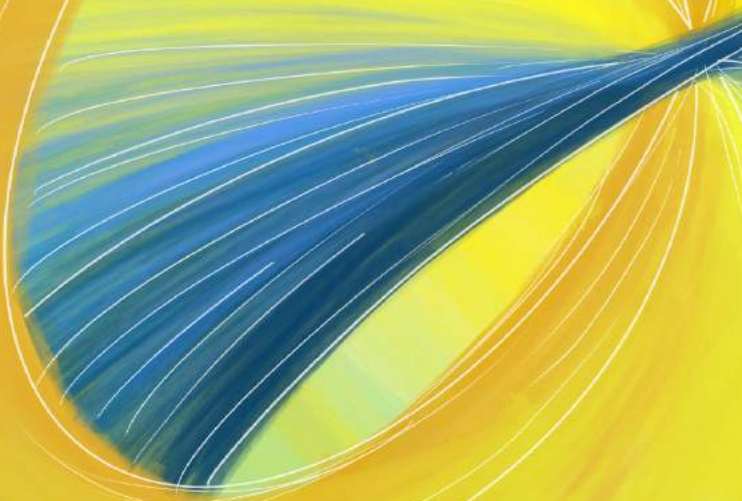
Ciência de dados: algoritmos e aplicações



Luerbio Faria
Fabiano de Souza Oliveira
Paulo Eustáquio Duarte Pinto
Jayme Luiz Szwarcfiter

impa

33º Colóquio Brasileiro de Matemática

Ciência de dados: algoritmos e aplicações

Ciência de dados: algoritmos e aplicações
Primeira impressão, julho de 2021

ISBN 978-65-89124-47-4
MSC (2020) Primary: 68R05, Secondary: 68R99, 05C81, 05C85, 05C99

Coordenação Geral Carolina Araujo

Produção Books in Bytes **Capa** Izabella Freitas & Jack Salvador

Realização da Editora do IMPA
IMPA
Estrada Dona Castorina, 110
Jardim Botânico
22460-320 Rio de Janeiro RJ

www.impa.br
editora@impa.br

Conteúdo

Prefácio

Ciência de Dados é uma área do conhecimento cuja criação é relativamente recente. O interesse por este estudo provém, inicialmente, da quantidade inimaginável de dados que são gerados a cada segundo, por exemplo, na internet e em outros meios de comunicação. Dada a importância dessas informações, e dada a impossibilidade física e lógica de manipular esses dados através dos meios convencionais utilizados até poucos anos atrás, novas técnicas e um novo campo de estudo foram estabelecidos, o que originou a *ciência de dados*. Juntamente com essa nova área foi criada uma nova linha de pesquisa, bastante ampla em sua abrangência. Além disso, abriram-se oportunidades para um novo profissional, denominado *cientista de dados*, cuja procura no mercado de trabalho segue intensa, mesmo no período da presente pandemia.

A ciência de dados poderia ser considerada parte da ciência da computação, mas pela sua imensa aplicação em inúmeras questões do cotidiano e devido ao papel preponderante que a teoria da probabilidade assumiu neste campo de conhecimento, a sua definição como nova área do conhecimento é perfeitamente justificável. Neste aspecto, deve ser ressaltado que ainda não há um consenso geral da abrangência exata da ciência de dados. Assim, dependendo da autoria e da fonte, ciência de dados pode assumir diferentes significados, desde um texto quase puramente matemático, com poucas vinculações diretas relativas a aplicações, até aspectos basicamente humanos de algumas de suas aplicações, sem maiores questões de fundamento. Desta maneira, qualquer texto de ciência de dados reflete a visão de seus autores sobre a matéria. Naturalmente, a presente proposta não poderia deixar de seguir esta linha.

O presente texto se propõe a apresentar um conteúdo de ciência de dados levando em consideração as suas aplicações. Procuramos manter a ênfase algorítmica, ao longo da exposição. Além disso, naturalmente, foi mantida a preocupação pelo rigor matemático necessário para abordar os diversos aspectos da matéria, tanto nas provas matemáticas, quanto na descrição de seu conteúdo.

As partes que compõem este texto de ciência de dados, foram agrupadas em alguns temas, os quais, basicamente compõem os capítulos do presente livro. Os temas englobam conceitos básicos dos seguintes assuntos: *teoria de probabilidades; algoritmos randomizados; análise probabilística de algoritmos; algoritmos de dados massivos; aprendizado de máquina; cadeias de Markov e passeios aleatórios*.

Esses temas são apresentados em uma linguagem acessível a um estudante de graduação, que seria o público alvo desta publicação. Apesar disso, o texto, ou partes do mesmo, podem ser utilizados como material em um curso de pós-graduação.

Gostaríamos de externar os nossas agradecimentos ao Prof. Daniel Ratton, da COPPE/UFRJ, o qual nos sugeriu a área de ciência de dados, através de suas excelentes palestras e conversas particulares.

Agradecemos à Profª Celina Figueiredo, também da COPPE/UFRJ, pela sugestão e incentivo de submeter este texto ao 33º Colóquio Brasileiro de Matemática. Esta submissão certamente veio acelerar a produção do texto.

Não podemos deixar de mencionar a contribuição dos alunos de doutorado Bruno Masquio, Lívia Medeiros, Raquel de Souza, Vitor de Luca, e também daqueles de iniciação científica Antônio de Sousa e Lucas de Carvalho, todos da UERJ, que auxiliaram no processo de revisão e no preparo do curso associado a este texto. Nosso agradecimento a todos eles.

Registramos também o apoio recebido pelo Instituto de Matemática e Estatística da UERJ, instituição de vínculo de seus autores.

Finalmente, agradecemos à Organização do 33º Colóquio Brasileiro de Matemática, bem como à excelência da Comissão Editorial do IMPA, pela cuidadosa revisão e produção do texto.

Rio de Janeiro, Abril 2021
Os Autores

1 Preliminares

1.1 Introdução

Ciência de dados é uma área que se constituiu a relativamente pouco tempo, cujas partes que a compõem ainda não representam um consenso entre seus pesquisadores. Além disso, as partes constituintes ainda se encontram em formação, não estão inteiramente consolidadas.

No presente texto, descrevemos alguns dos principais aspectos da ciência de dados, levando em consideração as suas aplicações, bem como questões de fundamento. É dada ênfase especial aos seus algoritmos, aos métodos gerais dos problemas motivados pelas suas aplicações. Procuramos manter a ênfase algorítmica ao longo do texto, bem como o rigor necessário para descrever os aspectos matemáticos envolvidos.

Os tópicos selecionados foram divididos em capítulos, cujos conteúdos descrevemos a seguir. Uma lista de exercícios e bibliografia comentada encerram cada um dos capítulos.

1.1.1 Introdução à Probabilidade

Nas questões suscitadas em ciência de dados, a probabilidade e a estatística, de um modo geral, assumem um papel preponderante. De fato, não é possível realizar um estudo apropriado desta área sem conhecimentos de probabilidade. Assim sendo, o texto se inicia com um tratamento básico desta matéria, apresentando os principais conceitos e resultados que serão utilizados ao longo do texto. Quando pertinente, as provas dos resultados são incluídas. O texto contém o detalhamento de alguns dos conceitos e propriedades básicos de probabilidade, como probabilidade condicional, independência de eventos, variáveis aleatórias, entre outras. O estudo de alguns tipos de variáveis aleatórias é incluído naquele capítulo com certa ênfase, pois essas variáveis são utilizadas ao longo de todo o livro.

1.1.2 Algoritmos Randomizados

Um dos focos principais do livro é a descrição de algoritmos para resolver os diversos problemas motivados pelo estudo da ciência de dados. Uma grande quantidade desses algoritmos são randomizados e não determinísticos. Assim, é necessário um estudo preliminar desta classe de algoritmos. O estudo se inicia com uma discussão sobre o significado da randomização. Em seguida são abordados os tipos frequentemente utilizados de algoritmos randomizados: os algoritmos de Monte Carlo e de Las Vegas. O estudo é ilustrado por meio da análise de problemas específicos. Para o algoritmo de Monte Carlo, são tratados os problemas da seleção do elemento majoritário de um conjunto; a determinação do corte mínimo de arestas de um grafo; a formulação de um teste de primalidade, entre outros. Em relação ao algoritmo de Las Vegas, são tratados o problema do elemento unitário de um conjunto e o método de Quicksort para ordenação.

1.1.3 Análise Probabilística de Algoritmos

Há um certo número de problemas e algoritmos para os quais a análise de pior caso não é uma abordagem prática ou tão relevante. Um dos exemplos é o método *simplex* para programação linear, cuja complexidade de tempo de pior caso é exponencial, mas que, em termos práticos, obtém resultados quase lineares. Outro exemplo de destaque é o Quicksort, cuja complexidade de pior caso é $O(n^2)$, sendo, na prática, usualmente o algoritmo utilizado, por ser tipicamente de execução muito rápida, superando os algoritmos de complexidade de pior caso $O(n \log n)$. Tais situações podem ser explicadas pelo fato de que entradas "ruins" são de ocorrência

rara. Os bons resultados, quando existentes, são evidenciados pelo estudo da complexidade "típica" do algoritmo, ou seja, a complexidade de caso médio, obtida normalmente por análises probabilísticas. Abordaremos as análises probabilísticas de caso médio para vários algoritmos básicos tais como a busca linear em vetor, busca binária em vetor ordenado e Quicksort.

1.1.4 Algoritmos para Dados Massivos

No estudo clássico da complexidade de algoritmos, um algoritmo é dito eficiente se sua complexidade de tempo é polinomial no tamanho da entrada. Como cada célula de memória alocada por um algoritmo é normalmente inicializada, a complexidade de tempo de um algoritmo é maior ou igual àquela de espaço. Desta maneira, se um algoritmo é eficiente, sua complexidade de espaço também é polinomial.

