\}^{n-1}$ é o evento "é *spam*" e $\overline{S} = \{0\} \times \{0,1\}^{n-1}$ é o evento "não é *spam*". Denotamos por C_i o evento "tem a característica i" que corresponde a todos os vetores cuja i-ésima coordenada é 1.

Vamos assumir que as características $2, 3 \ldots, n$ são mutuamente independentes condicionalmente

$$\begin{aligned} \mathbb{P}\left(\bigcap_{i \in I} C_i | S\right) &= \prod_{i \in I} \mathbb{P}(C_i | S) \\ \mathbb{P}\left(\bigcap_{i \in I} C_i | \overline{S}\right) &= \prod_{i \in I} \mathbb{P}(C_i | \overline{S}), \end{aligned}$$

para todo $I \subseteq \{2, 3, \ldots, n\}$, o que pode não ser uma hipótese muito realista. Por exemplo, se a característica 2 é conter a palavra *watch* e a característica 3 é conter a palavra *replica* então um email que contém *replica* tem muito mais chance de conter *watch*.[1]

Exercício 41. Prove que

$$\mathbb{P}\left(S \Big| \bigcap_{i\in I} C_i\right) = \frac{\prod_{i\in I}\mathbb{P}(S|C_i)}{\prod_{i\in I}\mathbb{P}(S|C_i) + \left(\mathbb{P}(S)/\mathbb{P}(\overline{S})\right)^{|I|-1}\prod_{i\in I}\mathbb{P}(\overline{S}|C_i)}. \tag{1.15}$$

(Dica: teorema de Bayes com S, $\overline{S}$ e $\bigcap_i C_i$.)

Agora, suponha que temos 1.000 mensagens, 500 *spam*s e 500 não *spam*s. O trabalho inicial é definir características das mensagens que são *spam*s das que não são. Por exemplo, nas minhas mensagens muitos dos *spam*s têm a palavra *watch* enquanto que muitos dos não *spam*s têm a palavra 'reunião'; a maioria das mensagens têm a palavra 'a', tantos as *spam*s quanto as não *spam*s, logo 'a' não deve ser uma característica classificatória.

Para a característica i, seja k_i a quantidade de *spam*s que têm a característica i e seja ℓ_i a quantidade de não *spam*s que têm a característica i; definimos

$$\mathbb{P}(C_i|S) = \frac{k_i}{500} \quad \text{e} \quad \mathbb{P}(C_i|\overline{S}) = \frac{\ell_i}{500}$$

e pelo teorema de Bayes

$$\mathbb{P}(S|C_i) = \frac{\mathbb{P}(C_i|S)\mathbb{P}(S)}{\mathbb{P}(C_i|S)P(S) + \mathbb{P}(C_i|\overline{S})\mathbb{P}(\overline{S})} = \frac{k_i\mathbb{P}(S)}{k_i\mathbb{P}(S) + \ell_i\mathbb{P}(\overline{S})}.$$

Recebida uma mensagem, filtramo-lá para determinar suas características, digamos C_i com $i \in I$, e (1.15) determina a probabilidade dessa mensagem ser *spam*.

1.4 Variáveis aleatórias discretas

Uma **variável aleatória** discreta X é uma função qualquer $X\colon \Omega \to \mathbb{R}$, onde Ω é o espaço amostral de um espaço de probabilidade discreto $(\Omega, \mathbb{P})$.

Escrevemos $[X = t]$ para o evento $\{\omega \in \Omega\colon X(\omega) = t\}$. Notemos que $(\mathrm{Im}(X), \mathbb{P}_X)$ com $\mathbb{P}_X(t) = \mathbb{P}[\,X = t\,]$ é um espaço de probabilidade.

[1] O meu endereço de *email* recebe várias dessas mensagens que anunciam replica de relógios e *watch* só aparece nesses *spam*s. Esse pode não ser o caso de um relojoeiro num país de língua inglesa.

Sejam $X_1, X_2, \ldots, X_n$ variáveis aleatórias definidas sobre o mesmo espaço amostral. Essas variáveis são **independentes** k-**a**-k se para qualquer $I \subset \{1, 2, \ldots, n\}$, $|I| = k$, e para quaisquer x_i, $i \in I$,

$$\mathbb{P}\left(\bigcap_{i\in I} [X_i = x_i]\right) = \prod_{i\in I} \mathbb{P}([X_i = x_i]) \tag{1.16}$$

e essas variáveis são **mutuamente independentes** se são k-a-k independente para todo k, com $2 \leqslant k \leqslant n$.

A **esperança**, ou **valor médio**, da variável aleatória X, que denotamos por $\mathbb{E}\,X$, é dado por

$$\mathbb{E}\,X = \sum_{\omega\in\Omega} X(\omega)\mathbb{P}(\omega)$$

quando $\sum_\omega |X(\omega)|\mathbb{P}(\omega)$ converge[2], e nesse caso também dizemos que a esperança é *finita*, caso contrário dizemos que a esperança *não existe* ou que *não é finita*. Podemos rearranjar os termos da soma de modo que

$$\mathbb{E}\,X = \sum_{i\in \mathrm{Im}(X)} \sum_{\substack{\omega\in\Omega \\ X(\omega)=i}} i\mathbb{P}(\omega) = \sum_{i\in\mathrm{Im}(X)} i\mathbb{P}\,[\,X = i\,].$$

No caso infinito enumerável $\mathbb{E}\,X$ acima existe quando $\sum_i |i|\mathbb{P}\,[\,X = i\,]$ converge.

Exemplo 42. Seja X uma variável aleatória inteira e positiva tal que $\mathbb{P}\,[\,X = i\,] = (ci^3)^{-1}$, onde c é o valor da série $\sum_{i\geqslant 1} 1/i^3$. Essa variável tem média

$$\mathbb{E}\,X = \sum_{i\geqslant 1} i\mathbb{P}\,[\,X = i\,] = \sum_{i\geqslant 1} \frac{1}{ci^2} = \frac{\pi^2}{6c}.$$

Por outro lado,

$$\mathbb{E}\,X^2 = \sum_{i\geqslant 1} i^2\mathbb{P}\,[\,X = i\,] = \sum_{i\geqslant 1} \frac{1}{ci} = \infty.$$

Exemplo 43. Seja X o número de lançamentos de uma moeda equilibrada até sair cara (veja exemplo 3), então

$$\mathbb{E}\,X = \sum_{i\geqslant 1} i\mathbb{P}\,[\,X = i\,] = \sum_{i\geqslant 1} \frac{i}{2^i} = 2.$$

Se a moeda é viciada e p é a probabilidade de sair cara, então a probabilidade de ω_n, definido no exemplo 3, supondo que os resultados são independentes é $(1-p)^{n-1}p$ e

$$\mathbb{E}\,X = \sum_{i\geqslant 1} i\mathbb{P}\,[\,X = i\,] = \sum_{i\geqslant 1} i(1-p)^{i-1}p = p\sum_{i\geqslant 1} i(1-p)^{i-1} = \frac{1}{p}.$$

[2]Exigimos que a série convirja absolutamente. Essa hipótese não é necessária no caso de espaços finitos.

Teorema 44 (Linearidade da esperança). *Se X e Y são variáveis aleatórias finitas sobre Ω e $c \in \mathbb{R}$ uma constante qualquer então*

$$\mathbb{E}(cX + Y) = c\,\mathbb{E}\,X + \mathbb{E}\,Y. \tag{1.17}$$

Demonstração. Usando a definição e a convergência absoluta no caso de variável sobre um espaço infinito (isso permite rearranjar os termos da série)

$$\begin{aligned}\mathbb{E}(cX + Y) &= \sum_{\omega} (cX(\omega) + Y(\omega))\mathbb{P}(\omega)\\ &= \sum_{\omega} cX(\omega)\mathbb{P}(\omega) + \sum_{\omega} Y(\omega)\mathbb{P}(\omega)\\ &= c\,\mathbb{E}\,X + \mathbb{E}\,Y.\end{aligned}$$

□

O teorema acima estende-se, facilmente, usando indução para provar que se $X = \sum_{i=1}^{n} X_i$ então

$$\mathbb{E}\,X = \sum_{i=1}^{n} \mathbb{E}\,X_i,$$

e, também pode-se provar que se $X = \sum_{i\geqslant 1} X_i$ e $\sum_{i\geqslant 1} \mathbb{E}\,|X_i|$ converge, então

$$\mathbb{E}\,X = \sum_{i\geqslant 1} \mathbb{E}\,X_i.$$

Exemplo 45. Considere os inteiros positivos com probabilidade $\mathbb{P}(n) = 2^{-n}$ e a as variáveis aleatórias $\{X_n\}_{n\geqslant 1}$ dadas por

$$X_n(j) = \begin{cases} 2^n, & \text{se } j = n,\\ -2^{n+1}, & \text{se } j = n+1,\\ 0, & \text{caso contrário.}\end{cases}$$

Logo

$$\mathbb{E}\,X_n = \sum_{j\geqslant 1} X_n(j)\mathbb{P}(j) = 2^n 2^{-n} + \left(-2^{n+1}\right) 2^{n+1} = 0,$$

portanto, $\sum_{n\geqslant 1} \mathbb{E}\,X_n = 0$. No entanto, para a variável aleatória $X = \sum_{n\geqslant 1} X_n$ temos $X(1) = 2$ e

$$X(j) = \sum_{n\geqslant 1} X_n(j) = 0 + \cdots + 0 + (-2)^j + 2^j + 0 + 0 + \cdots = 0$$

para todo $j \geqslant 2$, portanto, $\mathbb{E}\,X = \sum_{n\geqslant 1} X(n)\mathbb{P}(n) = 2\frac{1}{2} + 0 + 0 + \cdots = 1$. Logo, não vale que $\mathbb{E}\sum_n X_n = \sum_n \mathbb{E}\,X_n$.

A **esperança condicional** de uma variável aleatória com respeito ao evento E é definida, naturalmente, por

$$\mathbb{E}\left[X\middle|E\right] = \sum_{i \in \mathrm{Im}(X)} i\mathbb{P}\left[\, X = i\middle|E\,\right]. \tag{1.18}$$

Exemplo 46. Se dois dados equilibrados são lançados, X_1 e X_2 são os valores obtidos e $X = X_1 + X_2$ então

$$\begin{aligned}\mathbb{E}\left[X|X_1 = 2\right] &= \sum_{i=2}^{12} i\mathbb{P}\left[\, X = i|X_1 = 2\,\right] = \frac{11}{2} \text{ e}\\ \mathbb{E}\left[X_1|X = 5\right] &= \sum_{i=1}^{6} i\mathbb{P}\left[\, X_1 = i|X = 5\,\right] = \frac{5}{2}.\end{aligned}$$

Exercício 47. Prove que para variáveis aleatórias X e Y valem

$$\mathbb{E}\,X = \sum_{y \in \mathrm{Im}(Y)} \mathbb{E}\left[X|Y = y\right]\mathbb{P}\left[\, Y = y\,\right].$$

Exercício 48. Suponha que temos uma fonte de bits aleatórios que devolve 1 com probabilidade p e 0 com probabilidade $q = 1 - p$, independentemente, com $p \neq q$. Mostre como obter bits com probabilidade $1/2$ a partir dessa fonte. Qual é o número total esperado de bits gerados para se obter uma seqüência de k bits com todas as seqüências equiprováveis?

1.4.1 Distribuições de Bernoulli, binomial, geométrica e uniforme

Uma variável aleatória X com **distribuição de Bernoulli** assume valores em $\{0, 1\}$, vale 1 com probabilidade de *sucesso* p e 0 com probabilidade de *fracasso* $1 - p$; o valor esperado é

$$\mathbb{E}\,X = \mathbb{P}\left[\, X = 1\,\right] = p.$$

Nesse caso, escrevemos $X \sim \mathrm{Be}(p)$.

Consideremos n variável aleatórias $X_1, \ldots, X_n$ independentes e com distribuição de Bernoulli com probabilidade de sucesso $p > 0$. O número de sucessos é a variável aleatória $S = \sum_{i=1}^{n} X_i$ tal que

$$\mathbb{P}\left[\, S = k\,\right] = \binom{n}{k} p^k (1-p)^{n-k}. \tag{1.19}$$

Nesse caso, S assume valores em $\{0, 1, \ldots, n\}$ e dizemos que S tem **distribuição binomial** com parâmetros n e p, o que é denotado por $S \sim \mathrm{Bi}(n, p)$.

Exercício 49. Prove que $\sum_{k=0}^{n} \mathbb{P}[\, S = k \,] = 1$ e que $\mathbb{E}\, S = np$.

Um experimento que tem dois resultados possíveis, sucesso e fracasso é chamado de **experimento de Bernoulli**. Seja G o número de experimentos independentes de Bernoulli até que ocorra um sucesso, então G assume valores em $\mathbb{N}$ e dizemos que G é uma variável aleatória com **distribuição geométrica**, que denotamos por $G \sim \mathrm{Ge}(p)$,

$$\mathbb{P}[\, G = k \,] = (1-p)^{k-1}p.$$

Do exemplo 43 sabemos que se $G \sim \mathrm{Ge}(p)$, então

$$\mathbb{E}\, G = \frac{1}{p}.$$

Exemplo 50. Seja X o número de lançamentos de uma moeda até sair cara, onde $\mathbb{P}(\text{cara}) = p \in (0,1)$. Nesse exemplo, vamos usar esperança condicional para mostrar que $\mathbb{E}\, X = 1/p$. Seja E o evento $\{\omega_1\}$, ou seja, o primeiro lançamento é cara (veja a definiçã de ω_n no exemlo 3), então

$$\mathbb{E}\, X = \mathbb{E}\left[X\middle|E\right]\mathbb{P}(E) + \mathbb{E}\left[X\middle|\overline{E}\right]\mathbb{P}(\overline{E}) = 1 \cdot p + \mathbb{E}\left[X\middle|\overline{E}\right](1-p).$$

Agora, se denotamos por X' o número de lançamentos, depois do primeiro, até que saia cara, então $\mathbb{E}\left[X\middle|\overline{E}\right] = \mathbb{E}\left[X'+1\right] = \mathbb{E}[X'] + 1 = \mathbb{E}[X] + 1$, essa última igualdade segue do exercício 9, página 45. Substituindo na equação acima ficamos com

$$\mathbb{E}\, X = \mathbb{E}\, X(1-p) + 1 + p$$

portanto $\mathbb{E}\, X = 1/p$.

Exemplo 51 (Uma análise do algoritmo 5). No caso do gerador de números aleatórios, algoritmo 5 na página 22, se I é o número de iterações do **repita** então $I \sim \mathrm{Ge}(\frac{M}{2^k})$, logo $\mathbb{E}\, I = 2^k/M \leqslant 2$. Ainda, como cada iteração do algoritmo 5 sorteia um bit em tempo constante, o sorteio de um número custa $O(k)$ e a computação de N e a comparação custam $O(k)$, portanto o tempo esperado de execução é $O(k)\,\mathbb{E}\, I = O(k) = O(\log M)$.

Exercício 52. Mostre que se $X \geqslant 0$ então $\mathbb{E}\, X = \sum_{n \geqslant 0} \mathbb{P}(X \geqslant n)$.

Exercício 53. Mostre que se $Z \sim \mathrm{Ge}(p)$ então $\mathbb{P}[\, Z \geqslant n \,] = (1-p)^{n-1}$.

Exercício 54. Prove que o algoritmo 5 pára com probabilidade 1.

Distribuição uniforme. Seja X uma variável aleatória que assume valores em $\{x_1, x_2, \ldots, x_m\}$. Dizemos que X tem **distribuição uniforme** se $\mathbb{P}[\, X = x_i \,] = 1/m$ para qualquer $1 \leqslant i \leqslant m$, isto é, X assume qualquer um dos m valores com probabilidade $1/m$. Nesse caso escrevemos $X \sim U(m)$, ou mais especificamente que $X \in_R \{x_1, x_2, \ldots, x_m\}$.