Tal definição de eficiência, na prática, é insuficiente, pois ela permite polinômios de grau arbitrariamente grande. Um algoritmo com complexidade de tempo expressa por um polinômio de grau 100, por exemplo, é eficiente pela definição, mas não encontra emprego prático. Para uma aplicação real, espera-se que os valores dos graus sejam pequenos, não excedendo algumas poucas unidades. Neste tema, estudaremos algoritmos para os quais o tamanho da entrada é muito grande, além dos limites considerados usuais, de modo que as restrições sobre o grau do polinômio tornam-se ainda mais severas. São fartos os exemplos de aplicações desse tipo particularmente na *web*, onde os serviços *online* servem milhões (ou mesmo, bilhões) de usuários, gerando uma quantidade imensa de dados a serem processados pelas diversas aplicações. Como consequência, a leitura da entrada nestas aplicações deverá ser feita apenas uma única vez, de forma sequencial. Tais algoritmos serão chamados de *algoritmos de dados massivos*. Dada uma sequência de dados massivos, exemplos de problemas que serão abordados incluem: escolher um deles de forma aleatória com distribuição uniforme; estimar a quantidade desses dados que satisfazem certa propriedade, empregando memória muito limitada; determinar a quantidade de dados que são distintos; a pertinência de um elemento arbitrário ao conjunto, entre outros.

1.1.5 Aprendizado de Máquina

Métodos baseados em aprendizado de máquina, para resolver problemas das mais diversas áreas, são cada vez mais utilizados. Em alguns contextos, o aprendizado de máquina é apresentado como a própria ciência de dados, e não como uma parte

da mesma. Esta popularidade se deve, principalmente, ao êxito alcançado por alguns de seus métodos. O sucesso é medido pelos resultados empíricos, às vezes impressionantes. Este livro aborda alguns dos tópicos do aprendizado de máquina. Em especial, o problema de classificação. Uma formulação do algoritmo Perceptron é apresentada em detalhe, incluindo exemplos de aplicação. Descrevemos também a dimensão de Vapnik-Chervonenkis, com uma certa ênfase na utilização dos resultados destes autores em aprendizado de máquina. Finalmente, o texto contém também o método de regressão linear, para o problema de classificação.

1.1.6 Cadeias de Markov

Cadeias de Markov são uma ferramenta poderosa da ciência da computação e da matemática. Suas aplicações incluem a resolução de problemas combinatórios, a previsão do clima, o movimento Browniano, a dispersão de gases, e o espalhamento de doenças.

Vamos ver alguns tópicos relacionados às cadeias de Markov. Mostramos como a matriz potência P^m da matriz de transição P de uma cadeia de Markov é usada para resolver problemas combinatórios. Em especial, o problema da previsão climática. Relacionamos por meio de um algoritmo polinomial, um processo estocástico não Markoviano com um processo Markoviano equivalente. Dentro do estudo das propriedades topológicas, consideramos a classe das cadeias de Markov, cujos estados são unicamente transientes ou absorventes, obtendo os tempos esperados para absorção e a probabilidade de absorção por um determinado estado. E finalmente, mostramos como o método da potência é utilizado para, dada uma cadeia de Markov ergódica, determinar sua distribuição estacionária.

1.1.7 Passeios Aleatórios

Introduzidos em 1905 em um artigo da Nature, os passeios aleatórios são utilizados nas mais diversas áreas de conhecimento. Suas aplicações incluem o estudo do deslocamento de animais, o comportamento de supercordas, e o mercado de ações. Mais recentemente, os passeios aleatórios celebrizaram-se por sua aplicação no algoritmo Pagerank usado como principal ferramenta na máquina de busca da Google.

O capítulo aborda algumas aplicações do tema. Em especial, uma formulação do algoritmo Pagerank é apresentada em detalhe, incluindo um exemplo de aplicação. Na classe dos passeios aleatórios de dimensão 1, estudamos o problema da ruína do jogador e um algoritmo aleatório para o problema 2-SAT.

O presente capítulo introdutório contém conceitos básicos necessários ao bom entendimento do material exposto no livro. O capítulo está dividido em seções. A Seção 2 contém uma descrição da linguagem em que os algoritmos do textos estão formulados. Em seguida, a Seção 3 apresenta os conceitos básicos de complexidade de algoritmos, utilizados ao longo de todo o livro. A seção seguinte contém os fundamentos da tratabilidade de problemas, tópico básico para toda a grande área de algoritmos e complexidade. A Seção 5 descreve as noções básicas de teoria de grafos, cujo conhecimento é necessário para a compreensão dos Capítulos de Cadeias de Markov e Passeios Aleatórios.

1.2 A Linguagem de Descrição dos Algoritmos

Um algoritmo pode ser associado a uma estratégia para resolver um desejado problema. Os dados do problema constituem a *entrada* ou os *dados* do algoritmo. A solução do problema corresponde a sua *saída*. Um *algoritmo determinístico* é caracterizado por uma função f, a qual associa uma saída $S = f(E)$ a cada entrada E. Diz-se, então, que o algoritmo *computa* a função f. Assim sendo, é justificável considerar a entrada como a variável independente básica, em relação à qual são produzidas as saídas do algoritmo, bem como são analisados os comportamentos de tempo e espaço do mesmo.

Neste texto, os algoritmos serão apresentados em uma linguagem quase livre de formato, contendo, porém, sentenças especiais com significado pré-definido. Isto é, na linguagem de apresentação dos algoritmos, há flexibilidade suficiente para que se possa descrever livremente em língua portuguesa a estratégia desejada. Contudo, para tornar a apresentação mais simples e curta, é possível também a utilização de um conjunto de sentenças especiais, às quais correspondem algumas operações que aparecem com grande frequência nos algoritmos.

A linguagem de apresentação assemelha-se às linguagens modernas de programação, porém em português. Todas elas, de certa maneira, herdam a nomenclatura ALGOL, que foi a primeira linguagem de programação amplamente utilizada a empregar estruturação. As ideias gerais nas quais se baseia são as seguintes.

A apresentação de um algoritmo em uma folha de papel é naturalmente dividida em *linhas*. Cada linha possui uma *margem* correspondente ao ponto da folha de papel onde se inicia a linha em questão. Sejam r, s duas linhas de um algoritmo, r antecedendo s. Diz-se que a linha r *contém* a linha s quando (i) a margem de r é mais à esquerda do que a de s e (ii) se $t \neq r$ é uma linha do algoritmo compreendida entre r e s, então a margem de r é também mais à esquerda que a de t.

(a)
p calcular o produto das matrizes M_1, M_2
z somar o produto à matriz M_3
q seja M_4 a matriz resultante

(b)
p calcular o produto das matrizes M_1, M_2
z somar o produto à matriz M_3
q seja M_4 a matriz resultante

(c)
p calcular o produto das matrizes M_1, M_2
z somar o produto à matriz M_3
q seja M_4 a matriz resultante

Figura 1.1: Linhas de um algoritmo.

Por exemplo, a linha p contém a linha q na Figura 1.1(a), mas não a contém na Figura 1.1(b) nem na Figura 1.1(c).

A unidade básica da estrutura de apresentação dos algoritmos é o bloco. Um *bloco* é um conjunto composto de uma linha qualquer r com todas as linhas s que r contém. Consequentemente as linhas que formam um bloco são necessariamente contíguas. Além disso, um bloco é sempre formado por uma sequência de blocos contíguos. Isto é, um bloco B pode ser decomposto como $B = B_1 \cup B_2 \cup \ldots \cup B_k$ onde cada B_i é um bloco, sendo B_j contíguo a B_{j+1}, $1 \leqslant j < k$.

Observe que a margem esquerda de uma linha atua como delimitador de blocos. No exemplo da Figura 1.1(a), o bloco iniciado pela linha p contém z e q. Na Figura 1.1(b), ele contém apenas p, mas o iniciado por z contém q. Na Figura 1.1(c), os blocos com início em p e z são ambos formados por uma única linha.

Dentre os blocos a que uma linha r pertence, um possui especial interesse: aquele definido por r, que é o maior de todos que se iniciam na linha r.

Além da estrutura acima descrita, a linguagem de apresentação dos algoritmos possui também sentenças especiais. A utilização ou não delas é opcional. Contudo, se empregadas devem obedecer à estrutura geral acima. Além disso, cada uma das sentenças possui uma sintaxe própria. Uma sentença especial inicia uma linha da apresentação e, portanto, inicia um bloco chamado *bloco especial*. Este, naturalmente, corresponde à porção do algoritmo onde se estende o efeito da sentença correspondente.

As seguintes sentenças especiais são de interesse.

1. **algoritmo** <*comentário*>

   Se presente, essa linha deve ser a primeira da apresentação do algoritmo e o bloco definido por **algoritmo** deve conter toda a apresentação. O <*comentário*> é opcional e normalmente se refere à finalidade do **algoritmo**.

2. **dados:** <*descrição*>

   Essa sentença é utilizada para informar os dados de entrada do algoritmo. A <*descrição*> deve caracterizar esses dados, em formato livre. Em geral, a única sentença que antecede **dados** é aquela definida no primeiro item.

3. **procedimento/função** <*nome*>

   Um **procedimento** ou **função** altera a ordem sequencial de interpretação dos algoritmos. O bloco definido por **procedimento/função** deve ser saltado na computação sequencial do processo. A interpretação desse bloco é realizada imediatamente após a detecção de uma linha que invoque o <*nome*> do **procedimento/função**. Encerrado este último, a sequência de interpretação retorna para logo após o ponto interrompido. O <*nome*> pode envolver parâmetros. O nome **função** será empregado quando o retorno do algoritmo devolve algum dado computado. Caso contrário, o nome **procedimento** será empregado.

4. **se** <*condição*> **então** <*sentença 1*> **senão** <*sentença 2*>

   Se a <*condição*> é verdadeira então <*sentença 1*> é interpretada e <*sentença 2*> desconsiderada. Se a <*condição*> for falsa, vale o oposto. Em qualquer caso, o bloco seguinte a essa sentença é interpretado em seguida. A parte **senão** <*sentença 2*> pode ser omitida.

5. **para** <*variação*> **:** <*sentença*>

   A <*variação*> consiste de um conjunto $S = \{s_1, \ldots, s_k\}$ e de uma variável j. O bloco definido na sentença é interpretado k vezes, a primeira com $j = s_1$, a segunda com $j = s_2$, e assim por diante. A <*variação*> pode também ser denotada por $j \leftarrow s_1, \ldots, s_k$.