Exemplo 55. Note que na nossa notação $N \leftarrow_R \{0, \ldots, b\}$ podemos interpretar N como uma variável aleatória com distribuição uniforme. O espaço amostral é $\{0,1\}^k$ e $N(b_0, b_1, \ldots, b_{k-1}) = \sum_i b_i 2^i$.

1.4.2 MAX 3-SAT

O **MAX 3-SAT** (*Maximum 3-Satisfiability Problem*) é o problema: dado uma fórmula booleana $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ com cada cláusula C_i sendo uma conjunção de 3 literais sobre um conjunto de variáveis $V = \{x_1, x_2, \ldots, x_n\}$, determinar uma valoração $v\colon V \to \{0,1\}$ que satisfaça o maior número possível de cláusulas. Esse é um problema de otimização NP-difícil.

Dado	: cláusulas $\mathcal{C} = \{C_1, \ldots, C_k\}$ em 3-CNF sobre variáveis $V = \{x_1, \ldots, x_n\}$.
Devolve:	$v\colon V \to \{0,1\}$ que satisfaça o máximo de cláusulas de $\mathcal{C}$.

Problema MAX 3-SAT.

Exemplo 56. Por exemplo, $\mathcal{C} = \big\{\{x_1, \overline{x_2}, x_3\}, \{x_2, \overline{x_3}, x_4\}, \{x_3, \overline{x_4}, x_1\}\big\}$, é uma instância de 3-SAT que corresponde a fórmula booleana

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_4} \vee x_1).$$

No que segue, vamos supor que as cláusulas não tenham variáveis repetidas e vamos denotar por $\mathrm{opt} = \mathrm{opt}(\mathcal{C})$ o valor ótimo do problema de otimização descrito acima com instância $\mathcal{C}$.

Sejam $v \in_R \{0,1\}^n$ e $X_i(v) = 1$ se a cláusula C_i foi satisfeita por v, caso contrário $X_i(v) = 0$, então $X(v) = \sum_i X_i(v)$ é o número de cláusulas satisfeitas pela valoração v. Agora, para todo i, $X_i \sim \mathrm{Be}(1 - \frac{1}{8})$, pois uma cláusula não é satisfeita se cada uma de seus literais não o for, o que ocorre com probabilidade $1/8$, e $X \sim \mathrm{Bi}(k, 7/8)$. Pelo exercício 49, $\mathbb{E}\, X = 7k/8$ e como o valor ótimo não excede k temos o seguinte resultado.

Teorema 57. *O valor esperado do número de cláusulas satisfeitas numa valoração aleatória das variáveis de uma fórmula* 3-SAT *é pelo menos* $\frac{7}{8}\mathrm{opt}$. □

Corolário. *Para toda instância de* 3-SAT*, existe uma valoração que satisfaz pelo menos* $\frac{7}{8}$ *de todas as cláusulas.*

Demonstração. Se a média é $7k/8$ então deve haver uma valoração que que satisfaz $7/8$ das cláusulas. □

Como no máximo $1/8$ das cláusulas não é satisfeita por uma valoração ótima, temos que toda instância de 3-SAT com no máximo 7 cláusulas é satisfatível. Disso tudo, podemos tirar o seguinte algoritmo aleatorizado para esse problema.

Algoritmo 9: Algoritmo aproximativo para MAX 3-SAT.

Dado : cláusulas $\mathcal{C} = \{C_1, \ldots, C_k\}$ em 3-CNF sobre variáveis $V = \{x_1, \ldots, x_n\}$.

Devolve: uma valoração que satisfaz $\geqslant \frac{7}{8}k$ cláusulas.

repita
  para cada $i \in \{1, \ldots, n\}$ **faça** $x_i \leftarrow_R \{0, 1\}$;
  avalie $C_1, \ldots, C_k$;
até que *pelo menos* $\frac{7}{8}k$ *cláusulas estejam satisfeitas* ;
devolva $x_1, \ldots, x_n$.

Agora, qual é o valor esperado do número de valorações aleatórias até descobrir um que satisfaça pelo menos $\frac{7}{8}k$ cláusulas? Seja Z o número de rodadas do **repita** até que uma valoração que satisfaça a condição de parada é descoberta. Cada iteração é um experimento de Bernoulli com probabilidade de sucesso p então $Z \sim \mathrm{Be}(p)$.

Para estimar p, seja p_j é a probabilidade de uma valoração aleatória satisfazer exatamente j cláusulas, então $\mathbb{E}\, X = \sum_{j=0}^{k} j p_j$ e

$$\frac{7k}{8} = \sum_{j=0}^{k} j p_j = \sum_{j<\frac{7}{8}k} j p_j + \sum_{j\geqslant\frac{7}{8}k} j p_j \leqslant \sum_{j<\frac{7}{8}k} \left\lfloor \frac{7}{8}k \right\rfloor p_j + \sum_{j\geqslant\frac{7}{8}k} j p_j \leqslant \left\lfloor \frac{7}{8}k \right\rfloor (1-p) + kp,$$

portanto $kp \geqslant 7k/8 - \lfloor 7k/8 \rfloor \geqslant 1/8$, logo $p \geqslant 1/8k$. Como Z tem distribuição geométrica $\mathbb{E}\, Z \leqslant 8k$.

Para a análise da complexidade do algoritmo, vamos supor que a avaliação de cada cláusula pode ser feita em tempo constante. Assim, cada iteração do **repita** tem custo $O(k+n)$, portanto, esse algoritmo tem tempo esperado $8kO(k+n) = O(k^2+kn)$. Mais que isso pode ser dito nesse caso, a probabilidade de uma execução exceder o tempo esperado é pequena. De fato, pelo exercício 53 a probabilidade do número de iterações ultrapassar duas vezes o valor esperado é

$$\mathbb{P}\,[\, Z \geqslant 16k \,] \leqslant \left(\frac{8k-1}{8k}\right)^{16k-1} < 0,17$$

para todo $k \geqslant 1$, a probabilidade do número de iterações ser da ordem de k^2 é

$$\mathbb{P}\left[\, Z \geqslant k^2 \,\right] \leqslant 1,5414013468218464 \times 10^{-6}$$

para todo $k \geqslant 107$, ou seja, *muito* raramente o tempo de execução é da ordem de k^3. A probabilidade de escolher aleatoriamente de um baralho comum (52 cartas) as 5 cartas mais altas de um mesmo naipe é $1,5390771693292702 \times 10^{-6}$.

Exercício 58. O **MAX SAT** (*Maximum Satisfiability Problem*) é o problema: dado uma fórmula booleana $(C_1 \wedge \cdots \wedge C_k)$ com cada cláusula C_i sendo uma conjunção de literais sobre um conjunto de variáveis $V = \{x_1, x_2, \ldots, x_n\}$, determinar uma valoração $\nu\colon V \to \{0,1\}$ que satisfaça o maior número possível de cláusulas. Esse também é um problema de otimização NP-difícil. Prove que todo fórmula booleana $C_1 \wedge \cdots \wedge C_k$ admite uma valoração que satisfaz metade das cláusulas. Derive desse fato um algoritmo que recebe uma fórmula SAT e devolva uma valoração que satisfaça metade das cláusulas. Analise seu algoritmo de modo análogo às análises feitas para o algoritmo 9.

1.4.3 *Quicksort*

Considere a versão probabilística do *Quicksort* apresentada na introdução, algoritmo 3, página 11. Vamos mostrar que o número esperado de comparações entre elementos de S, denotado por X, é $O(n \log n)$ com alta probabilidade. Para tal, vamos supor que na entrada não há repetição, ou seja, todos os elementos de S são distintos.

O particionamento de S (linhas 1, 2, 3 e 4) é considerado um *sucesso* se $\min\{|S_1|, |S_2|\} \geqslant (1/8)|S|$. Caso contrário, dizemos que o particionamento foi um fracasso. A probabilidade de sucesso é de pelo menos $3/4$ e a de fracasso é de no máximo $1/4$.

Seja X_i o número de comparações entre o elemento x_i da entrada e algum pivô durante uma execução. Cada elemento x_i participará de no máximo $\lceil \log_{8/7} n \rceil$ particionamentos com sucesso e em cada particionamento é comparado uma única vez (com o pivô); note que se x_i é escolhido pivô então ele nunca mais participará de um particionamento e nunca mais será comparado com outro elemento de S. Assim, o número total de comparações durante uma execução é $X = \sum_{i=1}^{n} X_i$.

Fixe $i \in \{1, 2, \ldots, n\}$. Se Y_k é o número de particionamentos ocorridos entre o k-ésimo particionamento com sucesso (exclusive) do qual x_i participa e o próximo particionamento com sucesso (inclusive) do qual x_i participa, então $Y_k \sim \mathrm{Ge}(p)$,

onde $p \geqslant 3/4$, logo $\mathbb{E}\, Y_k \leqslant 4/3 < 2$. Temos que

$$X_i = \sum_{k=1}^{\lceil \log_{8/7} n \rceil} Y_k.$$

Pela linearidade da esperança $\mathbb{E}\, X_i < 2 \lceil \log_{8/7} n \rceil$, portanto $\mathbb{E}\, X < 2n \lceil \log_{8/7} n \rceil = O(n \log n)$.

Com isso, provamos que o número esperado de comparações numa execução do algoritmo é $O(n \log n)$. É sabido, da Análise de Algoritmos, que todo algoritmo de ordenação baseado em comparação faz no mínimo da ordem de $n \log n$ comparações para ordenar n elementos, ou seja, em média o *Quicksort* aleatorizado é o melhor possível. Porém, saber que um algoritmo é bom em média, não nos garante que ele é bom quase-sempre. Vamos mostrar que com alta probabilidade o desempenho do algoritmo está próximo da média.

Consideremos o evento $X_i > \lceil 8 \log_{8/7} n \rceil$, ou seja, x_i participou com mais que quatro vezes o valor esperado para o número de comparações com x_i realizadas pelo algoritmo. Notemos que $X_i > \lceil 8 \log_{8/7} n \rceil$ se em $\lceil 8 \log_{8/7} n \rceil$ particionamentos ocorrem menos que $\lceil \log_{8/7} n \rceil$ sucessos, ou seja, o número de fracassos em $\lceil 8 \log_{8/7} n \rceil$ particionamentos é maior que $\lceil 8 \log_{8/7} n \rceil - \lceil \log_{8/7} n \rceil$.

Seja Z_i o número de particionamentos com fracasso pelos quais x_i passa em $\lceil 8 \log_{8/7} n \rceil$ particionamentos, $Z_i \sim \mathrm{Bi}(\lceil 8 \log_{8/7} n \rceil, 1-p)$ com $p \geqslant 3/4$. Vamos estimar a probabilidade do evento $Z_i > \lceil 8 \log_{8/7} n \rceil - \lceil \log_{8/7} n \rceil$.

De $\lceil 8 \log_{8/7} n \rceil - \lceil \log_{8/7} n \rceil > 8 \log_{8/7} n - (\log_{8/7}(n) + 1)$ temos

$$\begin{aligned}
\mathbb{P}\left[Z_i > \lceil 8 \log_{8/7} n \rceil - \lceil \log_{8/7} n \rceil\right] &< \mathbb{P}\left[\, Z_i \geqslant 7 \log_{8/7} n \,\right] \\
&\leqslant \sum_{j \geqslant \lceil 7 \log_{8/7} n \rceil} \binom{\lceil 8 \log_{8/7} n \rceil}{j} (1-p)^j \, p^{\lceil 8 \log_{8/7} n \rceil - j} \\
&\leqslant \sum_{j \geqslant \lceil 7 \log_{8/7} n \rceil} \binom{\lceil 8 \log_{8/7} n \rceil}{j} \left(\frac{1}{4}\right)^j \\
&\leqslant \sum_{j \geqslant \lceil 7 \log_{8/7} n \rceil} \left(\frac{e \lceil 8 \log_{8/7} n \rceil}{4j}\right)^j \\
&\leqslant \sum_{j \geqslant \lceil 7 \log_{8/7} n \rceil} \left(\frac{e \lceil 8 \log_{8/7} n \rceil}{4 \lceil 7 \log_{8/7} n \rceil}\right)^j
\end{aligned}$$

mas

$$\frac{e \lceil 8 \log_{8/7} n \rceil}{4 \lceil 7 \log_{8/7} n \rceil} < 0,7 \frac{\lceil 8 \log_{8/7} n \rceil}{\lceil 7 \log_{8/7} n \rceil} < 0,7 \left(\frac{8}{7} + \frac{1}{7 \log_{8/7} n}\right)$$

que é menor que $7/8$ para todo $n \geqslant 2$. Assim,

$$\sum_{j \geqslant \lceil 7 \log_{8/7} n \rceil} \left(\frac{e \lceil 8 \log_{8/7} n \rceil}{4 \lceil 7 \log_{8/7} n \rceil} \right)^j \leqslant \sum_{j \geqslant \lceil 7 \log_{8/7} n \rceil} \left(\frac{7}{8} \right)^j = O(n^{-7}).$$

portanto

$$\mathbb{P}\left(\bigcup_{i=1}^{n} \left[X_i > 8 \log_{8/7} n \right] \right) = O(n^{-6}),$$

logo, com probabilidade $1 - O(n^{-6})$ vale que $X_i \leqslant 8 \log n$ para todo i e que, com essa probabilidade, *Quicksort* aleatorizado executa $\leqslant 8n \log n$ comparações entre elementos da entrada.

Observação 59. Não há nada de especial na escolha de $1/8$. Sabe-se que se uma proporção for mantida no particionamento, digamos que a menor parte tem uma fração α da entrada, então a recorrência $t(n) \leqslant t(\lfloor \alpha n \rfloor) + t(\lfloor (1-\alpha) n \rfloor) + cn$ para constantes $\alpha, c > 0$ tem solução $O(n \log n)$.

Análise de caso médio.

A seguir apresentamos um esboço da análise de caso médio do *Quicksort* determinístico (com pivô fixo, digamos na primeira posição). Consideramos que a seqüência de entrada é uma seqüência de elementos distintos com todas as permutações desses elementos sendo equiprováveis. Denote por C_n o número médio de comparações do *Quicksort* sobre uma entrada de tamanho n. A probabilidade do particionamento ocorrer no k-ésimo elemento da seqüência ordenada é $1/n$ e o número de comparações é

$$C_n = \frac{1}{n} \sum_{i=1}^{n} (n - 1 + C_{k-1} + C_{n-k}) = n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} C_i.$$

para todo $n > 1$ e $C_1 = 0$. Podemos ainda simplificar essa recorrência e chegar em

$$C_n = \frac{(n+1)C_{n-1} + 2(n-1)}{n}$$

para $n > 2$ e $C_2 = 1/2$. Agora, pode-se provar (por indução) que $C_n = \Theta(n \log n)$.

Note que a média é feita considerando-se a distribuição de probabilidade nas entradas do algoritmo (não há sorteio de bits).

1.4.4 Gerador de prováveis primos

Vamos considerar o problema de escolher um número primo em $\{n+1,\ldots,2n\}$ para um dado $n \in \mathbb{N}$. Para esse problema não se conhece um algoritmo determinístico de tempo subexponencial. O postulado de Bertrand garante que existe pelo menos um primo nesse intervalo, entretanto, da prova de Chebyshev para esse postulado decorre que

$$\frac{1}{3}\frac{n}{\log n} < \pi(2n) - \pi(n) < \frac{7}{5}\frac{n}{\log n} \tag{1.20}$$

onde $\pi(n)$ é a quantidade de primos menores ou iguais a n (Ribenboim, 1996). Um sorteio uniforme em $\{n+1,\ldots,2n\}$ tem probabilidade $\rho = (\pi(2n) - \pi(n))/n$ com

$$\frac{1}{3\log n} < \rho < \frac{7}{5\log n} \tag{1.21}$$

de resultar um primo.

Veremos a seguir um algoritmo aleatorizado de tempo esperado polinomial.

Algoritmo 10: Gerador de números primos aleatórios.