6. **enquanto** <*condição*> **:** <*sentença*>

   A <*condição*> é avaliada e se for verdadeira, o bloco definido por essa sentença é interpretado. Após o qual a <*condição*> é novamente avaliada e o processo se repete. Se a <*condição*> é falsa, então o bloco em questão deve ser saltado.

7. **repetir** <*sentença*> **até que** <*condição*>

   O bloco referente à sentença é interpretado. Ao atingir a linha **até que** a condição é avaliada. Caso seja falsa, todo o processo se repete. Caso contrário, a sequência de interpretação passa à linha seguinte à que contém **até que**.

As demais sentenças podem ser escritas em formato livre, atendendo contudo às condições da estrutura em blocos. Utilizam-se, obviamente, símbolos matemáticos correntes e o *operador de atribuição* $\leftarrow$. Tal operador é empregado na forma

$$x_1, \ldots, x_k \leftarrow y_1, \ldots, y_k$$

que significa atribuir a x_i o valor de y_i, para cada $1 \leqslant i \leqslant k$. Para tais atribuições assumem-se que são feitas de forma simultânea. Do lado esquerdo, assumem-se também que os identificadores $x_1, \ldots, x_k$ são todos distintos.

Os elementos de uma variável s do tipo *vetor* com n elementos serão denotados por $s[1], s[2], \ldots, s[n]$, embora os índices de início e fim possam variar conforme a necessidade.

Como exemplo da utilização da linguagem de apresentação de algoritmos, considere o seguinte caso: é dada uma sequência $s[1], s[2], \ldots, s[n]$ de termos. O objetivo é inverter a sua ordem, isto é, rearranjar os termos na sequência, de modo que $s[n]$ apareça na posição 1, $s[n-1]$ na posição 2, e assim por diante. O Algoritmo 1.1 seguinte descreve o processo.

Algoritmo 1.1 Inversão de uma sequência

Dados: sequência $s[1..n]$
  para $j \leftarrow 1, \ldots, \lfloor n/2 \rfloor$ **:**
    $s[j], s[n-j+1] \leftarrow s[n-j+1], s[j]$

Note que o efeito dessa atribuição do exemplo é de trocar os valores entre os elementos $s[j]$ e $s[n-j+1]$.

1.3 Complexidade de Algoritmos

Na área de algoritmos, de um modo geral, é inerente uma preocupação computacional para com os processos desenvolvidos. Essa preocupação, por sua vez, implica no objetivo de procurar elaborar algoritmos que sejam eficientes, o mais possível. Consequentemente, torna-se imperioso o estabelecimento de critérios que possam avaliar esta eficiência.

No início, os critérios de medida de eficiência eram em geral empíricos, principalmente no que se refere à eficiência de tempo. Baseado em uma certa estratégia, um algoritmo era descrito e implementado. Em seguida, efetuava-se uma avaliação prática de seu comportamento. Isto é, um programa implementando o algoritmo era executado em um computador, para alguns conjuntos de dados de entrada diferentes. Para cada execução, media-se o tempo correspondente. Ao final da experiência eram obtidas algumas curvas destinadas a avaliar o comportamento de tempo do algoritmo.

Em geral, esses eram os critérios utilizados até aproximadamente a primeira metade do século passado. Se bem que as informações obtidas por meio desse processo sejam também úteis, os critérios mencionados não permitem aferir, realmente, o comportamento do algoritmo. E por vários motivos. As curvas obtidas traduzem apenas os resultados de medidas empíricas para dados particulares. Além disso, essas medidas são dependentes de uma implementação particular, portanto, da qualidade de programador, do compilador específico utilizado, do computador empregado e até das condições locais de processamento no instante da realização das medidas.

Esses fatos justificam a necessidade da adoção de algum processo que seja analítico, para avaliação da eficiência. O critério descrito a seguir tenta se aproximar desse objetivo.

A tarefa de definir um critério analítico torna-se mais simples se a eficiência a ser avaliada for relativa a alguma máquina específica. Recordamos que o método de avaliação empírico acima mencionado, produz sempre resultados relativos ao computador utilizado nas experiências. Ao invés de escolher uma máquina particular, em relação à qual a eficiência dos algoritmos seria avaliada, é certamente mais conveniente utilizar um modelo matemático de um computador. Uma possível formulação desse modelo é a *RAM* (*Random Access Machine*), que é composta de uma *unidade de entrada, unidade de saída, memória e controle/processador*

O funcionamento de uma RAM é semelhante ao de um computador hipotético elementar. O processador dispõe de *instruções* que podem ser executadas. A memória armazena os *dados* e o *programa*. Esse último consiste de um conjunto de instruções que implementa o algoritmo. Cada instrução I do modelo possui um *tempo de instrução* $t(I)$. Assim sendo, se para a execução de um programa P, para uma certa entrada fixa, são processadas r_1 instruções do tipo I_1, r_2 instruções do tipo $I_2, \ldots, r_m$ instruções do tipo I_m, então o *tempo de execução* do programa P é dado por $\sum_{j=1}^{m} r_j \cdot t(I_j)$. O que se segue é uma tentativa de avaliação desse somatório.

Para completar a descrição do modelo a ser utilizado, introduzimos a seguinte

simplificação adicional. Supõe-se que $t(I) = 1$ para toda instrução I, isto é, que o tempo de execução de cada instrução seja constante e igual a 1. À primeira vista, essa simplificação talvez se afigure como não razoável. Para justificá-la, observe inicialmente que a relação $t(I_1)/t(I_2)$ entre os tempos de execução das instruções I_1 e I_2 pode ser aproximada a uma constante, considerando-se fixos os tamanhos dos operandos. Por exemplo, uma instrução de multiplicação seria aproximadamente k, constante, vezes mais lenta que uma comparação. Essa simplificação parece razoável. Sejam agora T' e T'' respectivamente, os valores dos tempos de execução do programa, para os caso em que $t(I_1)/t(I_2)$ = constante e $t(I) = 1$. Isto é, T' corresponde ao valor do tempo de execução do programa, supondo apenas a simplificação de que a relação entre dois tempos de instruções arbitrárias seja uma constante. Enquanto isso, T'' corresponde ao tempo de execução do programa no caso em que os tempos de instruções são todos unitários. Segue-se, então, que existe uma constante que limita T'/T'', ou seja, a simplificação adicional $t(I) = 1$ introduz uma distorção de apenas uma constante, em relação à simplificação anterior $t(I_1)/t(I_2) = k$.

Com $t(I) = 1$, o valor do tempo de execução de um programa torna-se igual ao número total de instruções computadas. Denomina-se *passo de um algoritmo* α à computação de uma instrução do programa P que o implementa. A *complexidade local* do algoritmo α é o número total de passos necessários para a computação completa de P, para uma certa entrada E. Assim sendo, considerando-se as simplificações introduzidas, a complexidade local de α é equivalente ao seu tempo de execução, para a entrada E. O interesse é determinar o número total de passos acima, para entradas consideradas suficientemente grandes.

Nesse sentido, torna-se relevante avaliar o tamanho da entrada. Supondo que ela seja composta de n símbolos, o seu comprimento seria o somatório dos tamanhos das codificações correspondentes aos n símbolos, segundo o critério de codificação utilizado. Para simplificar a análise, a menos que seja explicitamente dito o contrário, o tamanho da codificação de cada símbolo será considerado constante. Assim sendo, o tamanho da entrada pode ser expresso por um número proporcional a n, sendo considerado grande quando n o for. Essa simplificação é amplamente satisfatória quando a codificação de cada símbolo puder ser armazenada em uma palavra de computador, como ocorre frequentemente. Nesse caso, o número de palavras utilizadas exprime obviamente o comprimento da entrada.

A avaliação da eficiência para problemas grandes é, sob alguns aspectos, a mais importante. Além disso, facilita a manipulação de sua expressão analítica. Consequentemente, seria também razoável aceitar uma medida analítica que refletisse um limite superior para o número de passos, em lugar de um cálculo exato.

Define-se, então, *complexidade (local) assintótica* de um algoritmo como sendo um limite superior da sua complexidade local, para uma certa entrada suficientemente grande. Naturalmente, em se tratando de um limite superior, o interesse é determiná-lo o mais justo, ou seja, o menor possível. A complexidade assintótica deve ser escrita em relação às variáveis que descrevem o tamanho da entrada do algoritmo.

Para exprimir analiticamente a complexidade assintótica, é conveniente utilizar a notação seguinte, denominada *notação O*, empregada para expressar ordens de grandeza de funções. Sejam f, h funções reais não negativas da variável inteira $n \geqslant 0$. Diz-se que f é $O(h)$, denotando-se $f = O(h)$, quando existirem constantes $c, n_0 > 0$ tais que $f \leqslant ch$, para $n \geqslant n_0$. Por exemplo, $3x^2 + 2x + 5$ é $O(x^2)$, $3x^2 + 2x + 5$ é $O(x^3)$, $4x^2 + 2y + 5x$ é $O(x^2 + y)$, $2x + \log x + 1$ é $O(x)$, 542 é $O(1)$, e assim por diante. É imediato verificar que $O(h_1 + h_2) = O(h_1) + O(h_2)$ e $O(h_1 \cdot h_2) = O(h_1) \cdot O(h_2)$. Além disso, se $h_2 = O(h_1)$, então $O(h_1 + h_2) = O(h_1)$. Seja k uma constante, então $O(k \cdot h) = k \cdot O(h) = O(h)$. Observe que os termos de ordem mais baixa são irrelevantes. Quando a função f é $O(h)$, diz-se também que f é da *ordem* de h.

A notação O, naturalmente, é útil para descrever limites superiores. Por outro lado, muitas vezes desejamos mencionar um limite superior que não seja justo. Para esses casos, utilizamos a notação denominada *o pequeno*, assim definida.

Sejam f, h funções reais não negativas da variável inteira $n \geqslant 0$. Diz-se que f é $o(h)$, denotando-se $f = o(h)$, quando, para uma dada constante c, existir constante $n_0 > 0$ tais que $f < ch$, para $n \geqslant n_0$. Por exemplo, x^2 é $o(x^3)$, mas não é $o(x^2)$, apesar de ser $O(x^2)$. Assim sendo, a notação O se presta para descrever limites superiores, enquanto que a notação o descreve limites superiores não justos.

A complexidade assintótica de um algoritmo obviamente não é única, pois a entradas diferentes podem corresponder números de passos diferentes. Dentro dessa diversidade de entradas, é sem dúvida importante aquela que corresponde ao *pior caso*. Talvez para a maioria das aplicações esse é o caso mais relevante. Além disso, é de tratamento analítico mais simples. Define-se, então, *complexidade de pior caso* (ou simplesmente *complexidade*) de um algoritmo como o valor máximo dentre todas as suas complexidades assintóticas, para entradas de tamanho suficientemente grandes. Ou seja, a complexidade de pior caso traduz um limite superior do número de passos necessários à computação da entrada mais desfavorável de tamanho suficientemente grande.

A complexidade de um algoritmo é, sem dúvida, um indicador importante para a avaliação da sua eficiência de tempo. Mas, certamente, também possui aspectos

desvantajosos e tampouco é o único indicador existente. A complexidade procura traduzir analiticamente uma expressão da eficiência de tempo do pior caso. Contudo, há exemplos em que é por demais pessimista. Isto é, o pior caso pode corresponder a um número de passos bastante maior do que os casos mais frequentes. Além disso, a expressão da complexidade não considera as constantes, sendo um outro fator onde distorções podem ser introduzidas. Seja o exemplo em que o número de passos necessários para a computação do pior caso de um algoritmo é $c_1x^2 + c_2x$ onde c_1, c_2 são constantes com $c_1 \ll c_2$ e x uma variável que descreve o tamanho da entrada. Então, como a complexidade é assintótica, ela seria escrita como $O(x^2)$, o que pode não exprimir satisfatoriamente as condições do exemplo.

Por analogia, define-se também um indicador para o melhor caso do algoritmo. Assim sendo, a *complexidade de melhor caso* é o valor mínimo dentre todas as complexidades assintóticas do algoritmo, para entradas suficientemente grandes. Isto é, a complexidade de melhor caso corresponde a um limite superior do número de passos necessários à computação da entrada mais favorável de tamanho suficientemente grande. Analogamente ao pior caso, esse novo indicador deve ser expresso em função do tamanho da entrada.

A seguinte notação, denominada Ω é útil no estudo de complexidade. Sejam f, h funções reais não negativas da variável inteira $n > 0$. Então $f = \Omega(h)$ significa que existem constantes $c, n_0 > 0$ tais que $f \geqslant ch$, para $n \geqslant n_0$. Por exemplo, $5n^2 + 2n + 3$ é $\Omega(n^2)$, mas é também $\Omega(n)$. Por outro lado, $2n^3 + 5n$ é $\Omega(n^2)$, e assim por diante. Assim sendo, a notação Ω é adequada para exprimir limites inferiores.

Finalmente, utilizamos a *notação* Θ, adequada para exprimir valores de complexidades justos. Sejam f, h funções reais não negativas da variável inteira $n > 0$. Então $f = \Theta(h)$ se e somente se f é $O(h)$ e também é $\Omega(h)$. Por exemplo, x^3 é $\Theta(x^3)$, mas não $\Theta(x^4)$, nem $\Theta(x^2)$.

Além de considerar as medidas relativas ao pior e melhor caso, na execução de um algoritmo, surge naturalmente a ideia de computar o caso médio da complexidade, dentre todas as entradas possíveis. Para conceituar esse indicador, consideramos o seguinte modelo.

Seja $E_1, \ldots, E_m$ o conjunto das entradas de um algoritmo. É conhecida uma distribuição das probabilidades de ocorrências dessas entradas, bem como o número de passos executados em cada caso. Assim, supomos que a entrada E_i possua probabilidade p_i e requeira a execução de t_i passos. A *complexidade de caso*

médio de um algoritmo é dada pela seguinte expressão

$$\sum_{i=1}^{m} p_i . t_i$$

A complexidade de caso médio, naturalmente, é das mais relevantes para avaliar a qualidade de um algoritmo. Comentamos, contudo, as dificuldades que surgem na sua determinação. A primeira é o conhecimento prévio da distribuição das probabilidades. Além disso, a sua computação, em muitos casos, não é uma tarefa simples, em particular, a obtenção de uma fórmula fechada de sua expressão.

Examinamos, em seguida, as complexidades de todos os algoritmos que possam resolver um certo problema. Seja P um problema algorítmico, cuja entrada possui tamanho $n > 0$ e g uma função real positiva da variável n. Diz-se que $\Omega(g)$ é um *limite inferior* de P quando qualquer algoritmo α que resolva P requerer pelo menos $O(g)$ passos. Isto é, se $O(f)$ for a complexidade (pior caso) de α, então g é $O(f)$. Por exemplo, se $\Omega(n^2)$ for um limite inferior para P, não poderá existir algoritmo que resolva em $O(n)$ passos. Um algoritmo α_0 que possua complexidade $O(g)$ é denominado *ótimo* e, nesse caso, g é o *limite inferior máximo* de P. Observe que α_0 é o algoritmo de complexidade mais baixa dentre todos os que resolvem P. Esse último indicador é obviamente importante e, frequentemente, de difícil determinação. É também mais geral que os anteriores, pois é relativo ao problema e não a alguma solução específica.

1.4 Tratabilidade de Problemas

Na Seção 1.3, examinamos a questão da conceituação da complexidade computacional. Restou a questão importante de distinguir, por meio da complexidade, os algoritmos eficientes daqueles não eficientes. Essa questão será examinada nesta seção.

Utiliza-se o seguinte critério. Um *algoritmo eficiente* é precisamente aquele cuja complexidade pode ser expressa por um polinômio no tamanho de sua entrada. Caso contrário, será classificado como *não eficiente*. Para que essas definições façam sentido é necessário supor que a informação dos dados de entrada de um algoritmo sejam codificados de maneira a não conter elementos espúrios, e codificados de modo a não alterar o caráter, polinomial ou não, do tamanho de sua entrada. Normalmente, os dados de entrada dos problemas são codificados em binário.

Consideramos, em seguida, a seguinte extensão desse conceito. Seja Π um problema algorítmico, e examinemos a coleção de todos os possíveis algoritmos que o resolvem. Dizemos que Π é um *problema tratável* se existir algum algoritmo eficiente que o resolva. Caso contrário, denomina-se *intratável*.

Os problemas algorítmicos, que consideramos nesta seção, são os denominados *problemas de decisão*, que são precisamente aqueles cuja resposta é apenas uma informação binária "sim" ou "não". Há uma maneira padronizada de formular problemas de decisão. Assim, representamos um problema de decisão Π através da dupla (D, Q), onde D corresponde ao conjunto de dados de Π, e Q representa a questão que Π deve resolver, ao fornecer de uma resposta "sim" ou "não". Por exemplo, o problema NÚMEROS PRIMOS consiste em decidir se um dado inteiro k é primo.

NÚMEROS PRIMOS
DADOS: Inteiro $k > 0$
QUESTÃO: O inteiro k é primo ?

Desta forma, os dados D do problema NÚMEROS PRIMOS $\Pi(D, Q)$ consistem na codificação binária do inteiro k, e a questão Q consiste em decidir se k é primo.

Observamos que a codificação binária de k possui comprimento igual a $1 + \lfloor \log_2 k \rfloor$ dígitos binários. Assim, para que este problema seja tratável, é necessário a existência de um algoritmo cuja complexidade seja um polinômio em $O(\log_2 k)$. Tal algoritmo foi concebido há alguns anos. Portanto, esse algoritmo possui complexidade polinomial, o que implica que NÚMEROS PRIMOS é um problema tratável.

1.4.1 As classes $\mathcal{P}$ e $\mathcal{NP}$

A classe de problemas $\mathcal{P}$ consiste dos problemas de decisão tratáveis, isto é que admitem algoritmo polinomial no tamanho de sua entrada. Por exemplo, já verificamos que NÚMEROS PRIMOS $\in \mathcal{P}$.

Consideramos, em seguida, uma variação da questão relacionada aos problemas de decisão. Dado um problema de decisão $\Pi(D, Q)$, em lugar de decidir "sim" ou "não", para o problema Π, desejamos apenas certificar uma dada resposta "sim". Isto é, se for exibida uma justificativa para a resposta "sim", o objetivo consiste tão somente em confirmar, por meio de um algoritmo polinomial no tamanho de Π, que a resposta é válida. Essa justificativa é denominada *certificado*. Por

exemplo, seja o problema de decisão DIVISÃO INTEIRA, descrito em seguida:

DIVISÃO INTEIRA
DADOS: Inteiros a, b, com $b \neq 0$.
QUESTÃO: O inteiro a é múltiplo de b ?

Um certificado para a resposta "sim" de DIVISÃO INTEIRA consiste em exibir um inteiro c, tal que $b.c = a$. É imediato verificar que esse certificado pode ser verificado em tempo polinomial no tamanho da entrada de DIVISÃO INTEIRA.

A classe de problemas $\mathcal{NP}$ consiste dos problemas de decisão que admitem certificados que podem ser verificados por meio de um algoritmo de tempo polinomial no tamanho dos problemas considerados. Assim, DIVISÃO INTEIRA $\in \mathcal{NP}$.

Do exposto acima, segue-se que se um problema de decisão $\Pi \in \mathcal{P}$, então $\Pi \in \mathcal{NP}$. Isto é, $\mathcal{P} \subseteq \mathcal{NP}$. A pergunta natural é se essa contenção é de fato uma igualdade, isto é, motiva a pergunta

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

Ainda não se conhece a resposta a essa questão. De fato, é uma das questões em aberto mais importantes da matemática contemporânea!