Dado : inteiro positivo n.

Devolve: um primo escolhido uniformemente em $\{n+1,\ldots,2n\}$.

```
1 repita
2     m ←_R {n+1, n+2, ..., 2n};
3 até que Teste_Primo(m) ;
4 devolva m
```

Primeiro consideraremos que Teste_Primo(m) é uma algoritmo determinístico que decide se m é primo. Notemos que nesse caso temos uma distribuição de probabilidades no conjunto das instâncias do algoritmo, aqui Teste_Primo recebe uma entrada com probabilidade uniforme (veja a análise do caso médio do *Quicksort* na seção anterior).

O espaço amostral é composto pelas seqüências $\omega = (m_1, m_2, \ldots, m_p)$ tais que m_i é composto para $1 \leqslant i < p$ e m_p é primo. Denote por I o número de iterações do **repita**, ou seja, $I(\omega)$ é o comprimento de ω. Essa variável I tem distribuição geométrica com probabilidade de sucesso igual a probabilidade de $\rho = \mathbb{P}_{m\in_R\{n+1,\ldots,2n\}}[\, m \text{ é primo }]$, portanto, o número esperado de iterações é linear no tamanho de n

$$\mathbb{E}\, I = \frac{1}{\rho} = \Theta(\log n). \tag{1.22}$$

Denote por $TP(m)$ o tempo de execução do algoritmo de teste de primalidade de para m. Assumindo que $TP(m)$ é uma função crescente em m podemos limitar o

tempo dos testes por $TP(2n)$ e o tempo esperado do algoritmo fica estimado por

$$O\left(\log(n) + TP(2n)\right) O(\log n) \tag{1.23}$$

que podemos assumir ser $O(\log(n)TP(2n))$ já que é razoável supor que testar primalidade custa mais que gerar o número testado.

Exemplo 60. No caso do AKS, algoritmo de teste de primalidade citado na introdução, $TP(n) = O(\log^7 n)$, portanto $\mathbb{E}\, T = O(\log^8 n)$, i.e., o tempo esperado é polinomial, com polinômio de grau 8.

Considerando a distribuição de probabilidades no conjunto das instâncias do algoritmo Teste_Primo, o tempo esperado para testar primalidade é

$$\mu(n) = \sum_{m=n+1}^{2n} TP(m)\mathbb{P}(m) = \frac{\sum_{m=n+1}^{2n} TP(m)}{n}. \tag{1.24}$$

Agora, se T_i é o tempo da i-ésima iteração, então $\mathbb{E}\, T_i = E[T_i|I \geqslant i]\mathbb{P}[\, I \geqslant i\,]$e usando (1.24)

$$\mathbb{E}[T_i|I \geqslant i] = O\left(\mu(n)\right)$$

logo, a partir do exercício 52 deduzimos que

$$\mathbb{E}\, T = \sum_{i\geqslant 1} \mathbb{E}\, T_i = O\left(\mu(n)\right) \sum_{i\geqslant 1} \mathbb{P}\left[\, I \geqslant i\,\right] = O\left(\mu(n)\right) \mathbb{E}\, I = O\left(\mu(n)\log(n)\right).$$

Exercício 61. Prove que no caso do AKS vale $\mu(n) = \Omega(\log^7 n)$ (portanto, $\mu(n) = \Theta(\log^7 n)$), logo $\mathbb{E}\, T = O(\log^8 n)$ como foi estimado considerando o pior caso do AKS.

Exercício 62. Mostre que m, a resposta dada pelo algoritmo, é uma variável aleatória com distribuição uniforme.

Agora, suponha que Teste_Primo(n) é um algoritmo aleatorizado que recebe um natural, devolve *falso* se n é composto, senão devolve *verdadeiro* e nesse caso n é provavelmente primo, ou seja, o algoritmo erra declarando algumas vezes que um número composto é primo e nunca ao contrário.

Como há falsos positivos, a probabilidade de Teste_Primo(m) resultar verdadeiro é maior ou igual a probabilidade de m ser de fato um número primo, ou seja, essa probabilidade é pelo menos ρ, logo o número esperado de rodadas é

$$\mathbb{E}\, I < 3\log n.$$

O tempo médio para testar primalidade tomado sobre as escolhas $m \in_R \{n+1, \ldots, 2n\}$ e sobre os sorteios feitos por $TP(m)$ é

$$\mu(n) = \frac{\sum_{m=n+1}^{2n} \mathbb{E}\, TP(m)}{n}$$

portanto o tempo esperado do algoritmo é $O(\log(n)\mu(n))$.

Para estimar a probabilidade do algoritmo responder errado, suponhamos que que Teste_Primo(m) responda errado com probabilidade no máximo ε, para algum $\varepsilon > 0$.

Seja C o evento "m é composto", o qual ocorre com probabilidade $\gamma = 1 - \rho$, e E o evento "Teste_Primo(m) declara m primo". A probabilidade do algoritmo 10 devolver um número composto é, usando o teorema de Bayes,

$$\mathbb{P}(C|E) = \frac{\mathbb{P}(E|C)\mathbb{P}(C)}{\mathbb{P}(E|C)\mathbb{P}(C) + \mathbb{P}(E|\overline{C})\mathbb{P}(\overline{C})} \leqslant \frac{\varepsilon\gamma}{\varepsilon\gamma + \rho}.$$

Teorema 63. *No algoritmo 10, se ρ é a densidade de primos em $\{n+1, \ldots, 2n\}$ e ε é a probabilidade de* Teste_Primo(m) *errar, então a probabilidade da resposta ser um número composto é*

$$\leqslant \frac{\varepsilon(1-\rho)}{\varepsilon(1-\rho) + \rho}.$$

□

Observação 64. Na prática, após sortear m é mais eficiente testar divisibilidade de m por inteiros primos até uma constante k antes de testar primalidade, pois é relativamente grande a quantidade de números que tem um divisor pequeno. De imediato números pares podem ser descartados. Testar divisibilidade pelos primos até 256 descarta 80% dos números ímpares com k bits candidatos a primo (Menezes, van Oorschot, and Vanstone, 1997). Veremos mais sobre esse problema na seção 6.6.4.

Exercícios complementares

1. Qual é a probabilidade de que se distribuímos uniformemente n bolas idênticas em n caixas distintas nenhuma fica vazia? E de exatamente uma ficar vazia?

2. Numa cidade com $n+1$ habitantes, uma pessoa conta um boato para uma segunda pessoa, que por sua vez conta o boato para uma terceira pessoa e

assim por diante. Em cada passo o ouvinte do boato é escolhido aleatoriamente de maneira uniforme dentre as n pessoas restante na cidade. Qual a probabilidade do boato ser contado r vezes sem repetir nenhuma pessoa?

3. Prove que se dois números, a e b, são escolhidos aleatoriamente no conjunto $\{1, 2, \ldots, n\}$ então $\mathbb{P}[\ \mathrm{mdc}(a, b) = 1\ ] \geqslant 1/4$.

4. Seja Ω o conjunto dos inteiros positivos e defina para todo $E \subseteq \Omega$ e cada $n \geqslant 1$

$$\mathbb{P}_n(E) = \frac{|\{1, 2, \ldots, n\} \cap E|}{n}.$$

Prove que $\mathbb{P}_n$ é uma função de probabilidade.

Defina $p(E) = \lim_{n\to\infty} \mathbb{P}_n(E)$ quando o limite existe e seja $\mathcal{P}$ a família de eventos para os quais o limite existe. Prove que se A e B são eventos disjuntos de $\mathcal{P}$ então $A \cup B \in \mathcal{P}$ e $p(A \cup B) = p(A) + p(B)$ e que esse não é o caso se os eventos não são disjuntos ou se temos uma seqüência enumerável infinita deles. Finalmente, prove que p é invariante pro translação, i.e., se $p(A)$ existe então $p(\{a + 1 : a \in A\}) = p(A)$.

5. (Emparelhamento perfeito em grafos) Seja $G = (V, E)$ um grafo bipartido e defina a matriz anti-simétrica $A = (a_{i,j})$ pondo

$$a_{i,j} = \begin{cases} x_{i,j} & \text{se } \{i, j\} \in E \text{ e } i < j \\ -x_{i,j} & \text{se } \{i, j\} \in E \text{ e } i > j \\ 0 & \text{caso contrário.} \end{cases}$$

Prove que G tem um emparelhamento perfeito se e somente de $\det(A) \neq 0$. Escreva um algoritmo aleatorizado que decide se um grafo bipartido tem emparelhamento perfeito. Analise a probabilidade de erro do algoritmo.

6. Dois amigos querem decidir quem pagará a conta do restaurante com uma aposta. Cada um deles escolhe uma seqüência de três caras ou coroas e eles jogam uma moeda até que saia uma das duas seqüências: aquele que tiver escolhido a primeira seqüência a sair ganhou a aposta. Por exemplo, André é o primeiro e fica com a seqüência coroa, coroa, cara enquanto Renato responde com cara, coroa, coroa. Eles jogam a moeda obtendo coroa, cara, coroa, cara, coroa, cara, coroa, coroa e neste momento Renato é o vencedor. Mostre que nesse caso a probabilidade do Renato ganhar o jogo é $3/4$.

Mostre que o segundo jogador sempre tem uma escolha mais vantajosa pra ele.

7. Suponha que você tem três moedas e uma delas é viciada, $\mathbb{P}(\text{cara}) = 2/3$. Escolhendo uma delas ao acaso a probabilidade de acertar qual é a viciada é um terço. Agora, suponha que o resultado do lançamento de cada uma delas é cara, cara, coroa. Mostre, usando a lei de Bayes, que a probabilidade da primeira moeda ser a viciada é $2/5$.

8. Vamos gerar um subconjunto de $\{1, 2, \ldots, n\}$ da seguinte forma: para cada elemento lançamos uma moeda honesta, se o resultado for cara colocamos esse elemento no subconjunto, caso contrário não colocamos. Mostre que o subconjunto resultante é com igual probabilidade qualquer subconjunto de $\{1, 2, \ldots, n\}$.

   Suponha que escolhemos dois subconjuntos X e Y de modo uniforme e independente dentre todos os 2^n subconjuntos de $\{1, 2, \ldots, n\}$. Determine $\mathbb{P}(X \subseteq Y)$ e $\mathbb{P}[\, X \cup Y = \{1, 2, \ldots, n\}\,]$.

9. Prove que se $X \sim \mathrm{Ge}(p)$ e $n > 0$ então

$$\mathbb{P}[\, X = n + t | X > t\,] = \mathbb{P}[\, X = n\,].$$

10. Sejam X e Y variáveis aleatórias geométricas independentes e com parâmetro p e q, respectivamente. Determine

    (a) $\mathbb{P}[\, X = Y\,]$.

    (b) $\mathbb{E}\,\max(X, Y)$.

    (c) $\mathbb{P}[\, \min(X, Y) = k\,]$.

    (d) $\mathbb{E}[X | X \leq Y]$.

11. Prove que $(\mathbb{E}\,XY)^2 \leqslant \mathbb{E}\,X^2\,\mathbb{E}\,Y^2$.

12. Seja Y uma variável aleatória que assume valores em $\mathbb{N}$ e com esperança positiva. Prove que

$$\frac{(\mathbb{E}\,Y)^2}{\mathbb{E}\,Y^2} \leqslant \mathbb{P}[\, Y \neq 0\,] \leqslant \mathbb{E}\,Y.$$

13. Seja $a_1, a_2, \ldots, a_n$ uma permutação de $1, 2, \ldots, n$. Determine o número esperado de inversões, isto é, pares $i < j$ tais que $a_i > a_j$.

14. Uma permutação pode ser vista como uma função $\pi\colon \{1,\ldots,n\} \to \{1,\ldots,n\}$: $\pi(i)$ é o elemento da i-ésima posição. Determine o valor esperado de pontos x tais que $\pi(x) = x$ quando escolhemos uma permutação de maneira uniforme.

    Uma permutação pode ser representada da seguinte forma por um grafo orientado: a cada elemento de $\{1,\ldots,n\}$ corresponde um vértice e (i,j) é uma aresta de i para j se $\pi(i) = j$. As seqüências de vértices que formam um circuito orientado são chamadas de *ciclos* da permutação e, portanto, todo permutação pode ser representada por um conjunto de ciclos. Determine o número esperado de ciclos de uma permutação escolhida uniformemente.

15. Projete um algoritmo aleatorizado, Seleção(n,k), que recebe uma lista de n elementos de um tipo ordenado e devolve o k-ésimo maior elemento da lista. O número esperado de comparações deve ser $O(n)$. Determine a probabilidade com que o algoritmo faz mais que $n \lg n$ comparações.

16. Escreva um algoritmo aleatorizado eficiente que recebe um inteiro $M \geqslant 2$ e devolve $N \in \{0,1,\ldots,M-1\}$ tal que $\mathrm{mdc}(M,N) = 1$.

17. Considere o seguinte gerador de números aleatórios

    Dado : inteiro positivo $M > 0$.
    Devolve: um inteiro escolhido uniformemente em $\{0,1,\ldots,M-1\}$.

    1 seja k o número de bits de M;
    2 sorteie $(d_0, d_1, \ldots, d_{k+t-1}) \in_R \{0,1\}^{k+t}$;
    3 $N \leftarrow \sum_i d_i 2^i$;
    4 **devolva** $N \mod M$.

    Vimos que no caso $t = 0$ a distribuição da resposta não é uniforme (exercício 23). Prove que quanto maior é t mais próximo a distribuição de N é da uniforme, no seguinte sentido

    $$\sum_{0 \leqslant n < M} \left| \mathbb{P}\,[\, N = n \,] - \frac{1}{M} \right| \leqslant \frac{1}{2^{t-1}}.$$

18. Suponha que Teste_Primo(n) é um algoritmo aleatorizado que recebe um natural $n \in D$ e devolve *composto* ou *provavelmente primo*, ou seja, se o algoritmo devolve provavelmente primo significa que a entrada é um número primo com probabilidade positiva (fixa), mas pode ser que seja composto. Suponha que o algoritmo erra com probabilidade no máximo $1/100$ e que

10% dos naturais em D são primos. Se Teste_Primo(n) resultou verdadeiro, qual a probabilidade de n ser primo?

CAPÍTULO 2

Computação probabilística

Conteúdo

2.1 Modelos de Computação

O objetivo da Teoria da Complexidade Computacional é a análise de recursos computacionais envolvidos na solução de problemas computacionais, por isso se faz necessário modelos abstratos que capturem aspectos práticos da computação para a total compreensão das limitações do modelo computacional e da dificuldade inerente de se resolver poblemas computacionais algoritmicamente.

2.1.1 Máquinas de Turing

Máquina de Turing é um modelo de computação que, juntamente com o λ-cálculo de Church, nasceu por volta de 1936 pra formalizar o que intuitivamente os matemáticos chamavam de algoritmo. Esses dois modelos são equivalentes no sentido de que tudo que pode ser computado num, também pode ser no outro e vice-versa.

Uma *máquina de Turing determinística* é uma sêxtupla $(Q, \Sigma, \Gamma, q_0, \delta, F)$, onde Q é o conjunto de *estados* da máquina, Σ é um *alfabeto* finito o qual podemos e vamos assumir, sem perda de generalidade, ser $\{0, 1\}$, Γ é o *alfabeto da fita* que contém Σ e um símbolo especial $\flat$ que representa *espaço em branco*, q_0 é o *estado inicial*, δ é a *função de transição* definida em $Q \times \Gamma$ e que assume valores em $(Q \cup F) \times \Gamma \times \{\leftarrow, —, \rightarrow\}$ e F é o conjunto $\{q_{\text{aceita}}, q_{\text{rejeita}}, q_{\text{pára}}\}$ dos *estados finais*, disjunto de Q. Denotamos por Σ^* o conjunto de todas as seqüências de símbolos do alfabeto Σ.