1.4.2 A Classe NP-completo

Sejam $\Pi_1(D_1, Q_1)$ e $\Pi_2(D_2, Q_2)$ problemas de decisão. Suponha que seja conhecido um algoritmo A_2 para resolver Π_2. Se for possível transformar o problema Π_1 em Π_2, sendo conhecido um processo de transformar a solução de Π_2 numa solução de Π_1, então o algoritmo A_2 pode ser utilizado para resolver o problema Π_1. Observe que, como o objetivo consiste em resolver Π_1 através de A_2, o que se supõe dado é uma instância de Π_1 para a qual se deseja conhecer a resposta. A partir da instância $I_1 \in D_1$, elabora-se a instância $I_2 \in D_2$. Aplica-se, então, o algoritmo A_2 para resolver Π_2. O retorno ao problema Π_1 é realizado por meio da transformação da solução de Π_2 para a de Π_1. Se a transformação de Π_1 em Π_2, bem como a da solução de solução de Π_2 na solução de Π_1, puder ser realizada em tempo polinomial, então diz-se que existe uma *transformação polinomial* de Π_1 em Π_2 e que Π_1 é *polinomialmente transformável* em Π_2. Formalmente, uma *transformação polinomial* de um problema de decisão $\Pi_1(D_1, Q_1)$ no problema de decisão $\Pi_2(D_2, Q_2)$ é uma função $f : D_1 \rightarrow D_2$ tal que as seguintes condições são satisfeitas:

- (i) f pode ser computada em tempo polinomial;
- (ii) para toda instância $I \in D_1$ do problema Π_1, tem-se: $\Pi_1(I)$ possui resposta "sim" se e somente se $\Pi_2(f(I))$ também o possuir.

Observe que as transformações de maior interesse, entre problemas, são precisamente as polinomiais. Isto ocorre porque essas últimas preservam a natureza (polinomial ou não) do algoritmo A_2 para Π_2, quando utilizado para resolver Π_1. Assim sendo, se A_2 for polinomial e existir uma transformação polinomial de Π_1 em Π_2, então Π_1 também pode ser resolvido em tempo polinomial. Denota-se $\Pi_1 \propto \Pi_2$ (lê-se "Π_1 se transforma polinomialmente a Π_2") para indicar que Π_1 pode ser transformado polinomialmente em Π_2. Observe que a relação $\propto$ é transitiva (isto é, $\Pi_1 \propto \Pi_2$ e $\Pi_2 \propto \Pi_3 \Rightarrow \Pi_1 \propto \Pi_3$).

Observe que, quando $\Pi_1 \propto \Pi_2$, a transformação f deve ser aplicada sobre uma instância I genérica de Π_1. A instância que se obtém de Π_2, contudo, é particular, pois é fruto da transformação f utilizada. Assim sendo, de certa forma, Π_2 pode ser considerado como um problema de dificuldade maior ou igual a Π_1. Pois que, mediante f, um algoritmo que resolve a instância particular de Π_2, obtida por meio de f, resolve também a instância arbitrária de Π_1 dada.

Considere agora dois problemas Π_1 e Π_2, tais que $\Pi_1 \propto \Pi_2$ e $\Pi_2 \propto \Pi_1$. Então Π_1 e Π_2 são denominados *problemas equivalentes*. Assim, segundo a observação acima, dois problemas equivalentes são de idêntica dificuldade, no que se refere à existência ou não de algoritmo polinomial para resolvê-los. Note que, quando $\Pi_1 \propto \Pi_2$, a particularização de Π_2 construída pela transformação f é equivalente ao problema geral Π_1. Isto é, representado por $f(D_1)$, o conjunto imagem de $f : D_1 \to D_2$, o problema $\Pi_1(D_1, Q_1)$ é equivalente à particularização $(f(D_1), Q_2)$ do problema $\Pi_2(D_2, Q_2)$.

Observe que é possível utilizar a relação $\propto$ para dividir $\mathcal{NP}$ em classes de problemas equivalentes entre si. Obviamente, os problemas pertencentes a $\mathcal{P}$ formam uma dessas classes. E, nesse sentido, podem ser considerados como os de "menor dificuldade" em $\mathcal{NP}$.

Em contrapartida, existe outra classe de problemas equivalentes entre si, que correspondem aos de "maior dificuldade" dentre todos em $\mathcal{NP}$. Os problemas dessa nova classe são denominados *NP-completo*, cuja definição se segue. Um problema de decisão Π é denominado *NP-completo* quando as seguintes condições forem ambas satisfeitas:

(i) $\Pi \in \mathcal{NP}$;

(ii) todo problema de decisão $\Pi' \in \mathcal{NP}$ satisfaz $\Pi' \propto \Pi$.

Observamos que (ii) implica que todo problema da classe $\mathcal{NP}$ pode ser transformado polinomialmente no problema NP-completo. Isto é, se um problema NP-completo Π puder ser resolvido em tempo polinomial, então *todo* problema de $\mathcal{NP}$ admite também algoritmo polinomial e, consequentemente, nessa hipótese $\mathcal{P} = \mathcal{NP}$. Vale, pois, $\Pi \in \mathcal{P}$ se e somente se $\mathcal{P} = \mathcal{NP}$, justificando o fato de que a classe NP-completo corresponda aos problemas de maior dificuldade dentre os pertencentes a $\mathcal{NP}$.

Caso somente a condição (ii) da definição de NP-completo seja considerada, não importando se (i) é ou não satisfeita, o problema Π é denominado *NP-difícil*. Consequentemente, a "dificuldade" de um problema NP-difícil é não menor do que a de um problema NP-completo.

Para que a definição de NP-completo seja relevante é necessário identificar problemas que pertençam à classe. O primeiro provado ser NP-completo é o problema SATISFATIBILIDADE da área de lógica, cuja formulação é a seguinte.

SATISFATIBILIDADE (SAT)
DADOS: Uma expressão booleana E na FNC
QUESTÃO: E é satisfatível?

Para enunciar o primeira problema da lista, considere um conjunto de variáveis booleanas $x_1, x_2, x_3, \ldots$, e denote por $\overline{x}_1, \overline{x}_2, \overline{x}_3, \ldots$ seus complementos. Isto é, cada variável x_i assume um valor *verdadeiro* (V) ou *falso* (F) e x_i é verdadeira se e somente se $\overline{x}_i$ for falso. De um modo geral, os símbolos $x_1, \overline{x}_1, x_2, \overline{x}_2, x_3, \overline{x}_3, \ldots$ são denominados *literais*. Denote por $\wedge$ e $\vee$, respectivamente, as operações binárias usuais de conjunção (e) e disjunção (ou). Uma *cláusula* é uma disjunção de literais. Uma *expressão booleana* é uma expressão cujos operandos são literais e cujos operadores são conjunções ou disjunções. Uma expressão booleana é dita na *forma normal conjuntiva* (FNC) quando for uma conjunção de cláusulas. Se atribuirmos um valor, verdadeiro ou falso, a cada variável de uma expressão booleana E, podemos computar o *valor*, V ou F, de E, calculando-se os resultados das operações de conjunção e disjunção indicadas na expressão. Sabe-se que toda expressão booleana pode ser colocada na FNC, mediante a aplicação de transformações que preservam o seu valor. Uma expressão booleana é dita *satisfatível* se existe uma atribuição de valores, verdadeiro ou falso, às variáveis de tal modo que o valor da expressão seja verdadeiro. O *problema de satisfatibilidade (SAT)* consiste em, dada uma expressão booleana na FNC, verificar se ela é satisfatível.

O problema 3-SAT é uma restrição do problema SAT, em que todas as cláusulas possuem, no máximo, três literais. Quando restringimos um problema, claro, ele se torna mais simples, e, nesse caso, pode não mais pertencer à classe NP-completo. Contudo, pode ser provado, que o problema 3-SAT permanece NP-completo. Similarmente, o problema 2-SAT é uma restrição de 3-SAT, em que cada cláusula possui, no máximo, dois literais. Essa nova restrição produz o efeito de tornar o problema tratável, isto é, existe um algoritmo de tempo polinomial que resolve 2-SAT. Então 2-SAT $\in \mathcal{P}$.

Observamos, ainda, que se um problema Π pertence à classe NP-completo, nada é possível afirmar quanto a sua tratabilidade. Tanto é possível $\Pi \in \mathcal{P}$ ou não. Por outro lado, existe uma quantidade enorme de problemas de interesse, seja teórico ou aplicado, que são atualmente classificados como NP-completos. Esse fato, aliado ao desafio matemático, torna a questão $\mathcal{P} = \mathcal{N}P$ ainda mais intrigante.

1.5 Noções Básicas de Teoria de Grafos

Um *grafo* $G(V, E)$ é um conjunto finito não vazio V e um conjunto E de pares não ordenados de elementos distintos de V. O grafo $G(V, E)$ é chamado *trivial* quando $|V| = 1$. Se necessário, utilizamos o termo *grafo não direcionado*, para designar um grafo. Os elementos de V são os *vértices* e os de E são as *arestas* de G respectivamente. Cada aresta $e \in E$ será denotada pelo par de vértices $e = vw$ que a forma. Nesse caso, os vértices v, w são os *extremos* (ou *extremidades*) da aresta e, sendo denominados *adjacentes*. A aresta e é dita *incidente* a ambos v, w. Duas arestas que possuem um extremo comum são chamadas de *adjacentes*. Utilizaremos a notação $n = |V|$ e $m = |E|$.

Um grafo pode ser visualizado por meio de uma *representação geométrica*, na qual seus vértices correspondem a pontos distintos do plano em posições arbitrárias, enquanto que a cada aresta vw é associada uma linha arbitrária unindo os pontos correspondentes a v, w. Para maior facilidade de exposição, é usual confundir-se um grafo com a sua representação geométrica. Isto é, no decorrer do texto, será utilizado o termo *grafo* significando também a sua representação geométrica.

Em um grafo $G(V, E)$, define-se *grau* de um vértice $v \in V$, denotado por $d(v)$, como sendo o número de vértices adjacentes a v. Um grafo é *regular* de *grau* r, quando todos os seus vértices possuírem o mesmo grau r. Observe que cada vértice v é incidente a $d(v)$ arestas e cada aresta é incidente a 2 vértices.

Logo, $\sum_{v \in V} d(v) = 2|E|$. Um vértice que possui grau zero é chamado *isolado*.

Uma sequência de vértices $v_1, \ldots, v_k$ tal que $v_j v_{j+1} \in E$, $1 \leqslant j < k$, é denominado *caminho* de v_1 a v_k. Diz-se, então, que v_1 *alcança* ou *atinge* v_k. Um caminho de k vértices é formado por $k-1$ arestas $v_1v_2, v_2v_3, \ldots, v_{k-1}v_k$. O valor $k-1$ é o *comprimento* do caminho. Se todos os vértices do caminho $v_1, \ldots, v_k$ forem distintos, a sequência recebe o nome de *caminho simples* ou *elementar*. Se as arestas forem distintas, a sequência denomina-se *trajeto*. Um *ciclo* é um caminho $v_1, \ldots, v_k, v_{k+1}$ sendo $v_1 = v_{k+1}$ e $k \geqslant 3$. Se o caminho $v_1, \ldots, v_k$ for simples, o ciclo $v_1, \ldots, v_k, v_{k+1}$ também é denominado *simples* ou *elementar*. Um grafo que não possui ciclos é *acíclico*. Um *triângulo* é um ciclo de comprimento 3. Dois ciclos são considerados idênticos se um deles puder ser obtido do outro, por meio de uma rotação de seus vértices. Para maior simplicidade de apresentação, quando não houver ambiguidade, os termos "caminho" e "ciclo" serão utilizados com o significado de "caminho simples" e "ciclo simples", respectivamente.

Um grafo $G(V, E)$ é denominado *conexo* quando existe caminho entre cada par de vértices de G. Caso contrário G é *desconexo*. Observe que a representação geométrica de um grafo desconexo é necessariamente descontínua. Em especial, o grafo G será *totalmente desconexo* quando não possuir arestas.

Seja S um conjunto e $S' \subseteq S$. Diz-se que S' é *maximal* em relação a uma certa propriedade P, quando S' satisfaz à propriedade P e não existe subconjunto $S'' \supset S'$, que também satisfaz P. Observe que a definição anterior não implica necessariamente que S' seja o *maior* subconjunto de S satisfazendo P. Implica apenas no fato de que S' não está propriamente contido em nenhum subconjunto de S que satisfaça P. De maneira análoga, define-se também conjunto *minimal* em relação a uma certa propriedade. A noção de conjunto maximal (ou minimal) é frequentemente encontrada em combinatória. Ela pode ser aplicada, por exemplo, na definição que se segue.

Denominam-se *componentes conexos* de um grafo G os subgrafos maximais de G que sejam conexos. Observe que a propriedade P, nesse caso, é equivalente a "ser conexo". Os componentes conexos de um grafo G são os subgrafos de G correspondentes às porções contíguas de sua representação geométrica. Denomina-se *distância* $d(v, w)$ entre dois vértices v, w de um grafo o comprimento do menor caminho entre v e w.

Seja $G(V, E)$ um grafo, $e \in E$ uma aresta. Denota-se por $G - e$ o grafo obtido de G, pela exclusão da aresta e. Se v, w é um par de vértices não adjacentes em G, a notação $G + vw$ representa o grafo obtido adicionando-se a G a aresta vw. Analogamente, seja $v \in V$ um vértice de G. O grafo $G - v$ denota aquele obtido de G pela remoção do vértice v. Observe que excluir um vértice implica em remover

de G o vértice em questão e as arestas a ele incidentes. Da mesma forma, $G + w$ representa o grafo obtido adicionando-se a G o vértice w.

Observa-se ainda que essas operações de inclusão e exclusão de vértices e arestas podem ser generalizadas. De um modo geral, se G é um grafo e S um conjunto de arestas ou vértices, $G - S$ e $G + S$ denotam, respectivamente, o grafo obtido de G pela exclusão e inclusão de S.

Um grafo é *completo* quando existe uma aresta entre cada par de seus vértices. Utiliza-se a notação K_n, para designar um grafo completo com n vértices. O grafo K_n possui o número máximo possível de arestas para um dado n, ou seja, $\binom{n}{2}$ arestas.

Um grafo $G(V, E)$ é *bipartido* quando o seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1, V_2, tais que toda aresta de G une um vértice de V_1 a outro de V_2. É útil denotarmos o grafo bipartido G por $(V_1 \cup V_2, E)$. Um grafo *bipartido completo* possui uma aresta para cada par de vértices v_1, v_2, sendo $v_1 \in V_1$ e $v_2 \in V_2$. Sendo $n_1 = |V_1|$ e $n_2 = |V_2|$, um grafo bipartido completo é denotado por K_{n_1,n_2} e obviamente possui $n_1 n_2$ arestas. Pode-se indagar se existe algum processo eficiente para se reconhecer grafos bipartidos. A resposta é afirmativa.

Um *subgrafo* $G_2(V_2, E_2)$ de um grafo $G_1(V_1, E_1)$ é um grafo tal que $V_2 \subseteq V_1$ e $E_2 \subseteq E_1$. Se, além disso, G_2 possuir toda aresta vw de G_1 tal que ambos v e w estejam em V_2, então G_2 é o *subgrafo induzido pelo subconjunto de vértices* V_2. Diz-se, então, que V_2 *induz* G_2.

Denomina-se *clique* de um grafo G a um subgrafo de G que seja completo. Chama-se *conjunto independente de vértices* a um subgrafo induzido de G, que seja totalmente desconexo. Ou seja, numa clique existe uma aresta entre cada par de vértices distintos. Num conjunto independente de vértices não há aresta entre qualquer par de vértices. O *tamanho* de uma clique ou conjunto independente de vértices é igual à cardinalidade de seu conjunto de vértices. Dado um grafo G, o problema de encontrar em G uma clique ou conjunto independente de vértices com um dado tamanho k pode ser facilmente resolvido por um processo de força bruta, examinando o subgrafo induzido de cada um dos $\binom{n}{k}$ subconjuntos de V com k vértices. Esse método corresponde a um algoritmo de complexidade $\Omega(n^k)$, ou seja, não é eficiente para k arbitrário. De fato, esses problemas pertencem à classe NP-completo, o que significa que é desconhecido se existe algum processo para resolvê-los de modo eficiente.

Um *corte de vértices* de um grafo conexo G é um subconjunto de vértices cuja remoção desconecta G ou o reduz a um grafo trivial. Um *corte de arestas* de um grafo conexo G é um subconjunto de arestas cuja remoção desconecta G.

Em um grafo completo $K_n, n > 1$ não existe subconjunto próprio de vértices $V' \subset V$ cuja remoção desconecte G, mas removendo-se $n - 1$ vértices resulta o grafo trivial. Denomina-se *conectividade de vértices* c_V de G à cardinalidade do menor corte de vértices de G. Analogamente, a *conectividade de arestas* c_E de G é igual à cardinalidade do menor corte de arestas de G. Se G é um grafo desconexo, então $c_V = c_E = 0$. Sendo k um inteiro positivo, diz-se que um grafo G é *k-conexo em vértices* quando a sua conectividade de vértices é $\geqslant k$. Analogamente G é *k-conexo em arestas* quando sua conectividade de arestas é $\geqslant k$. Em outras palavras, se G é k-conexo em vértices (arestas), então não existe corte de vértices (arestas) de tamanho $< k$.

Sejam $G(V, E)$ um grafo e $E' \subseteq E$ um corte de arestas de G. Então é sempre possível encontrar um corte de vértices V' de tamanho $|V'| \leqslant |E'|$. Basta escolher para V' precisamente o subconjunto de vértices v tais que toda aresta $e \in E'$ é incidente a algum $v \in V'$. Consequentemente, para todo grafo, vale $c_V \leqslant c_E$. Seja w o vértice de grau mínimo no grafo G, suposto não completo. Então é possível desconectar G, removendo-se do grafo as $d(w)$ arestas adjacentes a w. Então $d(w) \geqslant c_E \geqslant c_V$. Observe que se G for o grafo completo K_n, então $d(w) = c_E = c_V = n - 1$.

Os grafos examinados até o momento são também denominados *não direcionados*. Isto porque suas arestas vw não são direcionadas. Assim, se vw é aresta de $G(V, E)$, então tanto v é adjacente a w quanto w é adjacente a v. Em contrapartida, um *grafo direcionado (digrafo)* $D(V, E)$ é um conjunto finito não vazio V (*os vértices*) e um conjunto E (as *arestas*) de pares ordenados de vértices distintos. Portanto, num digrafo, cada aresta ou *arco* (v, w) possui uma única direção de v para w. Diz-se também que (v, w) é *divergente* de v e *convergente* a w. Boa parte da nomenclatura e dos conceitos é análoga nos dois casos. Assim sendo, definem-se, por exemplo, caminho simples, trajeto, ciclo e ciclo simples de forma análoga às definições de grafos. Contudo, ao contrário dos grafos, os digrafos podem possuir ciclos de comprimento 2, no caso em que ambos (v, w) e (w, v) são arestas do digrafo. Os caminhos e ciclos em um digrafo devem obedecer ao direcionamento das arestas.

Seja $D(V, E)$ um digrafo e um vértice $v \in V$. O *grau de entrada* de v, denotado por $d^-(v)$, é o número de arestas convergentes a v. O *grau de saída* de v, denotado por $d^+(v)$, é o número de arestas divergentes de v. Uma *fonte* é um vértice com grau de entrada nulo, enquanto que um *sumidouro* (ou *poço*) é um com grau de saída nulo.

Se forem retiradas as direções das arestas de um digrafo D obtém-se um multigrafo não direcionado, chamado *grafo subjacente* a D.

Um digrafo $D(V, E)$ é *fortemente conexo* quando para todo par de vértices $v, w \in V$ existir um caminho em D de v para w e também de w para v. Se ao menos um desses caminhos existir para todo $v, w \in V$, então D é *unilateralmente conexo*. D é chamado *fracamente conexo* ou *desconexo*, conforme seu grafo subjacente seja conexo ou desconexo, respectivamente. Observe que se um digrafo é fortemente conexo, então também é unilateralmente e fracamente conexo. Se v alcançar todos os vértices de D, então v é chamado *raiz* do digrafo.

Um digrafo é *acíclico* quando não possui ciclos. Observe que o grafo subjacente a um digrafo acíclico não é necessariamente acíclico.

Finalmente, uma *árvore direcionada enraizada* é um digrafo D tal que exatamente um vértice r possui grau de entrada nulo, enquanto que para todos os demais vértices o grau de entrada é igual a um. É imediato observar que D é acíclico com exatamente uma raiz r. Além disso, o grafo subjacente T de D é uma árvore. Por isso, é usual aplicar a nomenclatura de árvores a essa classe de digrafos.