Uma *computação* numa máquina de Turing determinística com uma *entrada* $w_1 w_2 \ldots w_n \in \Sigma^*$ que está escrita na primeiras n posições de uma fita que é infinita para a direita e tem nas outras posições o símbolo $\flat$, começa com a máquina no estado q_0 e lê w_1 na posição da fita mais a esquerda. Se a máquina está num estado $q \in Q$ e lê $\gamma \in \Gamma$ na fita então o próximo passo é determinado por $\delta(q, \gamma) = (q', \gamma', \ell)$ onde $q' \in Q$, $\gamma' \in \Gamma$ é o símbolo que a máquina deixa na posição que estava o símbolo γ e $\ell \in \{\leftarrow, —, \rightarrow\}$ diz qual símbolo é o próximo a ser lido, o que está a esquerda do atual, o atual ou o símbolo a direita do atual e a computação continua a partir de $\delta(q', \gamma')$. Se M é uma máquina de Turing determinística e $w \in \Sigma^*$ uma entrada para M, a computação de M com w pode

- terminar em q_{aceita}, nesse caso escrevemos $M(w) = 1$ e dizemos que a máquina M *aceita* a entrada w,

- terminar em q_{rejeita}, nesse caso escrevemos $M(w) = 0$ e dizemos que a máquina M *rejeita* a entrada w,

- terminar em $q_{\text{pára}}$, nesse caso escrevemos $M(w) = y$ onde y é a seqüência de símbolos na fita desde a posição mais a esquerda até a posição antes do primeiro $\flat$,

- não terminar.

Se $L \subseteq \Sigma^*$ é uma linguagem e M uma máquina de Turing determinística, então dizemos que M **decide** L (ou seja, decide pertinência na linguagem L) se

1. $w \in L$ implica em M aceita w;

2. $w \notin L$ implica em M rejeita w.

Ainda, se denotamos por L também a função característica[1] de L, então L é decidida pela máquina de Turing determinística M se para todo $w \in \Sigma^*$ vale que $M(w) = L(w)$.

Uma *máquina de Turing não-determinística* é uma sêxtupla $(Q, \Sigma, \Gamma, q_0, \delta, F)$ como acima a menos da função de transição que é definida em $Q \times \Gamma$ e assume valores no conjunto das partes $\mathcal{P}((Q \cup F) \times \Gamma \times \{\leftarrow, —, \rightarrow\})$ e associa a cada par (q, γ) do domínio um elemento da forma $\{(q_1, \gamma_1, m_1), \ldots, (q_t, \gamma_t, m_t)\}$ no contra-domínio, para algum $t = t(q, \gamma) \geqslant 1$. A computação numa máquina não-determinística é uma árvore cujas ramificações correspondem aos diferentes elementos da imagem da função de transição num ponto. Se com entrada $w \in \Sigma^*$ algum ramo da árvore

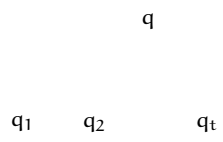

Figura 2.1: ramificação.

correspondente à máquina M leva ao estado q_{aceita} então dizemos que a máquina M **aceita** a entrada w. Dizemos que M **decide** a linguagem $L \subset \Sigma^*$ quando para todo $w \in \Sigma^*$ vale que $w \in L$ se e somente se M aceita w.

O poder computacional dessas máquinas, determinística e não-determinística, é o mesmo no seguinte sentido.

Exercício 65. Prove que para toda máquina de Turing não-determinística pode ser simulada por uma máquina de Turing determinística.

[1] A *função característica de um conjunto* A é a função $f\colon A \to \{0, 1\}$ tal que $f(x) = 1$ se e somente se $x \in A$.

2.1.2 Circuitos booleanos.

Máquina de Turing é um modelo *uniforme* de computação, num modelo *não-uniforme* temos um dispositivo de computação para cada tamanho possível de instância; o modelo não-uniforme mais conhecido é o circuito booleano. Entre modelos uniformes e não uniformes não há equivalência computacional, A não uniformidade dos circuitos faz com que ele computem funções que não são computáveis por Máquinas de Turing, pois há funções booleanas que não são computáveis por máquinas de Turing (veja exercício 67) e são computáveis por circuitos.

Um **circuito booleano** é um grafo orientado acíclico onde cada vértice está associado ou a uma porta lógica dentre *e*, *ou*, *não*, ou a um valor lógico dentre 0 e 1, ou a uma variável x_i para $i \in \{1, 2, \ldots, n\}$ para algum $n \in \mathbb{N}$, ou uma saída s_i, $i \in \{1, 2, \ldots, m\}$ para algum $m \in \mathbb{N}$. O grau de entrada dos vértices pode assumir um de três valores: 0 (variáveis e valores lógicos), 1 (porta *não* e saídas) e 2 (portas *e* e *ou*). Um circuito computa uma função booleana, *que denotaremos por* C *também*, $C\colon \{0,1\}^n \to \{0,1\}^m$ dada por $C(x_1, x_2, \ldots, x_n) = (s_1, s_2, \ldots, s_m)$.

Se L é uma linguagem, então uma família de circuitos $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ **decide** L se para cada $x = x_1x_2\ldots x_n \in \Sigma^*$ o circuito C_n com entradas $x_1x_2\ldots x_n$ responde 1 se e somente se $x \in L$, em outras palavras, se denotamos também por L a função característica $L\colon \{0,1\}^* \to \{0,1\}$ do conjunto L, temos que $C_n(x) = L(x)$.

Exemplo 66. O circuito na figura abaixo computa a paridade de $x = x_1x_2x_3x_4$, ou seja, devolve 1 se e somente se o número de ocorrências de 1's em x é ímpar.

Exercício 67. Seja $I \subset \Sigma^*$ uma linguagem indecidível qualquer[2], defina

$$L = \{1^n\colon n \text{ em base 2 pertence a } I\}.$$

Prove que L é indecidível. Exiba uma família de circuitos $(C_n)_{n\in\mathbb{N}}$ com número de vértices polinomial em n e que decide pertinência em L.

Máquina de Acesso Aleatório — RAM.

O modelo uniforme que mais se aproxima da noção concreta de computador é o modelo RAM — *Random Acess Machine* — que tem uma memória principal, registradores (onde ocorrem operações aritméticas), instruções para transferência entre memória e registradores, de endereçamento, aritméticas (soma e subtração)

[2]Por exemplo, a linguagem dada pelos pares compostos por uma máquina de Turing determinística M e uma palavra w tais que M com entrada x termina (problema da Parada).

e de desvios condicionais (veja Papadimitriou (1994) seção 2.6 para uma descrição detalhada). A principal diferença é que uma máquina RAM tem memória ilimitada, enquanto que um computador tem memória limitada.

Algoritmos.

Máquinas de Turing que não terminam são úteis para classificar problemas computacionais mas não são dispositivos úteis de computação já que nunca sabemos até quando devemos esperar para saber o resultado da computação. No restante desse capítulo estaremos interessados somente em máquinas que terminam a computação , portanto, são modelos para o que conhecemos intuitivamente como algoritmo. Assim, de agora em diante chamamos de **algoritmo** máquinas de Turing que terminam. O leitor não familiarizado com máquinas de Turing pode valer-se da equivalência computacional com o modelo RAM (veja a observação 70 abaixo) e trocar as ocorrências de Máquina de Turing por Algoritmo no sentido corriqueiro, escrito em linguagem alto-nível. A tradução de alto-nível para programa RAM não é um mistério, é essencialmente uma *compilação* no sentido usual da palavra (veja mais na seção 8.6 de Hopcroft, Motwani, and Ullman (2000)).

2.1.3 Algoritmos aleatorizados

Algoritmos aleatorizados são máquinas de Turing não-determinísticas (que sempre terminam), a diferença fundamental está na interpretação de uma computação.

Uma **máquina de Turing probabilística** M é uma máquina de Turing não-determinística em que cada passo não-determinístico tem dois movimentos legais, os quais correspondem ao sorteio de um bit. A diferença está na *definição de* aceite *por uma máquina probabilística*; ao invés de aceitar uma entrada quando existe um ramo que termina em q_{aceita}, a máquina probabilística aceita a entrada quando a "maioria ponderada" dos ramos terminam a computação em q_{aceita}, onde o peso de cada ramo é 2^{-s} e s é o número de nós não-determinísticos no ramo.

Uma máquina probabilística M **decide** uma linguagem $L \subseteq \Sigma^*$ com probabilidade de erro $0 < \varepsilon < 1/2$ se para todo $w \in \Sigma^*$ vale que

1. se $w \in L$, então $\mathbb{P}[\ M \text{ aceita } w\ ] \geqslant 1 - \varepsilon$, e
2. se $w \notin L$, então $\mathbb{P}[\ M \text{ aceita } w\ ] < \varepsilon$;

onde $\mathbb{P}$ é definida a seguir. Dada uma entrada w para M, um ramo r que aceita w tem probabilidade associada $(\frac{1}{2})^k$, onde k é o número de transições não determinísticas (sorteios de bit) do ramo. Para simplificar, podemos assumir que em cada transição ocorre um sorteio e escrevemos $\mathbb{P}[\text{ ramo } r\ ] = (\frac{1}{2})^{|r|}$, onde $|r|$ é o número de transições no ramo r; os bits gerados e não utilizados são ignorados. A probabilidade com que M aceita w é a soma das probabilidades dos ramos que terminam em q_{aceita}, ou seja $\mathbb{P}[\ M \text{ aceita } w\ ] = \sum_r \mathbb{P}[\text{ ramo } r\ ]$ onde r é um ramo da computação que aceita w.

Exercício 68. Dada uma máquina de Turing probabilística e uma entrada w, seja S o conjunto de todas as seqüências binárias que correspondem as ramos da computação de M com w. Mostre que não há duas seqüências $a, b \in S$ tais que uma seja prefixo da outra. Use esse fato para mostrar que $\sum_{a \in S}(1/2)^{|a|} \leqslant 1$, onde $|a|$ denota o número de símbolos de a no alfabeto.

Um **algoritmo probabilístico** M decide uma linguagem L com probabilidade de erro $\epsilon < 1/2$ se para todo $w \in \Sigma^*$

1. se $w \in L$, então $\mathbb{P}[\ M \text{ aceita } w\ ] \geqslant 1 - \varepsilon$, e
2. se $w \notin L$, então $\mathbb{P}[\ M \text{ rejeita } w\ ] \geqslant 1 - \varepsilon$;

onde $\mathbb{P}[\ M \text{ rejeita } w\ ] = 1 - \mathbb{P}[\ M \text{ aceita } w\ ]$, equivalentemente

$$\mathbb{P}[\ M(x) = L(x)\ ] \geqslant 1 - \varepsilon.$$

Notemos que máquina não-determinística é uma abstração que é conveniente para capturar algumas características importantes de problemas computacionais mas que não reflete um modelo realista de computação o que não é o caso da máquina probabilística, que é um modelo concreto de computação.

Máquina de Turing probabilística *off-line*

Um modelo alternativo para algoritmos aleatorizados é uma máquina de Turing determinística que tem uma fita auxiliar só de leitura, além da fita da entrada. Essa fita auxiliar contém uma seqüência b de bits aleatórios, cada posição tem um bit com distribuição uniforme e as ocorrências na seqüência são independentes. Cada posição da fita auxiliar é usada (lida) uma única vez (Goldreich, 2008).

Assim, uma máquina M com entrada w computa com (w, b). Esse modelo é chamado em algumas referências bibliográficas de máquina de Turing probabilística *off-line*.

Esses dois modelos probabilísticos são equivalentes, ou seja, tudo o que é computado por um, também é computado pelo outro, e vice-versa.

2.2 Classes de complexidade

Uma classe de complexidade computacional, com respeito a um modelo de computação, é uma coleção de linguagens sobre o alfabeto $\Sigma = \{0, 1\}$ que são reconhecidas por um dispositivo computacional naquele modelo com alguma restrição de recurso, como tempo ou espaço por exemplo. No que segue, o **tamanho** de uma palavra $w \in \Sigma^*$ é a quantidade de símbolos de Σ em w, denotado por $|w|$.

Seja M uma máquina de Turing não-determinística que termina com qualquer entrada $w \in \Sigma^*$. O **tempo** de M para uma entrada w, denotado $t_M(w)$, é o número de transições no ramo mais longo da computação de M com a entrada w. Uma máquina de Turing tem tempo $T(n)$ se $t(w) \leqslant T(n)$ para toda entrada w de tamanho n. Uma **máquina de tempo polinomial** é uma máquina de tempo $T(n) = O(n^k)$, para algum $k \in \mathbb{N}$.

A classe **P** — *Polynomial time* — é a classe das linguagens decididas por máquina de Turing determinística de tempo polinomial.

A classe **NP** — *Nondeterministic Polynomial time* — é a classe das linguagens decididas por máquina de Turing não-determinística de tempo polinomial.

Exercício 69. Prove que a seguinte definição é equivalente a dada acima. Uma linguagem $L \subseteq \Sigma^*$ está em NP se existe uma máquina determinística M de tempo polinomial e um polinômio $p : \mathbb{N} \to \mathbb{N}$ tal que para todo $w \in \Sigma^*$

1. se $w \in L$ então existe $y \in \Sigma^*$, $|y| = p(|w|)$, tal que $M(w, y) = 1$;
2. se $w \notin L$ então para todo $y \in \Sigma^*$, $M(w, y) = 0$.

Observação 70. Usualmente, no modelo RAM o tempo é o número de instruções da RAM que o programa executa. Uma diferença importante quando analisamos algoritmos no modelo RAM ocorre quando assumimos custo constante para as operações. Em máquinas de Turing as mesmas operações têm custo proporcional ao logaritmo dos operandos. De fato, a literatura considera dois modelos RAM, o de *custo logarítmico* que leva em conta o custo das operações em função do tamanho dos operandos, e o modelo de *custo uniforme* com instruções em tempo constante (que é o mais comum quando analisamos algoritmos nas salas de aula), cujo tempo corresponde ao tempo no modelo de custo logarítmico vezes um fator logarítmico no tamanho da entrada. Máquina de Turing e máquina RAM com custo logarítmico são polinomialmente equivalentes, ou seja, um programa RAM de tempo polinomial no tamanho da entrada pode ser simulado por uma máquina de Turing de tempo polinomial no tamanho da entrada e uma máquina de Turing de tempo polinomial no tamanho da entrada pode ser simulado por um programa RAM de tempo polinomial no tamanho da entrada (teoremas 2.4 e 2.5 de Papadimitriou (1994)). Nos outros capítulos deste texto, supomos o modelo RAM de custo uniforme quando analisamos a complexidade de tempo de um algoritmo, e para nos mantermos honestos aos modelos tomamos o cuidado de o tamanho dos operandos serem limitados polinomialmente no tamanho da entrada e dos sorteios serem feitos num conjunto de tamanho polinomial no tamanho da entrada.

O **tamanho do circuito** C, denotado por $|C|$, é o número de vértices do grafo e, em geral, estamos interessado no tamanho dos circuitos de uma família $\mathcal{C}$ em função do tamanho da entrada. Um circuito C tem tamanho mínimo se qualquer circuito equivalente a C tem tamanho pelo menos $|C|$.

Seja $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ uma família de circuitos. Dizemos que $\mathcal{C}$ é uma família de **tamanho** $s(n)$ se $|C_n| = O(s(n))$. A função $f\colon \{0,1\}^* \to \{0,1\}^*$ tem **complexidade**

$s_f(n)$ se f pode ser computada por uma família de circuitos booleanos de tamanho mínimo de complexidade $s_f(n)$.

Um fato importante e que será usado mais adiante é o seguinte.

Exercício 71. Mostre que se L pode ser decidida por uma máquina de Turing determinística de tempo $T(n)$ então L pode ser decidida por uma família de circuitos $(C_n)_{n\in\mathbb{N}}$ de complexidade $T(n)^2$.

A recíproca não vale pois, como já vimos, há linguagens indecidíveis por máquina de Turing que tem circuito de tamanho polinomial (exercício 67), ou seja, entre modelos uniformes e não uniformes não há equivalência polinomial na complexidade de tempo (veja o exercício complementar 2, página 77).