Examinamos, agora, algumas representações de grafos, adequadas ao uso em computador. Ou seja, estruturas que (i) correspondam univocamente a um grafo dado e (ii) possam ser armazenadas e manipuladas sem dificuldade, em um computador. Observe que a representação geométrica, por exemplo, não satisfaz à condição (ii). Dentre os vários tipos de representações adequadas ao computador, ressaltam-se as *representações matriciais* e as *representações por listas*. A seguir, são examinadas algumas dessas. Dado um grafo $G(V, E)$, a *matriz de adjacências* $R = (r_{ij})$ é uma matriz $n \times n$ tal que

$$r_{ij} = \begin{cases} 1, \text{ se } v_i v_j \in E \\ 0, \text{ caso contrário} \end{cases}$$

Ou seja, $r_{ij} = 1$ quando os vértices v_i, v_j forem adjacentes e $r_{ij} = 0$, caso contrário. É imediato verificar que a matriz de adjacências representa um grafo sem ambiguidade. Além disso, é de simples manipulação em computador. Algumas propriedades da matriz R são imediatas. Por exemplo, R é simétrica para um grafo não direcionado. Além disso, o número de 1's é igual a $2m$, pois cada aresta $v_i v_j$ dá origem a dois 1's em R, r_{ij} e r_{ji}.

Observe que uma matriz de adjacências caracteriza univocamente um grafo. Contudo, a um mesmo grafo G podem corresponder várias matrizes diferentes. De fato, se R_1 é matriz de adjacências de G e R_2 é outra matriz obtida por alguma permutação de linhas e colunas de R_1, então R_2 também é matriz de adjacências de G. Ou seja, para construir uma matriz de adjacências de $G(V, E)$, é necessário arbitrar uma certa permutação $v_1, v_2, \ldots, v_n$ para os vértices de V. Naturalmente,

permutações diferentes podem conduzir a matrizes diferentes. A matriz de adjacências pode ser também definida para digrafos. Basta tomar

$$r_{ij} = \begin{cases} 1, \text{ se } (v_i, v_j) \in E \\ 0, \text{ caso contrário} \end{cases}$$

Naturalmente, a matriz não é mais necessariamente simétrica e o número de 1's é exatamente igual a m.

Em relação ao espaço necessário ao armazenamento da matriz, em qualquer caso, existem n^2 informações binárias, o que significa um espaço $O(n^2)$. Ou seja, um espaço não linear com o tamanho do grafo (número de vértices e arestas) no caso em que este for esparso (isto é, $m = O(n)$), principal desvantagem dessa representação.

Uma outra representação matricial para o grafo $G(V, E)$ é a *matriz de incidências* $n \times m$, $B = (b_{ij})$, assim definida:

$$b_{ij} = \begin{cases} 1, \text{ se a aresta } e_j \text{ for incidente ao vértice } v_i \\ 0, \text{ caso contrário} \end{cases}$$

Ou seja, arbitram-se permutações para os vértices $v_1, \ldots, v_n$ e para as arestas $e_1, \ldots, e_m$ de G. Então $b_{ij} = 1$ precisamente quando o vértice v_i for um extremo da aresta e_j, e $b_{ij} = 0$, caso contrário. Novamente, é imediato verificar que a matriz de incidências representa univocamente um grafo, mas esse último pode ser representado, em geral, por várias matrizes de incidências diferentes. Observamos que cada coluna de B tem exatamente dois 1's. A complexidade de espaço da matriz de incidências é $O(nm)$, maior ainda do que a da matriz de adjacências.

Existem várias representações de grafos por listas. sendo muito comum a *estrutura de adjacências*. Seja $G(V, E)$ um grafo. A estrutura de *adjacências*, A de G, é um conjunto de n listas $A(v)$, uma para cada $v \in V$. Cada lista $A(v)$ é denominada *lista de adjacências* do vértice v, e contém os vértices w adjacentes a v em G. Ou seja, $A(v) = \{w \mid vw \in E\}$.

Observamos que cada aresta vw dá origem a duas entradas na estrutura de adjacências, correspondentes a $v \in A(w)$ e $w \in A(v)$. Logo, a estrutura A consiste de n listas com um total de $2m$ elementos. A cada elemento w de uma lista de adjacências $A(v)$ associa-se um ponteiro, o qual informa o próximo elemento, se houver, após w em $A(v)$. Além disso, é necessária também a utilização de um vetor p, de tamanho n, tal que $p(v)$ indique o ponteiro inicial da lista $A(v)$. Assim sendo, se $A(v)$ for vazia, $p(v) = \emptyset$. Pode-se concluir, então, que o espaço utilizado por uma estrutura da adjacências é $O(n + m)$, ou seja, linear com o tamanho

de G. Essa é uma das razões pelas quais a estrutura de adjacências talvez seja a representação mais comum nas implementações dos algoritmos em grafos.

A estrutura de adjacências pode ser interpretada como uma matriz de adjacências representada sob a forma de matriz esparsa, isto é, na qual os zeros não estão presentes. Nesse caso, $v_j \in A(v_i)$ corresponderia a afirmar que $r_{ij} = 1$ na matriz de adjacências. Finalmente observamos que a estrutura de adjacências pode ser definida para digrafos de forma análoga. Isto é, para um digrafo $D(V, E)$, define-se uma lista de adjacências $A(v)$ para cada $v \in V$. A lista $A(v)$ é formada pelos vértices divergentes de v, isto é, $A(v) = \{w \mid (v, w) \in E\}$. A estrutura de adjacências, nesse caso, contém m elementos. É imediato verificar que o espaço requerido pela estrutura A para representar o digrafo D é também linear com o tamanho de D.

1.6 Exercícios

1.1 Responder se Certo ou Errado
Se $\mathcal{P} = \mathcal{NP}$, então qualquer problema $\Pi \in \mathcal{NP}$ admite um algoritmo de complexidade $o(2^k)$ para alguma constante k.

Certo ( ) Errado ( )

1.2 Responder se Certo ou Errado

Se um problema Π possui limite inferior $\Omega(n)$ e admite um algoritmo A de complexidade $O(n)$, então A é um algoritmo ótimo para Π.

Certo ( ) Errado ( )

1.3 Provar ou dar contraexemplo:

As matrizes de adjacências e incidências de um grafo G não podem coincidir.

1.4 A *matriz clique* de um grafo G é uma matriz binária C, assim definida. As linhas correspondem às cliques maximais de G, as colunas correspondem aos vértices, e $c_{ij} = 1$ sse a clique i contém o vértice j, caso contrário $c_{ij} = 0$.

Provar ou dar contraexemplo:
A matriz clique de G é uma representação de G, isto é, identifica univocamente G.

1.5 Responder se Certo ou Errado:

Se um grafo G admitir cortes de vértices e arestas, ambos de tamanho $n-1$, então G é o grafo completo K_n.

Certo ( ) Errado ( )

1.7 Notas Bibliográficas

O livro que é considerado como a referência universal sobre a complexidade de algoritmos e a NP-completude é o de Garey e D. S. Johnson (1979). Mencionamos ainda o excelente texto de Papadimitriou e Steiglitz (1982). O artigo que introduziu a NP-completude e provou a existência do primeiro problema NP-completo é (Cook 1971). No que concerne à teoria de grafos, referenciamos o excelente livro de Bondy e Murty (2008). Em língua portuguesa, Szwarcfiter (2018) descreve algoritmos para problemas envolvendo a teoria de grafos. Mencionamos ainda o livro de Ziviani (1999), com ênfase em implementações.

2 Conceitos Básicos de Probabilidade

2.1 Introdução

Este texto de ciência de dados se inicia com um capítulo introdutório de probabilidade. A teoria de probabilidade é básica para diversas áreas da matemática, física e inclusive áreas humanas. Neste capítulo, abordamos alguns conceitos básicos utilizados ao longo do livro.

Iniciamos com o conceito de espaço amostral, seguido pela definição de probabilidade e algumas de suas propriedades básicas. São descritos, a seguir, os importantes conceitos de probabilidade condicional e eventos independentes. Em seguida, há uma descrição detalhada de variáveis aleatórias. Nesta ocasião, é verificada a linearidade da esperança. São descritas as variáveis aleatórias de Bernoulli, a binomial, a geométrica e a de Poisson. São apresentados os conceitos de variância e desvio padrão.

Ressaltamos que, em capítulos diferentes do livro, são utilizados conceitos gerais de probabilidade. O presente capítulo cobre, com uma pequena folga, os requisitos teóricos necessários para acompanhar o livro. Esse estudo está ao alcance de um aluno de graduação.

2.2 Espaço Amostral

Um *espaço amostral* é um conjunto $\mathcal{A}$ de elementos, chamados de *eventos elementares*. No nosso contexto, o espaço amostral $\mathcal{A}$ está associado a um experimento cujos resultados possíveis correspondem aos seus elementos. Para ilustrar o conceito, considere os experimentos abaixo e seus respectivos resultados.

Exemplo 1. O experimento consiste no lançamento de um dado. Nesse caso, os resultados correspondem às faces de 1 até 6. Portanto, o espaço amostral é

$$\mathcal{A} = \{1, 2, 3, 4, 5, 6\}$$

Exemplo 2. O experimento consiste no lançamento de uma moeda, cujos resultados podem ser cara ou coroa. Representando a cara por C e a coroa por c, o espaço amostral é

$$\mathcal{A} = \{C, c\}$$

Exemplo 3. Se o experimento consiste em lançar uma moeda duas vezes, o espaço amostral é o conjunto

$$\mathcal{A} = \{(C, C), (C, c), (c, C), (cc)\}$$

Exemplo 4. Se o experimento consiste em contar o número de elementos nulos de uma matriz $n \times m$, o espaço amostral é

$$\mathcal{A} = \{x \in \mathbb{N} \mid 0 \leqslant x \leqslant nm\}$$

Exemplo 5. O lançamento de um dado por duas vezes consecutivas corresponde ao espaço amostral, composto por 36 elementos,

$$\mathcal{A} = \{(i, j) \mid 1 \leqslant i, j \leqslant 6\}$$

Exemplo 6. Se o resultado de um experimento corresponde à possível ordem de classificação em um torneio de futebol com 20 clubes, identificados pelo conjunto $\{1, 2, \ldots, 20\}$, o espaço amostral associado possui cardinalidade 20! e corresponde ao conjunto de todas as permutações de $\{1, 2, \ldots, 20\}$.