A classe $\mathbf{P/poly}$ é a classe de todas as funções que são computáveis por uma família de circuitos de tamanho polinomial. O exercício acima prova que $P \subseteq P/poly$. Não é sabido se $NP \subseteq P/poly$ e se esse *não* for o caso então $P \neq NP$, resolvendo o problema mais importante da computação que até hoje não se sabe resolver

$$P{\neq}NP?$$

Observação 72. Segue do exercício acima que uma cota inferior para o tamanho dos circuitos que computam um determinado problema implica numa cota inferior para o tempo de uma máquina que computa o mesmo problema. Esse fato fornece uma estratégia de prova de que $P \neq NP$, por exemplo, se soubéssemos provar que SAT (ou qualquer outro problema em NP) não tem circuito de tamanho polinomial. Entretanto, provar cota inferior para tamanho de circuito mostra-se uma tarefa bastante árdua, atualmente não é conhecida nenhuma linguagem em NP com limitante inferior melhor que $5n$, onde n é o tamanho da entrada, ou seja, é um problema notadamente difícil descrever funções que tem cotas inferiores não triviais, principalmente, tendo em vista que funções com cotas não-triviais são abundantes, como enuncia o exercício 74 a seguir.

Exercício 73. Mostre que qualquer função booleana pode ser computada por um circuito de tamanho $O(n2^n)$.

Exercício 74. Prove que existe uma função booleana $\{0,1\}^n \to \{0,1\}$ não computada por circuito de tamanho menor que $\frac{2^n}{n}$, para todo natural $n \geqslant 2$. Mais que isso, prove que a maioria dessas funções não admite circuito com tamanho $< 2^{n/3}$.

A classe $\mathbf{BPP}$ — *Bounded-error, Probabilistic, Polynomial time* — é a classe de linguagens que são decididas por máquina de Turing probabilística de tempo

polinomial com probabilidade de erro $1/3$, ou seja, $L \subset \Sigma^*$ está em BPP se existe uma máquina de Turing probabilística M tal que

$$\mathbb{P}[\, M(w) = L(w) \,] \geqslant 2/3, \text{ para todo } w \in \Sigma^*.$$

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 se pode concluir do próximo resultado.

Lema 75. *Sejam $0 < \varepsilon < 1/2$ uma constante, $p(n)$ um polinômio e $L \subset \Sigma^*$ uma linguagem. Então para toda máquina de Turing M probabilística, de tempo polinomial e que decide L com probabilidade de erro ε existe uma máquina probabilística N de tempo polinomial e que decide L com probabilidade de erro $2^{-p(n)}$ nas entradas de tamanho n.*

Demonstração. Sejam ε, p, L e M como no enunciado. Tomemos $\delta = 4\varepsilon(1-\varepsilon)$ e notemos que $\delta < 1$ pois $\varepsilon < 1/2$. Definimos a máquina N que com entrada $w \in \Sigma^*$ executa da seguinte maneira:

1. computa $k = \left\lceil p(|w|)/\lg(\frac{1}{\delta})) \right\rceil$;
2. simula $2k+1$ vezes a máquina M com entrada w ;
3. se a maioria dos resultados das simulações terminou em *aceita*, então *aceita*, senão *rejeita*.

Claramente, N é uma máquina de tempo polinomial. Seja $S \in \{0,1\}^{2k+1}$ uma seqüência de respostas $M(w)$ do passo 2. 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 a rodada decidiu corretamente a pertinência de w em L.

Uma seqüência fixa S causa erro em N se $c < e$ e queremos limitar a probabilidade desse evento. Fixe S e suponha $c < e$. Como M tem probabilidade de erro limitada por ε, a probabilidade de uma seqüência S que faz N responder errado é no máximo $\varepsilon^e(1-\varepsilon)^c \leqslant \varepsilon^{k+1}(1-\varepsilon)^k$.

Assim, a probabilidade de erro é

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

onde a soma é sobre toda seqüência S que faz N responder errado. Pela escolha de k temos $\delta^k \leqslant 2^{p(|w|)}$. □

Exercício 76. Sejam M uma máquina de Turing probabilística e L uma linguagem tais que para constantes $0 < \varepsilon_1 < \varepsilon_2 < 1$ e todo $w \in \Sigma^*$

1. se $w \in L$, então $\mathbb{P}[\, M(w) = 1 \,] \geqslant \varepsilon_2$, e
2. se $w \notin L$, então $\mathbb{P}[\, M(w) = 1 \,] < \varepsilon_1$;

prove que L está em BPP.

Exemplo 77. Seja $p \in \mathbb{Q}[x_1, \ldots, x_n]$ um polinômio de grau total no máximo d e denote por $\langle p \rangle$ uma codificação de p em Σ. Seja $L = \{\langle p \rangle : p \equiv 0\}$. O algoritmo 6 corresponde a uma máquina de Turing M probabilística de tempo polinomial tal que se $\langle p \rangle \in L$ então $\mathbb{P}[\, M \text{ aceita } \langle p \rangle \,] = 1 \geqslant 2/3$ e, se $\langle p \rangle \notin L$ então $\mathbb{P}[\, M \text{ aceita } \langle p \rangle \,] \leqslant 1/3$, portanto, identidade polinomial está em BPP.

Exemplo 78. Seja $\mathbf{A}$ uma matriz quadrada de ordem n sobre $\mathbb{Q}$ e denote por $\langle A \rangle$ uma codificação de A em Σ. Seja $L = \{\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle : \mathbf{AB} = \mathbf{C}\}$. O algoritmo 7 corresponde a uma máquina de Turing probabilística de tempo polinomial M tal que se $\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle \in L$ então $\mathbb{P}[\, M \text{ aceita } \langle p \rangle \,] = 1 \geqslant 2/3$ e, se $\langle p \rangle \notin L$ então $\mathbb{P}[\, M \text{ aceita } \langle p \rangle \,] \leqslant 1/3$, portanto, L está em BPP.

Observação 79. No exemplo 78 a identidade das matrizes pode ser testada por um algoritmo determinístico de tempo polinomial, o produto de duas matrizes $n \times n$ pode ser feito com complexidade $O(n^{2{,}376})$, entretanto no caso do exemplo 77 não é conhecido nenhum algoritmo de tempo polinomial que testa identidade polinomial, mais que isso, nesse caso não se conhece algoritmo de tempo subexponencial.

Observação 80 (Definição alternativa de BPP). Usando o modelo *off-line* definido na seção 2.1.3 podemos definir BPP como a classe das linguagens L para as quais existe um polinômio p é uma máquina de Turing determinística de tempo polinomial, digamos $p(n)$, tais que para toda entrada $w \in \Sigma^*$ temos

$$\mathbb{P}_{b \in_R \{0,1\}^{p(|w|)}}[\, M(w, b) = L(w) \,] \geqslant 2/3.$$

Teorema 81 (Adleman). BPP $\subset$ P/poly.

Demonstração. Seja $L \in$ BPP e $p(n)$ um polinômio. Tome uma máquina probabilística M de tempo $p(n)$, *off-line* e que decide L com probabilidade de erro menor que $2^{-(n+2)}$.

Fixada uma entrada $w \in \{0,1\}^n$, o número de seqüências de bits aleatórios em $\{0,1\}^{p(n)}$ que fazem M errar é no máximo $2(2^{p(n)}/2^{n+2})$. Assim, somando-se sobre todo w, o número de seqüências de bits aleatórios que fazem M errar é $2^{p(n)-1}$,

portanto, há $2^{p(n)} - 2^{p(n)-1} = 2^{p(n)-1}$ seqüências para as quais M não erra qualquer que seja a entrada, escolhemos uma dessas seqüências, digamos $b \in \{0,1\}^{p(n)}$.

A máquina determinística $M(\,\cdot\,, b)$ não erra em nenhuma entrada de tamanho n e a partir dela construímos um circuito booleano C_n de tamanho $p(n)^2$ (veja exercício 71, página 57) tal que $C_n(w) = M(w, b)$, para todo w de tamanho n, logo $(C_n)_{n\geqslant 1}$ decide L, portanto $L \in P/poly$. □

A classe **RP** — *Randomized Polynomial time* — é formada pelas linguagens $L \subset \Sigma^*$ decididas por uma máquina de Turing probabilística M de tempo polinomial tal que para todo $w \in \Sigma^*$

1. se $w \in L$, então $\mathbb{P}[\ M \text{ aceita } w\ ] \geqslant \frac{1}{2}$, e
2. se $w \notin L$, então $\mathbb{P}[\ M \text{ aceita } w\ ] = 0$;

portanto, uma resposta *aceita* está sempre certa enquanto que uma resposta *rejeita* pode ser um falso negativo.

Exemplo 82. Sejam G um grafo, $k \in \mathbb{N}$ e $\langle G, k\rangle$ a codificação deles em Σ. Seja L a linguagem formada pelos pares $\langle G, k\rangle$ tais que $\text{mincut}(G) \leqslant k$. Do algoritmo 4 temos que $L \in RP$ (veja o teorema 17).

A classe **co-RP** é a classe das linguagens L para as quais existe uma máquina de Turing probabilística M de tempo polinomial, tal que para todo $w \in \Sigma^*$

1. se $w \in L$, então $\mathbb{P}[\ M \text{ aceita } w\ ] = 1$;
2. se $w \notin L$, então $\mathbb{P}[\ M \text{ aceita } w\ ] < 1/2$;

portanto, uma resposta *rejeita* está sempre certa enquanto que uma resposta *aceita* pode ser um falso positivo.

Exemplo 83. Nos exemplos 77 e 78 uma resposta *rejeita* está sempre certa enquanto que uma resposta *aceita* pode estar errada, logo as linguagens definidas nesses exemplos são linguagens em co-RP.

Como no caso BPP, a constante $1/2$ em RP e em co-RP é arbitrária, poderia ser qualquer constante $\varepsilon \in (0,1)$.

Exercício 84. Mostre que se trocarmos $1/2$ por qualquer constante $\varepsilon \in (0,1)$ nas definições de RP e de co-RP, obteremos as mesmas classes de linguagens.

Um fato interessante ocorre em RP ∩ co-RP; linguagens que estão em RP ∩ co-RP podem se beneficiar do resultado exato da máquina de Turing RP e da máquina de Turing co-RP. Uma máquina de Turing tem tempo esperado $T(n)$ se o tempo da máquina com qualquer entrada de tamanho n, é uma variável aleatória com valor esperado $O(T(n))$.

A classe **ZPP** — *Zero-error Probabilistic Polynomial time* — é a classe de complexidade de todas as linguagens para as quais existe uma máquina de Turing probabilística M de tempo *esperado* polinomial, tal que $\mathbb{P}[\, M(w) = L(w) \,] = 1$, para todo $w \in \Sigma^*$, ou seja

1. se $w \in L$, então $\mathbb{P}[\, M \text{ aceita } w \,] = 1$, e
2. se $w \notin L$, então $\mathbb{P}[\, M \text{ aceita } w \,] = 0$.

Lema 85. $\mathrm{ZPP} = \mathrm{RP} \cap \text{co-RP}$.

Demonstração. Seja $L \in \mathrm{ZPP}$ uma linguagem e M uma máquina de tempo esperado $p(n)$ que reconhece L. Definimos uma máquina N que computa da seguinte maneira: dada uma entrada $w \in \Sigma^*$

1. simula M com entrada w até no máximo $2p(|w|)$ transições;
2. se M aceitou w então devolve *aceita*, senão devolve *rejeita*.

Nesse caso, se $w \notin L$ então N termina sem aceitar w, portanto responde *aceita* com probabilidade 0. Agora, se $w \in L$ então a máquina N aceita w ou termina pelo limite das $2p(|w|)$ transições, no segundo caso N devolve a resposta errada, *rejeita*. A probabilidade da resposta errada é $\mathbb{P}[\, t_M(w) > 2p(|w|) \,]$, onde $t_M(w)$ é a variável aleatória para o tempo da máquina M com entrada w. Porém,

$$\begin{aligned} \mathbb{E}\, t_M(w) = p(|w|) = \sum_{i \geqslant 1} i\mathbb{P}[\, t_M(w) = i \,] \geqslant \sum_{i > 2p(|w|)} i\mathbb{P}[\, t_M(w) = i \,] \geqslant \\ > \sum_{i > 2p(|w|)} 2p(|w|)\mathbb{P}[\, t_M(w) = i \,] = 2p(|w|)\mathbb{P}[\, t_M(w) > 2p(|w|) \,], \end{aligned} \tag{2.1}$$

logo $\mathbb{P}[\, t_M(w) > 2p(|w|) \,] < 1/2$, portanto, se $w \in L$ então $\mathbb{P}[\, M \text{ aceita } w \,] \geqslant 1/2$ e com isso temos que $L \in \mathrm{RP}$.

Do mesmo modo, podemos provar que $\mathrm{ZPP} \subseteq \text{co-RP}$, definimos uma máquina N' que com entrada w

1. simula M com entrada w até no máximo $2p(|w|)$ transições;

2. se M rejeita w então devolve *rejeita*, senão devolve *aceita*.

Se $w \in L$ então $\mathbb{P}[\ M \text{ aceita } w\ ] = 1$. Se $w \notin L$ então a máquina pode aceitar erroneamente e $\mathbb{P}[\ M \text{ aceita } w\ ] = \mathbb{P}[\ t_M(w) > 2p(|w|)\ ] < 1/2$. Portanto, ZPP $\subseteq$ RP $\cap$ co-RP.

Agora, fixemos uma linguagem $L \in \text{RP} \cap \text{co-RP}$. Sejam M_{RP} a máquina que prova que L está em RP e $M_{\text{co-RP}}$ a máquina que prova que L está em co-RP. A partir dessas duas máquinas definimos a máquina M' que computa da seguinte forma: dada uma entrada $w \in \Sigma^*$

1. Repete até obter uma resposta:

   (a) simula M_{RP} com entrada w e se M_{RP} aceita w, então devolve *aceita*;

   (b) simula $M_{\text{co-RP}}$ com entrada w e se $M_{\text{co-RP}}$ rejeita w, então devolve *rejeita*.

Se M_{RP} responder *aceita*, então pela definição garantimos que $w \in L$. Analogamente, se $M_{\text{co-RP}}$ responder *rejeita*, então pela definição garantimos que $w \notin L$. Assim nunca obtemos uma resposta errada. O número esperado de execuções até o passo (a) ter sucesso é no máximo 2 e o mesmo vale para (b), portanto o tempo esperado de M' é polinomial em $|w|$. □

Exemplo 86. O problema de ordenação pode ser escrito como um problema de decidir a linguagem L dos pares $(s, \pi) \in \mathbb{N}^n \times \mathbb{S}^n$ tais que π é uma permutação que ordena a seqüência s formada de elementos distintos. A linguagem L está em ZPP e o algoritmo que prova essa afirmação é o *Quicksort* aleatorizado.

Exercício 87. A classe **PP** — *Probabilistic Polynomial time* — é a classe das linguagens para as quais existe uma máquina de Turing probabilística M de tempo polinomial tal que $\mathbb{P}[\ M(w) = L(w)\ ] \geqslant 1/2$ para todo $w \in \Sigma^*$.

Se C é uma classe de complexidade e L uma linguagem então $L \in \text{co-C}$ se, e somente se, o complemento de L está em C.

Prove as seguintes inclusões

1. $P \subseteq ZPP \subseteq RP \subseteq NP \subseteq PP$.

2. $P \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP$.

3. $PP = \text{co-PP}$.

4. $BPP = \text{co-BPP}$.

5. ZPP = co-ZPP.

Exercício 88. Prove que se NP $\subseteq$ BPP então NP = RP. Atualmente, não é sabido se NP $\subseteq$ BPP.

Exercício 89. Prove que se RP $\subseteq$ co-PP então ZPP = RP e RP $\subseteq$ NP $\cap$ co-NP. Com relação as classes de complexidade envolvidas nesse exercício, atualmente, não se sabe RP = co-RP, se RP $\subseteq$ NP $\cap$ co-NP e se NP = co-NP.

Figura 2.2: Panorama atual das classes de complexidade.

Os próximos exercícios revelam que em classes polinomiais, a classe BPP por exemplo, a classe permanece a mesma quando dispomos de bits aleatórios de uma fonte desbalanceada (não-uniforme).

Exercício 90. Mostre que se o i-ésimo bit de $p \in (0,1)$ pode ser determinado em tempo polinomial em i então a distribuição sobre bits com $\mathbb{P}(1) = p$ pode ser simulada por uma máquina de Turing probabilística de tempo esperado $O(1)$.

Exercício 91. Mostre que uma moeda com $\mathbb{P}[\text{ cara }] = 1/2$ pode ser simulada por uma máquina de Turing probabilística que lança moeda com $\mathbb{P}[\text{ cara }] = p$ com tempo esperado $O(1/(p-p^2))$.

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

2.2.1 BPP está na Hierarquia Polinomial

Nesta seção descrevemos um resultado importante mas cuja apreciação depende de um conhecimento avançado em Complexidade Computacional, os detalhes po-

dem ser lidos na seção 17.2 de Papadimitriou (1994). A omissão da leitura desta seção não prejudica a compreensão dos tópicos seguintes.

Se A é uma classe de complexidade (de tempo) e L uma linguagem então denotamos por A^L a classe das linguagens que são decididas por uma máquina, a qual satisfaz a restrição que define A, com oráculo L; uma máquina M com oráculo L pode consultar o oráculo para decidir pertinência em B sem custo adicional, isto é, a consulta ao oráculo custa $O(1)$. Se B é uma classe de complexidade (de tempo) e L uma linguagem *completa* em B então denotamos por A^B a classe A^L (veja mais de computação com oráculos em 14.3 de Papadimitriou (1994) ou 1.2.3.5 de Goldreich (2008)).

A **hierarquia polinomial** é a seqüência de classes:

- $\Delta_0 = \Sigma_0 = \Pi_0 = P$,
- e para $i \geqslant 0$

$$\Delta_{i+1} = P^{\Sigma_i},$$
$$\Sigma_{i+1} = NP^{\Sigma_i},$$
$$\Pi_{i+1} = \text{co-NP}^{\Sigma_i},$$

e $PH = \bigcup_{i \geqslant 0} \Sigma_i$. Dessa forma, $\Delta_1 = P^P = P$, $\Sigma_1 = NP^P = NP$ e $\Pi_1 = \text{co-NP}^P = \text{co-NP}$.

Antes de provarmos que BPP está na hierarquia polinomial, vamos relatar um resultado conhecido cuja prova pode ser vista em Papadimitriou (1994): se $\Sigma_i = \Pi_i$ para algum $i \geqslant 1$ então $\Sigma_j = \Pi_j = \Delta_j = \Sigma_i$ para todo $j > i$, ou seja $PH = \Sigma_i$.

Como conseqüência do resultado enunciado no parágrafo anterior, se $P = NP$ ou $NP = \text{co-NP}$, então toda hierarquia colapsa para o primeiro nível da hierarquia ($i = 1$), ou seja $PH = P$.

A partir de agora, vamos nos concentrar no segundo nível da hierarquia, $\Sigma_2 = NP^{NP}$ e $\Pi_2 = \text{co-NP}^{NP}$. A classe Σ_2 admite a seguinte caracterização: é a classe das linguagens L para as quais existe uma máquina de Turing M de tempo polinomial e um polinômio p tal que para todo $w \in \Sigma^*$

$$w \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|w|)}\ \forall v \in \{0,1\}^{p(|w|)},\ M(w,u,v) \text{ aceita } w.$$

Ainda, temos que $\Pi_2 = \{\overline{L} \colon L \in \Sigma_2\}$, ou seja, é a classe das linguagens L para as quais existe uma máquina de Turing M de tempo polinomial e um polinômio p tal que para todo $w \in \Sigma^*$

$$w \in L \Leftrightarrow \forall u \in \{0,1\}^{p(|w|)}\ \exists v \in \{0,1\}^{p(|w|)},\ M(w,u,v) \text{ aceita } w.$$

Exemplo 92. A linguagem $\{\langle G, k\rangle : G$ é um grafo com conjunto independente máximo de tamar está em Σ_2. De fato, um par $\langle G, k\rangle$ está na linguagem se e somente se existe um subconjunto de vértices I independente de tamanho k e todo subconjunto N de tamanho pelo menos $k+1$ não é independente.

Exercício 93. Formalize o exemplo acima. Prove que a linguagem do exemplo está em Π_2.

O principal resultado dessa seção é o seguinte.

Teorema 94 (Sipser-Gács)**.** $\mathrm{BPP} \subset \Sigma_2 \cap \Pi_2$.

Demonstração. Seja $L \in \mathrm{BPP}$ e $q(n)$ um polinômio. Tome uma máquina probabilística M de tempo $q(n)$, *off-line* e que decide L com probabilidade de erro menor que 2^{-n}. Seja $w \in \Sigma^*$ e defina

$$A_w = \{b \in \{0,1\}^{q(n)} : M(w, b) \text{ aceita } w\}.$$

Da definição de BPP temos

$$\begin{aligned} w \in L &\Rightarrow |A_w| \geqslant (1-2^{-n})2^{q(n)} \\ w \notin L &\Rightarrow |A_w| \leqslant 2^{-n}2^{q(n)}. \end{aligned}$$

Defina $k = q(n)/n+1$, considere um subconjunto qualquer $U = \{u_1, u_2, \ldots, u_k\} \subseteq \{0,1\}^{q(n)}$ e defina o grafo $G = G(U)$ por

$$G = \left(\{0,1\}^{q(n)}, \{\{r,s\} : r = s \operatorname{xor} u_i \text{ para algum } i\}\right)$$

note que $r = s \operatorname{xor} u_i$ é equivalente a $s = r \operatorname{xor} u_i$, portanto, as arestas estão bem definidas. Denote por $N(S)$ a vizinhança de $S \subseteq \{0,1\}^{q(n)}$ em G, ou seja, o conjunto dos vértices r tais que $\{r,s\}$ é aresta para algum $s \in S$. Em particular $|N(\{s\})| = k$.

Seja $S \subset \{0,1\}^{q(n)}$ tal que $|S| \leqslant 2^{q(n)-n}$. Então, para qualquer que seja o conjunto U como acima vale, em $G(U)$, que $|N(S)| \leqslant k|S| < 2^{q(n)}$, portanto, $N(S) \neq \{0,1\}^{q(n)}$.

Agora, se $S \subseteq \{0,1\}^{q(n)}$ é tal que $|S| \geqslant (1-2^{-n})2^{q(n)}$ então existe U como acima para o qual $N(S) = \{0,1\}^{q(n)}$ em $G(U)$. Para provar esse fato, tome $U \in_R \{V \subset \{0,1\}^{q(n)} : |V| = k\}$ e para $r \in \{0,1\}^{q(n)}$ considere o evento "$r \notin N(S)$" em $G(U)$. Vamos provar que $\sum_r \mathbb{P}_U[\, r \notin N(S)\,] < 1$. O evento "$r \notin N(S)$" ocorre se, e somente se, para todo $i \in \{1,2,\ldots,k\}$ temos $r \operatorname{xor} u_i \notin S$, mas $\mathbb{P}_U[\, r \operatorname{xor} u_i \notin S\,] < (2^{-n})^k < 2^{-q(n)}$, portanto, $\mathbb{P}_U\left[\, N(S) \neq \{0,1\}^{q(n)}\,\right] \leqslant \sum_r \mathbb{P}_U[\, r \notin S\,] < 1$. Assim, podemos concluir que existe U para o qual $N(S) = \{0,1\}^{q(n)}$.

Logo, a partir dos dois parágrafos anteriores podemos concluir que

$$w \in L \Leftrightarrow \exists u_1, u_2, \ldots, u_k \in \{0,1\}^{q(n)} \forall r \in \{0,1\}^{q(n)},\ M(w, r \operatorname{xor} u_i) \text{ aceita } w, \text{ para algum } i$$

e com isso temos $L \in \Sigma_2$. De BPP = co-BPP temos $L \in \Pi_2$ concluindo a prova do teorema. □

Como comentamos acima, se P = NP então P = PH e como BPP ⊂ PH, temos BPP ⊆ P. Já sabiamos que P ⊆ BPP, portanto, se P = NP então P = BPP. Com isso temos a seguinte conseqüência do teorema.

Corolário 95. *Se* P = NP *então* BPP = P.

Assim, se não há problemas difíceis no sentido de que todo problema NP tem algoritmo eficiente, como satisfatibilidade de fórmula booleana por exemplo, então as soluções probabilísticas eficientes podem ser desaleatorizadas.

A desaleatorização de algoritmos probabilístico é um assunto bastante interessante que será tratado na próxima seção. Particularmente interessante, é o fato de não sabermos se algoritmos probabilísticos eficientes (BPP) podem ser totalmente desaleatorizado mantendo-se a eficiência (P).

2.3 Desaleatorização de BPP

Uma pergunta natural que podemos fazer é se a aleatorização de algoritmos acrescenta algo à computação (veja a observação 79 acima); tecnicamente, a pergunta que mais interessa é

$$\text{P} = \text{BPP?} \tag{2.2}$$

e, atualmente, não é sabido se BPP ⊆ P.

A seguir, nessa seção, enunciamos alguns resultados conhecidos que indicam que a pergunta está fortemente relacionada a pergunta P = NP? e que resumem a situação atual desse problema importante da computação. A maioria desses resultados não são demonstrados pois as provas demandariam a apresentação de uma grande quantidade de conteúdo bastante técnico, o qual é mais apropriado para um texto de Complexidade Computacional. Muitas das demonstrações podem ser encontradas em Goldreich (2008).

Por um lado vimos que P = NP então P = BPP. Por outro lado, vários resultados mostram que a existência de problemas computacionalmente difíceis implica em desaleatorização eficiente. Por exemplo,

- Seja **E** a classe das linguagens decididas por máquinas determinísticas de tempo $2^{O(n)}$. Impagliazzo e Wigderson Impagliazzo and Wigderson (1999) provaram em 1997 que se existe uma linguagem em $L \in E$ (SAT, por exemplo) que não admite circuito booleano de tamanho $2^{o(n)}$ (i.e., tamanho subexponencial), então $BPP = P$.

- **NEXP** é a classe das linguagens decididas por máquinas de Turing não-determinísticas de tempo $2^{O(n^k)}$. Não é sabido se $NEXP \subseteq P/poly$. Kabanets e Impagliazzo provaram em 2004 que se o teste de identidade polinomial (algoritmo 6) pode ser desaleatorizado então ou $NEXP \not\subset P/poly$ ou Permanente não admite circuito aritmético[3] de tamanho polinomial (um resultado mais forte que $P \neq NP$),

ou seja, provar que $P = BPP$ requer provar cota inferior superpolinomial para circuito booleano ou circuito aritmético (veja observação 72). Esses resultados parecem indicar que $P = BPP$, porém sem as hipóteses de existência de problemas computacionalmente difíceis, atualmente, nem $BPP \neq NE$ está provado, onde **NE** é a classe das linguagens decididas em tempo $2^{O(n)}$ por máquinas não-determinísticas.

Também são conhecidos alguns resultados de desaleatorização em tempo subexponencial sob hipótese de haver problemas difíceis. **EXP** é a classe das linguagens decididas por máquinas de Turing determinísticas de tempo $2^{O(n^k)}$, para algum $k \in \mathbb{N}$. Babai, Fortnow, Nisan, e Wigderson Babai, Fortnow, Nisan, and Wigderson (1993) provaram que se $EXP \not\subset P/poly$ (o que não é sabido), então $BPP \subseteq SUBEXP$, onde **SUBEXP** é a classe das linguagens decididas por máquinas de Turing determinísticas de tempo $2^{O(n^\varepsilon)}$, para todo $\varepsilon > 0$.

Observamos que todo algoritmo aleatorizado admite uma desaleatorização trivial. Seja $L \in BPP$ e M uma máquina probabilística que reconhece L em tempo $p(n)$. Uma máquina determinística simula M nas $2^{p(n)}$ seqüências possíveis de bits aleatórios e devolve o resultado de maioria. Ainda, se o número de bits aleatórios usados de fato for $s(n)$ o tempo resultante é $O(2^{s(n)}p(n))$, logo teríamos um algoritmo determinístico de tempo polinomial caso $s(n) = O(\log n)$.

Do parágrafo acima podemos formalizar uma prova para o seguinte resultado, já que se a máquina M é de tempo polinomial então ela usa no máximo um número polinomial de bits aleatórios, ou seja, $s(n) = O(n^k)$.

[3] A definição é análoga ao de circuito booleano a não ser que as entradas são de $\mathbb{Z}^n$ e as operações são $+, -, \times$ ao invés de e, ou, não, portanto tal circuito computa uma função $\mathbb{Z}^n \to \mathbb{Z}$.

Proposição 96. $\mathrm{BPP} \subseteq \mathrm{EXP}$.

Não é sabido se todo problema intratável admite algoritmo aleatorizado eficiente, ou seja, não se sabe se $\mathrm{BPP} = \mathrm{EXP}$.

2.3.1 Desaleatorização usando geradores pseudoaleatórios

Uma técnica importante em desaleatorização é baseada em geradores pseudoaleatórios. Um gerador pseudoaleatório é, grosso modo, um algoritmo determinístico que recebe um seqüência binária r de tamanho k e devolve uma seqüência t de tamanho $s(k)$ de modo que um algoritmo aleatorizado eficiente não consegue distinguir t de uma seqüência $u \in_R \{0,1\}^{s(k)}$ na maioria da vezes, digamos em $8/9$ dos casos, ou seja, a saída do gerador tem distribuição com discrepância de no máximo $1/9$ com relação a distribuição uniforme.

Para a desaleatorização de um algoritmo M que usa $s(k(n))$ bits aleatórios com uma entrada de tamanho n, basta executar o gerador para todas as $2^{k(n)}$ possíveis seqüências de entrada, e executar o algoritmo aleatorizado M com cada saída produzida, e devolver o resultado de maioria. Se M erra com probabilidade $1/3$ com $u \in_R \{0,1\}^s$ e a probabilidade de erro varia $1/9$ com $t \in \{0,1\}^s$ então a probabilidade de erro de M com bits pseudoaleatórios é $4/9$, portanto a resposta da maioria está correta.

Uma idéia para construir esses geradores pseudoaleatórios é contar com a dificuldade de se computar determinadas funções, no mesmo estilo das bases da criptografia moderna que usa, por exemplo, a dificuldade de se computar a fatoração de um inteiro a seu favor. Se um gerador pseudoaleatório tem "embutido" uma função NP-difícil então um algoritmo de tempo polinomial que consegue distinguir uma seqüência pseudoaleatória de uma aleatória pode ser transformado numa solução eficiente para tal problema difícil (o que conjectura-se não existir; e se existir então $\mathrm{BPP} = \mathrm{P}$!, ou seja, saímos ganhando nos dois casos); essa estratégia foi usada no geradores de Blum–Micali–Yao Blum and Micali (1984), Yao (1982) e Nissan–Wigderson Nisan and Wigderson (1994), por exemplo.

No que segue, $k \in \mathbb{N}$ e $s\colon \mathbb{N} \to \mathbb{N}$. Uma função $G_k\colon \{0,1\}^k \to \{0,1\}^{s(k)}$ computável em tempo $2^{O(k)}$ é um **gerador pseudoaleatório** se para todo circuito C_k de tamanho no máximo $s(k)^3$

$$\left| \mathbb{P}_{x \in_R \{0,1\}^k} [\, C_k(G_k(x)) = 1 \,] - \mathbb{P}_{y \in_R \{0,1\}^{s(k)}} [\, C_k(y) = 1 \,] \right| \leqslant \frac{1}{9}. \tag{2.3}$$

Vamos assumir que s é não-decrescente e que $k\colon \mathbb{N} \to \mathbb{N}$ é computável em tempo

polinomial. No que segue também assumimos as máquinas de Turing probabilísticas no modelo *off-line*, lembramos que se M é uma máquina então $M(w, r)$ denota o resultado da computação de M com entrada w e bits aleatórios r, e se L é um conjunto então denotamos também por L a função característica de L.

Observamos que toda linguagem $L \in \mathrm{BPP}$ que é decidida por uma máquina de Turing probabilística M de tempo $O(s(k(n)))$ também é decidida por uma máquina de Turing determinística de tempo $2^{O(k(n))}$. De fato, de $L \in \mathrm{BPP}$ existe uma máquina probabilística M de tempo polinomial tal que

$$\mathbb{P}_{r \in_R \{0,1\}^m} [\, M(w, r) = L(w) \,] \geqslant \frac{2}{3} \tag{2.4}$$

onde $m \leqslant s(k(n))$. Considere a máquina de Turing determinística D que com entrada w gera todo $z \in \{0,1\}^{k(n)}$, computa $M(w, G_k(z))$ para todo z e devolve o resultado de maioria (caso $m < s$ então M, simulado por D, usa os m primeiros bits da seqüência devolvida por G_k). Esse algoritmo tem tempo $2^{O(k(n))}$, resta mostrar que responde corretamente; para isso mostraremos que mais da metade das respostas estão certas quando percorre-se $z \in \{0,1\}^{k(n)}$. Mais que isso, mostraremos que vale uma desigualdade um pouco mais fraca que (2.4).

Suponha que

$$\mathbb{P}_{z \in_R \{0,1\}^k} [\, M(w, G(z)) = L(w) \,] < 2/3 - 1/9$$

para uma seqüência infinita de w's. Então para cada palavra dessa seqüência

$$\left| \mathbb{P}_{z \in_R \{0,1\}^k} [\, M(w, G(z)) = L(w) \,] - \mathbb{P}_{r \in_R \{0,1\}^{s(k)}} [\, M(w, r) = L(w) \,] \right| > \frac{1}{9}$$

e usando o exercício 71 podemos construir um circuito booleano C_k tal que $C_k(r) = M(w, r)$ com tamanho $O(s(k(n))^2)$, contrariando (2.3).

Notemos que se existe um gerador pseudoaleatório com $s = 2^{\varepsilon k}$, para algum $\varepsilon > 0$, então D tem tempo $2^{O((1/\varepsilon) \lg s)} = s^{O(1)}$.

Teorema 97. *Se existe um gerador pseudoaleatório com* $s(k) = 2^{\varepsilon k}$*, para algum* $\varepsilon > 0$*, então* $\mathrm{BPP} = \mathrm{P}$. □

Analogamente

Teorema 98. *Se existe um gerador pseudoaleatório com* $s = k^c$*, para todo* $c > 1$*, então* $\mathrm{BPP} \subseteq \mathrm{SUBEXP}$. □

2.3.2 Construções de seqüências k-a-k independentes

Outra técnica de desaleatorização é baseada em construções de estruturas discretas pseudoaleatórias, a seguir veremos como esses objetos são úteis para desaleatorização parcial ou total de algoritmos.

Seqüências com distribuição uniforme e independência 2-a-2.

Vamos mostrar como construir uma seqüência $t \in \{0,1\}^{2^k-1}$ de bits 2-a-2 independentes a partir de $r \in_R \{0,1\}^k$.

Para $b \in \{0,1\}^k \setminus \{0\}^k$ defina a variável aleatória $Y_b\colon \{0,1\}^k \to \{0,1\}$ por

$$Y_b(r) = \sum_{j=1}^{k} r_j b_j, \text{ onde a soma é feita módulo 2}$$

e vamos mostrar, usando o princípio da decisão adiada, que se $r \in_R \{0,1\}^k$ então $Y_b(r) \in_R \{0,1\}$ e essas variáveis são 2-a-2 independentes.

Primeiro, notemos que $\mathbb{P}_{r\in_R\{0,1\}^k}[\, Y_b = 1\,] = 1/2$. De fato, dado $b \in \{0,1\}^k \setminus \{0\}^k$ fixamos $\ell \in \{1,\ldots,k\}$ tal que $b_\ell = 1$. Em seguida, particionamos o espaço amostral definindo para cada $(k-1)$-upla $(r'_1,\ldots,r'_{n-1}) \in \{0,1\}^{k-1}$, o evento

$$A_{r'_1,\ldots,r'_{k-1}} = \left\{(r'_1,\ldots,r'_{\ell-1},s,r'_\ell,\ldots,r'_{k-1})\colon s \in \{0,1\}\right\}$$

e notemos que $Y_b(r'_1,\ldots,r'_{\ell-1},0,r'_\ell,\ldots,r'_{k-1}) = 1 - Y_b(r'_1,\ldots,r'_{\ell-1},1,r'_\ell,\ldots,r'_{k-1})$ e assim

$$\mathbb{P}_{r\in_R\{0,1\}^k}[\, Y_b = 1\,] = \sum_{(r'_1,\ldots,r'_{k-1})} \mathbb{P}_{r\in_R\{0,1\}^k}\left([Y_b = 1] \cap A_{r'_1,\ldots,r'_{k-1}}\right) = \sum_{(r'_1,\ldots,r'_{k-1})} \frac{1}{2^k} = \frac{1}{2}.$$

Agora, para duas seqüências distintas, b e b', temos $\mathbb{P}_{r\in_R\{0,1\}^k}[\, Y_b + Y_{b'} = 1\,] = 1/2$, onde a soma é módulo dois. Ainda, seja $\ell \in \{1,\ldots,k\}$ tal que $b_\ell + b'_\ell = 1$, que existe pois $b \neq b'$. Da definição

$$Y_b + Y_{b'}(r) = \sum_{j=1}^{k} r_j b_j + \sum_{j=1}^{k} r_j b'_j = \sum_{j=1}^{k} r_j (b_j + b'_j)$$

com todas as somas módulo 2. Escrevendo $b + b'$ para denotar a soma (módulo 2) coordenada a coordenada temos que $Y_b + Y_{b'}(r) = Y_{b+b'}(r)$ e, como acima, concluímos que $\mathbb{P}_{r\in_R\{0,1\}^k}[\, Y_{b+b'} = 1\,] = 1/2$. Para finalizar vamos provar que Y_b e

$Y_{b'}$ são independentes. Temos

$$\begin{cases} \mathbb{P}([Y_b = 0] \cap [Y_{b'} = 0]) + \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 1]) & = 1/2 \\ \mathbb{P}([Y_b = 0] \cap [Y_{b'} = 1]) + \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 0]) & = 1/2 \\ \mathbb{P}([Y_b = 0] \cap [Y_{b'} = 0]) + \mathbb{P}([Y_b = 0] \cap [Y_{b'} = 1]) & = 1/2 \\ \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 1]) + \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 0]) & = 1/2 \\ \mathbb{P}([Y_b = 0] \cap [Y_{b'} = 0]) + \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 0]) & = 1/2 \\ \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 1]) + \mathbb{P}([Y_b = 0] \cap [Y_{b'} = 1]) & = 1/2 \end{cases}$$

onde as duas primeiras equações seguem de $\mathbb{P}_{r \in_R \{0,1\}^k}[\, Y_b + Y_{b'} = 1 \,] = 1/2$ e o restante de Y_b e $Y_{b'}$ uniformes. A solução desse sistema linear é

$$\mathbb{P}([Y_b = 0] \cap [Y_{b'} = 0]) = \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 1]) = \mathbb{P}([Y_b = 0] \cap [Y_{b'} = 1]) = \mathbb{P}([Y_b = 1] \cap [Y_{b'} = 0]) =$$

e como $Y_b \in_R \{0, 1\}$ concluímos que as variáveis aleatórias são independentes.

A seguir damos um exemplo de aplicação desse método de desaleatorização.

Exemplo 99 (Corte máximo). O seguinte algoritmo recebe um grafo $G = (V, E)$ e devolve um corte com pelo menos $|E|/2$ arestas.

Algoritmo 11: Corte

Dado : um grafo $G = (V, E)$.
Devolve: um subconjunto $S \subset V$ tal que o corte $E(S, \bar{S})$ tem $|E|/2$ arestas, pelo menos.

```
1 repita
2     S ← ∅;
3     para v ∈ V faça
4         b ←_R {0, 1}
5         se b = 1 então S ← S ∪ {v}
6 até que até que E(S, S̄) tenha pelo menos |E| arestas ;
```

Analisando, primeiro as linhas 2—5, vemos que para cada aresta $\{u, v\}$, temos

$$\mathbb{P}\left[\, \{u, v\} \in E(S, \bar{S}) \,\right] = \mathbb{P}[\, u \in S \,]\, \mathbb{P}[\, v \notin S \,] + \mathbb{P}[\, v \in S \,]\, \mathbb{P}[\, u \notin S \,] = 1/2, \quad (2.5)$$

portanto, o tamanho esperado do corte é $|E|/2$, logo existe corte em G com pelo menos metade das arestas de G.

Cada rodada das linhas 2—5 desse algoritmo sorteia $|V|$ bits aleatórios mutuamente independentes. Notemos que se os sorteios são 2-a-2 independentes, ao invés

de mutuamente independentes, ou seja cada par de vértices tem sorteios independentes, então (2.5) vale, ou seja, $(Y_b)_b$ define um corte tal que $\mathbb{P}\left[\ \{u,v\} \in E(S,\bar{S})\ \right] = 1/2$ e portanto para algum $r \in \{0,1\}^k$ o corte tem pelo menos metade das arestas.

Com isso podemos usar o esquema acima e com $r = \lceil \log_2(n+1) \rceil$ bits genuinamente aleatórios obtermos os n bits 2-a-2 independentes necessários para cada rodada do **repita**. Mais que isso, o algoritmo pode ser completamente desaleatorizado da seguinte forma. Suponha que os vértices de G são $1, 2, \ldots, n$ e denote por $b(v)$ a representação binária de $v \in \{1, 2, \ldots, n\}$.

Algoritmo 12: Corte desaleatorizado

Dado : um grafo $G = (V, E)$.
Devolve: um subconjunto $S \subset V$ tal que o corte $E(S, \bar{S})$ tem $|E|/2$ arestas, pelo menos.

1 **para cada** $r \in \{0,1\}^{\lceil \log_2(n+1) \rceil}$ **faça**
2 $S \leftarrow \varnothing$;
3 **para** $v \in V$ **faça**
4 **se** $Y_{b(v)}(r) = 1$ **então** $S \leftarrow S \cup \{v\}$;
5 **se** $|E(S, \bar{S})| \geqslant n/2$ **então devolva** S.

Exercício 100. Determine os tempos de execução dos algoritmos acima.

Uma outra construção que assume q valores 2-a-2 independentes é feita da seguinte maneira. Seja q um primo ou potência de primo e $\mathbb{F}_q$ o corpo com q elementos. A seguir as operações aritméticas são as do corpo. Para $(a,b) \in_R \mathbb{F}_q^2$ definimos para todo $x \in \mathbb{F}_q$ a variável aleatória $Y_x \colon \mathbb{F}_q^2 \to \mathbb{F}_q$ por

$$Y_x(a,b) = a \cdot x + b. \tag{2.6}$$

Para provar que Y_{x_1} e Y_{x2} são independentes, basta notar que

$$\begin{pmatrix} Y_{x_1} \\ Y_{x_2} \end{pmatrix} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

é uma bijeção, assim, para $x_1 \neq x_2$ temos $\mathbb{P}\left[\ [Y_{x_1} = i] \cap [Y_{x_2} = j]\ \right] = \mathbb{P}\left[\ [a = i'] \cap [b = j']\ \right]$ onde

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix}^{-1} \begin{pmatrix} i \\ j \end{pmatrix}$$

como $\mathbb{P}((a,b) = (i',j')) = 1/q^2$ e $\mathbb{P}\left[\ Y_x = i\ \right] = 1/q$ (justifique) temos que Y_{x_1} e Y_{x_2} são independentes.

Exercício 101. Use essa construção para desaleatorização do algoritmo 11.

Seqüências com distribuição uniforme e independência k-a-k.

Seja $\mathbb{F}_q$ o corpo de ordem q, onde q é primo ou potência de primo e consideramos a distribuição uniforme em $\mathbb{F}_q$. Vamos generalizar a construção acima para conseguirmos, com poucos bits aleatórios, uma seqüência de q^k bits k-a-k independentes, para $k \in \mathbb{N}$.

Para todo $x \in \mathbb{F}_q$ definimos a variável aleatória $Y_x \colon \mathbb{F}_q^k \to \mathbb{F}_q$ por

$$Y_x(r) = r_0 + r_1 x + \cdots + r_{k-1} x^{k-1}.$$

Vamos provar que quaisquer k dessas variáveis são independentes. Dados $x_1, \ldots, x_k \in \mathbb{F}_q$ a matriz

$$\begin{pmatrix} 1 & x_1 & {x_1}^2 & \cdots & {x_1}^{k-1} \\ 1 & x_2 & {x_2}^2 & \cdots & {x_2}^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & {x_k}^2 & \cdots & {x_k}^{k-1} \end{pmatrix}$$

é uma matriz de Vandermonde, portanto invertível. Logo, para $i \in \mathbb{F}_q^k$

$$\mathbb{P}_{r \in_R \mathbb{F}_q^k} \left[\bigcap_{j=1}^{k} \left[Y_{x_j} = i_j \right] \right] = \mathbb{P}_{r \in_R \mathbb{F}_q^k} \left[\bigcap_{j=1}^{k} \left[r_j = i'_j \right] \right]$$

onde

$$\begin{pmatrix} i'_1 \\ i'_2 \\ \vdots \\ i'_k \end{pmatrix} = \begin{pmatrix} 1 & x_1 & {x_1}^2 & \cdots & {x_1}^{k-1} \\ 1 & x_2 & {x_2}^2 & \cdots & {x_2}^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & {x_k}^2 & \cdots & {x_k}^{k-1} \end{pmatrix}^{-1} \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_k \end{pmatrix}.$$

A independência k-a-k segue do fato de $\mathbb{P}_{r \in_R \mathbb{F}_q^k} \left[\bigcap_{j=1}^{k} \left[r_j = i'_j \right] \right] = \mathbb{P}(r = i') = 1/q^k$ e $\mathbb{P}_{r \in_R \mathbb{F}_q^k} [\, Y_x(r) = y \,] = 1/q$. De fato, suponha que $r_1, \ldots, r_{k-1}$ foram sorteados, então $Y_x = y$ se sorteamos $y - (r_1 x + \cdots + r_{k-1} x^{k-1}) \in \mathbb{F}_q$.

Observação 102 (**Corpos binários**). Um caso particularmente interessante são os corpos com 2^k elementos, $k \in \mathbb{N}$. Os elementos de $\mathbb{F}_{2^k}$ podem ser identificados com as seqüências binárias $\{0,1\}^k$ da seguinte forma. Para $k = 1$, temos $\mathbb{F}_2$ isomorfo ao $\mathbb{Z}_2$, i.e., o conjunto $\{0,1\}$ com as operações binárias usuais de soma e multiplicação. Denotamos por $\mathbb{F}_2[x]$ o anel dos polinômios com coeficientes em $\mathbb{F}_2$ com soma e multiplicação usuais. O quociente $\mathbb{F}_2[x]/(f)$, onde $f \in \mathbb{F}_2[x]$ é um polinômio irredutível em $\mathbb{F}_2[x]$ de grau k e (f) o ideal gerado por f, é um corpo com 2^k elementos, dados por todos os polinômios de $F_2[x]$ de grau no máximo k. Dessa forma, $\mathbb{F}_{2^k}$ é

isomorfo a $\mathbb{Z}_2[x]/(f)$ e uma seqüência $(b_0, b_1, \ldots, b_{k-1}) \in \{0,1\}^k$ é identificada com os coeficientes dos polinômios de $\mathbb{F}_{2^k}$.

Exemplo 103 ($\mathbb{F}_4$, $\mathbb{F}_8$ e $\mathbb{F}_{16}$). O corpo $\mathbb{F}_{2^2}$ é isomorfo a $\mathbb{Z}_2[x]/(x^2+x+1)$ e pode ser representado por $\{0, 1, x, x+1\}$. A representação polinomial de $\mathbb{F}_{2^3}$ com $f(x) = x^3+x+1$ é dada pelos polinômios

$$\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}.$$

A soma é feita como o usual em módulo 2, por exemplo, em $\mathbb{F}_{2^3}$ temos $(x^2+1)+(x^2+x+1) = (1+1)x^2+x+(1+1) = x$ e o produto é o produto usual tomado módulo $f(x)$, por exemplo temos $(x^2+x)(x^2+x+1) = x^4+x$ que módulo $f(x)$ é x^2.

O corpo $\mathbb{F}_{16}$ é isomorfo a $\mathbb{Z}_2[x]/(x^4+x+1)$ e a $\mathbb{Z}_2[x]/(x^4+x^3+x^2+x+1)$.

Exemplo 104 (Desaleatorização de MAX 3-SAT). Sejam $v_1, v_2, \ldots, v_n$ variáveis de uma fórmula 3-CNF, tomemos o corpo $\mathbb{F}_q$ com $q = 2^{\lceil \log_2 n \rceil}$ elementos.

Para $r = (r_0, r_1, r_2) \in_R \mathbb{F}^3$, e $x_1, x_2, \ldots, x_n \in F$ definimos $Y_{x_i}(r) = r_0 + r_1 x_i + r_2 {x_i}^2$, para todo i, $1 \leqslant i \leqslant n$. Pela discussão acima temos $Y_{x_i} \in_R \mathbb{F}$ k-a-k independentes.

Valoramos v_i com o primeiro bit de $Y_{x_i}(r)$, $r \in_R \mathbb{F}_q^3$. Essa valoração aleatória é 3-a-3 independente e satisfaz, em média, pelo menos $7/8$ das cláusulas. Para determinar uma valoração que satisfaz pelo menos $7/8$ das cláusulas, percorremos todo ${\mathbb{F}_q}^3$ e determinamos $(Y_{x_i})_i$ para cada elemento e calculamos o número de cláusulas satisfeitas. Tudo isso é feito em tempo polinomial em n.

2.3.3 *Hash* Universal

Chamamos de **hash** uma função $h : \{0,1\}^m \to \{0,1\}^n$ com $m \geqslant n$. Uma família $\mathcal{H}$ de funções hash é dita **k-universal** se para todo $\{x_1, \ldots, x_k\} \subset \{0,1\}^m$ e quaisquer $y_1, \ldots, y_k \in \{0,1\}^n$ temos

$$\mathbb{P}_{h \in_R \mathcal{H}} [\, h(x_i) = y_i, \text{ para todo } i \,] = \frac{1}{n^k}. \tag{2.7}$$

Exemplo 105 (família 2-universal). Tomemos $\mathcal{H}$ a família das funções $h_{(a,b)} \in \{0,1\}^n \to \{0,1\}^n$, onde $(a,b) \in \{0,1\}^n \times \{0,1\}^n$, tais que

$$h_{(a,b)}(x) = a \cdot x + b,$$

com as operações $+$ e $\cdot$ relativas ao corpo $\mathbb{F}_{2^n}$. Notemos que se $(a,b) \in_R \{0,1\}^n \times \{0,1\}^n$ então para quaisquer $x \neq y$ temos $(h_{(a,b)}(x), h_{(a,b)}(y)) \in_R \{0,1\}^{2n}$ (veja (2.6)), ou seja, para quaisquer $\alpha,\ \beta \in \{0,1\}^n$, se $(a,b) \in_R \{0,1\}^n \times \{0,1\}^n$ então

$$\mathbb{P}\left[\, [h_{(a,b)}(x) = \alpha] \cap [h_{(a,b)}(y) = \beta] \,\right] = \mathbb{P}\left[\, h_{(a,b)}(x) = \alpha \,\right] \mathbb{P}\left[\, h_{(a,b)}(y) = \beta \,\right] = \frac{1}{2^{2n}}.$$

Exercício 106. A definição anterior pode ser relaxada. Dizemos que uma família $\mathcal{H}$ de funções hash é **fracamente k-universal** se para quaisquer $x_1, \ldots, x_k \in \{0,1\}^m$

$$\mathbb{P}_{h \in_R \mathcal{H}}[\, h(x_1) = h(x_2) = \cdots = h(x_k) \,] \leqslant \frac{1}{n^{k-1}}. \tag{2.8}$$

Prove que a família

$$h_{(a,b)}(x) = (ax + b \mod p) \mod 2^n$$

para $a,\ b \in \{0, 1, \ldots, p-1\}$ é fracamente 2-universal se p é um primo com $m+1$ bits na sua representação binária.

Agora, vamos mostrar como construir seqüências pseudoaleatórias a partir de famílias fracamente universais. Seja $\mathcal{H} = \{h_\lambda : \lambda \in \Lambda\}$ uma família de funções hash indexada por Λ, fracamente 2-universal e com $n = m - 4k$, onde k é o número de seqüências pseudoaleatórias que queremos gerar.

Algoritmo 13: **Gerador de Impagliazzo–Zuckerman**

1 $\lambda \leftarrow_R \Lambda$;
2 $Y_1 \leftarrow_R \{0,1\}^m$;
3 $S_1 \leftarrow_R \{0,1\}^{m-n}$;
4 **para cada** $i \in \{2, \ldots, k\}$ **faça**
5 $\quad Y_i \leftarrow h(Y_{i-1})S_{i-1}$;
6 $\quad S_i \leftarrow_R \{0,1\}^{m-n}$;
7 **devolva** $Y_1, Y_2, \ldots, Y_k$.

Na linha 5, $Y_i = h(Y_{i-1})S_{i-1}$ é a concatenação de duas seqüências. O caminho agora é mostrar a distribuição de $Y_1, Y_2, \ldots, Y_k$ é próxima da uniforme.

Considere a distribuição em $\mathcal{H} \times \{0,1\}^n$ dado pela probabilidade de $(\lambda, h_\lambda(r))$ com $\lambda \in_R \Lambda$ e $r \in_R W$ para algum $W \subseteq \{0,1\}^m$. Considere o vetor $P = (P_{\lambda, h_\lambda(r)})$ que na coordenada $(\lambda', h_{\lambda'}(r'))$ vale

$$P_{\lambda', h_{\lambda'}(r')} = \mathbb{P}_{(\lambda, h_\lambda(r))}[\, (\lambda, h_\lambda(r)) = (\lambda', h_{\lambda'}(r')) \,]]$$

e considere o vetor $\Pi = (\Pi_{\lambda, h_\lambda(r)})$ que na coordenada $(\lambda', h_{\lambda'}(r'))$ vale

$$\Pi_{\lambda', h_{\lambda'}(r')} = \frac{1}{|\Lambda| 2^n}$$

dado pela distribuição uniforme em $\mathcal{H} \times \{0,1\}^n$. Queremos estimar a discrepância

$$D(P) = \|P - \Pi\|_1 = \sum_{(\lambda', h_{\lambda'}(r'))} |P_{\lambda', h_{\lambda'}(r')} - \Pi_{\lambda', h_{\lambda'}(r')}|.$$

Pela desigualdade de Cauchy–Schwartz $\|P - \Pi\|_1^2 \leqslant (|\Lambda|2^n)\|P - \Pi\|_2^2$, i.e.,

$$\left(\sum_{(\lambda', h_{\lambda'}(r'))} \left|P_{\lambda', h_{\lambda'}(r')} - \Pi_{\lambda', h_{\lambda'}(r')}\right|\right)^2 \leqslant |\Lambda|2^n \sum_{(\lambda', h_{\lambda'}(r'))} \left|P_{\lambda', h_{\lambda'}(r')} - \Pi_{\lambda', h_{\lambda'}(r')}\right|^2$$

e o lado direito da equação acima é no máximo

$$|\Lambda|2^n \sum_{(\lambda', h_{\lambda'}(r'))} \left(\left|P_{\lambda', h_{\lambda'}(r')}\right|^2 + \left|\Pi_{\lambda', h_{\lambda'}(r')}\right|^2 - 2\frac{P_{\lambda', h_{\lambda'}(r')}}{|\Lambda|2^n}\right)$$

ou seja

$$D(P)^2 \leqslant |\Lambda|2^n \left(\|P\|_2^2 + \|\Pi\|_2^2 - \frac{2}{|\Lambda|2^n}\right) = |\Lambda|2^n \left(\mathbb{P}\left[\ (\lambda, h_\lambda(r)) = (\lambda', h_{\lambda'}(r'))\ \right] - \frac{1}{|\Lambda|2^n}.\right)$$

Sejam $\lambda,\ \lambda' \in_R \Lambda$ e $r,\ r' \in_R W$, então

$$\begin{aligned}
\mathbb{P}\left[\ (\lambda, h_\lambda(r)) = (\lambda', h_{\lambda'}(r'))\ \right] &= \mathbb{P}\left[\ h_\lambda(r) = h_{\lambda'}(r') \mid \lambda = \lambda'\ \right] \mathbb{P}\left[\ \lambda = \lambda'\ \right] \\
&= \mathbb{P}\left[\ h_\lambda(r) = h_\lambda(r')\ \right] \mathbb{P}\left[\ \lambda = \lambda'\ \right] \\
&\leqslant \left(\mathbb{P}\left[\ r = r'\ \right] + \mathbb{P}\left[\ h_\lambda(r) = h_\lambda(r') \mid r \neq r'\ \right]\right) \mathbb{P}\left[\ \lambda = \lambda'\ \right] \\
&\leqslant \left(\frac{1}{|W|} + \frac{1}{2^n}\right) \frac{1}{|\Lambda|} = \left(\frac{2^n}{|W|} + 1\right) \frac{1}{|\Lambda|2^n}.
\end{aligned}$$

Portanto, provamos o seguinte resultado.

Teorema 107 (*Leftover hash lemma*). *Se $\mathcal{H}$ é uma família de funções hash fracamente 2-universal e P é a distribuição em $\mathcal{H} \times \{0,1\}^n$ definida por $\lambda \in_R \Lambda$ e $r \in_R W$ para algum $W \subseteq \{0,1\}^m$, então*

$$D(P) \leqslant \sqrt{\frac{2^n}{|W|}}. \tag{2.9}$$

□

Exercício 108. Seja $L \in BPP$ decidida por uma máquina de Turing probabilística que usa $m = m(|w|)$ bits aleatórios para decidir se $w \in L$. Mostre como reduzir a probabilidade de erro para 2^{-k} em $O(k)$ rodadas da máquina original e usando $O(m + k^2)$ bits aleatórios.

Exercícios complementares

1. Prove que se $P = NP$ então $EXP = NEXP$.

2. Dizemos que uma família de circuitos $\{C_n\}_{n\in\mathbb{N}}$ é **uniforme** se existe uma máquina de Turing de Turing de tempo polinomial em n que com entrada n imprime uma descrição de C_n. Prove que as linguagens que são decididas por circuito uniforme são as linguagens em P.

3. (Adleman (1978)) Defina *circuito booleano aleatorizado* como um circuito booleana que além das n portas da entrada contém portas que recebem um bit uniformemente; esse circuito computa $f\colon \{0,1\}^n \to \{0,1\}$ se quando $f(x_1,\ldots,x_n) = 0$ o circuito com $(x_1,\ldots,x_n)$ responde 0 e quando $f(x_1,\ldots,x_n) = 1$ o circuito com $(x_1,\ldots,x_n)$ responde 1 com probabilidade pelo menos $1/2$. Prove que se $f\colon \{0,1\}^* \to \{0,1\}$ é computada por uma família de circuitos booleanos aleatorizados de tamanho polinomial então f é computada por uma família de circuitos booleanos de tamanho polinomial.

4. Mostre que se NP $\not\subseteq$ P/poly então BPP pode ser desaleatorizado em tempo subexponencial. Não é sabido se NP $\not\subseteq$ P/poly.

5. Dê uma prova direta (isto é, sem usar o teorema 81) de que RP $\subseteq$ P/poly.

6. Alice e Bob mantêm as bases de dados $a = a_1a_2\ldots a_n$ e $b = b_1b_2\ldots b_n$, respectivamente, em binário. Eles desejam saber se a base é a mesma comunicando $O(\log n)$ bits. Escreva um algoritmo em co-RP que resolva esse problema e erre com probabilidade $O(\lg n/n)$.

7. Dois grafos quaisquer $G_1 = (\{1,2,\ldots,n\}, E_1)$ e $G_2 = (\{1,2,\ldots,n\}, E_2)$ são isomorfos se e somente se existe uma permutação $\pi \in \mathbb{S}_n$ tal que $\{i,j\} \in E_1$ se e somente se $\{\pi(i),\pi(j)\} \in E_2$; e π é chamado de *isomorfismo*. O problema do não-isomorfismo de grafos é o seguinte problema computacional.

Dado	: grafos $G_1 = (V, E_1)$ e $G_2 = (V, E_2)$.
Devolve:	*sim* se os grafos são não-isomorfos, *não* caso contrário.

Problema do não-isomorfismo de grafos.

Não se conhece algoritmo de tempo polinomial no tamanho dos grafos para decidir se dois grafos não são isomorfos. Mais do que isso, não se conhece um algoritmo que receba como entrada uma terna (G, H, P) onde P é uma prova de que G e H não são isomorfos e que devolva de tempo polinomial no tamanho de (G, H) *sim* se G_1 não é isomorfo a G_2 e devolva *não* caso contrário, ou seja, *não se sabe se o problema do não-isomorfismo de grafos está na classe* NP *de complexidade computacional.*

Alice e Bob são irmãos gêmeos que cresceram em lugares distantes. Alice que cresceu brincando no jardim da usina de Angra, desenvolveu a estranha habilidade de decidir rapidamente se dois grafos não são isomorfos. Bob não tem essa habilidade, mas estudou probabilidade e usa uma moeda para tomar suas decisões. Nas horas vagas esses irmãos gostam de passar o tempo testando o não-isomorfismo de grafos, mas como a brincadeira não tem muita graça para Alice ela se diverte tentando provar para Bob que os grafos da vez são não-isomorfos, independentes deles o serem ou não. A brincadeira deles tem o seguinte protocolo

Bob:

(a) usa sua moeda (sem que Alice veja) para escolher $i \in_R \{1,2\}$ e para escolher uma permutação $\sigma \in_R \mathbb{S}_n$;

(b) computa $H = \sigma(G_i)$;

(c) mostra H para Alice e pergunta: para qual j os grafos G_j e H são isomorfos?

Alice:

(a) Responde com um elemento j de $\{1,2\}$.

Bob:

(a) Se $j = i$ então Bob aceita que G_1 e G_2 são não-isomorfos, caso contrário rejeita.

Use o princípio da decisão adiada para provar que: *se* G_1 *e* G_2 *são não-isomorfos então Alice consegue sempre convencer Bob, senão Bob é convencido com probabilidade* $1/2$.

Mostre que repetições independentes do protocolo diminuem a probabilidade de Bob ser convencido erroneamente.

8. Seja $\mathcal{H}$ uma família qualquer de funções hash. Dados $x,\ y \in \{0,1\}^m$ e $h \in \mathcal{H}$ defina

$$\delta(x,y,h) = \begin{cases} 1, & \text{se } x \neq y \text{ e } h(x) = h(y) \\ 0, & \text{caso contrário.} \end{cases}$$

Prove que

$$\sum_{h \in \mathcal{H}} \delta(x,y,h) > \frac{|\mathcal{H}|}{2^n} - \frac{|\mathcal{H}|}{2^m}.$$