Um subconjunto $E \subseteq \mathcal{A}$ é denominado *evento*. Assim, no Exemplo 1, o evento $E = \{5\}$ corresponde a um lançamento do dado cujo valor foi 5. No Exemplo 5, o subconjunto $E = \{(1, 5), (2, 6), (5, 1), (6, 2)\}$ está associado ao evento em que os valores dos dados nos dois lançamentos diferem, em módulo, por 4. Ainda no Exemplo 5, o evento em que a diferença entre os valores nos dois lançamentos é igual a 6 corresponde ao evento $E = \emptyset$.

Dados dois eventos $E, F \subseteq \mathcal{A}$, o evento $E \cup F$ é a *união* dos eventos E, F, isto é, a união dos subconjuntos E e F de $\mathcal{A}$. No Exemplo 1, se $E = \{5\}$ e $F = \{3\}$, o evento $E \cup F = \{3, 5\}$ corresponde ao resultado 3 ou 5 no lançamento do dado. Simplesmente, o evento $E \cap F$ é a *interseção* dos eventos E, F. Assim, ainda no Exemplo 1, com $E = \{5\}$ e $F = \{3\}$, $E \cap F = \emptyset$. No Exemplo 3, sejam E o evento em que o resultado do lançamento de duas moedas contenha pelo menos uma cara e F aquele evento cujo resultado contenha pelo menos uma coroa. Assim, $E = \{(C, C), (C, c), (c, C)\}$ e $F = \{(C, c), (c, C), (c, c)\}$. Assim, $E \cap F = \{(C, c), (c, C)\}$ corresponde ao evento em que no lançamento dos dois dados aparecem em ambos uma cara e uma coroa.

As definições de união e interseção de dois eventos se estendem para o caso de mais de dois eventos, de uma maneira similar à união e interseção de vários conjuntos. Se a interseção de dois eventos for vazia, eles são denominados *mutuamente exclusivos*. No caso geral, $n \geqslant 2$ eventos são denominados *mutuamente exclusivos* quando assim o forem dois a dois.

Para um evento $E \subseteq \mathcal{A}$, o *complemento* de E é o evento definido como $\overline{E} = \mathcal{A} \setminus E$, correspondendo ao subconjunto de eventos de $\mathcal{A}$ que são distintos daqueles de E.

2.3 Conceito Geral de Probabilidade

Sejam $\mathcal{A}$ um espaço amostral e $E \subseteq \mathcal{A}$. Ao espaço $\mathcal{A}$, associamos uma *função de probabilidade* P a qual atribui a certo evento $E \subseteq \mathcal{A}$ um valor $P(E)$, satisfazendo os seguintes axiomas:

Axioma 2.1.

$$0 \leqslant P(E) \leqslant 1$$

Axioma 2.2.

$$P(\mathcal{A}) = 1$$

Axioma 2.3. *Se $E_1, E_2, \ldots$ são eventos mutuamente exclusivos (isto é, $E_i \cap E_j = \emptyset$, para todo $i \neq j$), então*

$$P(\bigcup_{i=1,2,\ldots} E_i) = \sum_{i=1,2,\ldots} P(E_i)$$

O Axioma 2.1 afirma que a probabilidade de qualquer evento é um número entre 0 e 1. Pelo Axioma 2.2, concluímos que o resultado de um experimento está necessariamente contido no espaço amostral. O Axioma 2.3 indica que, em uma sequência de eventos mutuamente exclusivos, a probabilidade de um desses eventos ocorrer é igual à soma das probabilidades desses eventos.

Intuitivamente, a probabilidade de um evento pode ser entendida como o valor para o qual converge a taxa de ocorrências desse evento caso o experimento seja repetido um grande número de vezes. Por exemplo, seja E o evento em que o resultado do lançamento de uma moeda seja cara. Então se $P(E) = 1/2$, significa que repetindo-se o experimento um grande número de vezes, a taxa de ocorrência do resultado cara converge para o valor $1/2$.

Como exemplo, seja um dado honesto, isto é, cujas probabilidades de seus resultados são idênticas entre si. Assim, a probabilidade do evento $E = \{5\}$, em particular, é igual a $1/6$. No experimento do lançamento de uma moeda duas vezes consecutivamente, a probabilidade do resultado corresponder a duas caras é $1/4$. Ambos os exemplos anteriores decorrem diretamente da observação do espaço amostral dos respectivos experimentos.

2.4 Propriedades Básicas da Probabilidade

Nesta seção, apresentaremos algumas propriedades fundamentais da probabilidade. Em seguida, apresentaremos alguns exemplos de aplicação. A primeira dessas propriedades relaciona a probabilidade $P(E)$ do evento E com a probabilidade $P(\overline{E})$ do complemento de E.

Teorema 2.4. *Se $E \subseteq \mathcal{A}$ é um evento do espaço amostral $\mathcal{A}$, então*

$$P(\overline{E}) = 1 - P(E)$$

Demonstração. Por definição, $\overline{E} = \mathcal{A} \setminus E$. Isto é, $\overline{E} \cup E = \mathcal{A}$ e, portanto,

$$P(\overline{E} \cup E) = P(\mathcal{A})$$

Como $\overline{E}$, E são mutuamente exclusivos, pelo Axioma 2.3,

$$P(\overline{E} \cup E) = P(\overline{E}) + P(E)$$

Aplicando o Axioma 2.2, obtemos

$$P(\overline{E}) = 1 - P(E)$$

□

O evento vazio ($\emptyset$), naturalmente, é parte do espaço amostral $\mathcal{A}$. Comprovando a intuição, a propriedade seguinte estabelece que o evento vazio possui probabilidade nula.

Teorema 2.5. $P(\emptyset) = 0$

Demonstração. Naturalmente, temos que $\mathcal{A} = \mathcal{A} \cup \emptyset$ e, portanto,

$$P(\mathcal{A}) = P(\mathcal{A} \cup \emptyset)$$

Como $\mathcal{A}$ e $\emptyset$ são mutuamente exclusivos, temos, pelo Axioma 2.3, que

$$P(\mathcal{A}) = P(\mathcal{A}) + P(\emptyset)$$

o que resulta em

$$P(\emptyset) = 0$$

□

A propriedade seguinte, também intuitiva, indica que a probabilidade de um evento não pode ser maior do que a de um outro que o contém.

Teorema 2.6. *Se $E, F \subseteq \mathcal{A}$ e $E \subseteq F$, então $P(E) \leqslant P(F)$.*

Demonstração. Como $E \subseteq F$, vale

$$E \cup (\overline{E} \cap F) = F$$

Mas, E e $\overline{E} \cap F$ são mutuamente exclusivos. Logo, pelo Axioma 2.3,

$$P(E \cup (\overline{E} \cap F)) = P(E) + P(\overline{E} \cap F) = P(F)$$

Como $P(\overline{E} \cap F) \geqslant 0$ pelo Axioma 2.2, temos que

$$P(E) \leqslant P(F)$$

□

A próxima propriedade diz respeito a espaços amostrais que podem ser particionados em um número finito de eventos de probabilidades idênticas.

Proposição 2.7. *Sejam $\mathcal{A}$ um espaço amostral e $E_1, \ldots, E_n$, eventos de probabilidades idênticas, um particionamento de $\mathcal{A}$. Seja E um evento formado pela união de t destes eventos. Então,*

$$P(E) = \frac{t}{n}$$

Demonstração. Os eventos E_i possuem a mesma probabilidade, isto é,

$$P(E_1) = \ldots = P(E_n) = p$$

Pelo Axioma 2.3,

$$P(E_1 \cup \cdots \cup E_n) = P(E_1) + \cdots + P(E_n) = np$$

Mas $P(E_1 \cup \cdots \cup E_n) = P(\mathcal{A}) = 1$ pelo Axioma 2.1. Logo, obtemos

$$p = \frac{1}{n}$$

A probabilidade do evento E, formado pela união de t eventos E_i, cada um com probabilidade $p = 1/n$, é dada, aplicando-se o Axioma 2.3, por

$$P(E) = P(\bigcup_{E_i \subset E} E_i) = \sum_{E_i \subset E} P(E_i) = \frac{t}{n}$$

□

Proposição 2.8. *Se $E_1, E_2 \subseteq \mathcal{A}$ são dois eventos do espaço amostral $\mathcal{A}$, então*

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$$

Demonstração. Para qualquer par de eventos $E_1, E_2 \subseteq \mathcal{A}$, vale $E_1 \cup E_2 = E_1 \cup (\overline{E_1} \cap E_2)$. Logo,

$$P(E_1 \cup E_2) = P(E_1 \cup (\overline{E_1} \cap E_2)) \tag{2.1}$$

Como E_1, $\overline{E_1} \cap E_2$ são mutuamente exclusivos, aplicando o Axioma 2.3,

$$P(E_1 \cup (\overline{E_1} \cap E_2)) = P(E_1) + P(\overline{E_1} \cap E_2) \tag{2.2}$$

De (2.2) em (2.1), obtemos

$$P(E_1 \cup E_2) = P(E_1) + P(\overline{E_1} \cap E_2) \tag{2.3}$$

Mas $E_2 = (E_1 \cap E_2) \cup (\overline{E_1} \cap E_2)$. Podemos novamente aplicar o Axioma 2.3 e obtemos

$$P(E_2) = P(E_1 \cap E_2) + P(\overline{E_1} \cap E_2) \tag{2.4}$$

Aplicando (2.4) em (2.3), concluímos que

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$$

□

Intuitivamente, a proposição acima se justifica pelo fato de que, ao somar as probabilidades $P(E_1) + P(E_2)$, cada elemento de $E_1 \cup E_2$ é considerado duas vezes.

A Proposição 2.9 é uma generalização da Proposição 2.8, ao considerar a probabilidade de $n \geqslant 2$ eventos.

Proposição 2.9 (Príncipio da Inclusão e Exclusão)**.** *Se $E_1, \ldots, E_n \subseteq \mathcal{A}$ são eventos do espaço amostral $\mathcal{A}$, então*

$$\begin{aligned} P\Big(\bigcup_{1\leqslant i\leqslant n} E_i\Big) = &\sum_{1\leqslant i\leqslant n} P(E_i) - \sum_{1\leqslant i_1<i_2\leqslant n} P(E_{i_1} \cap E_{i_2}) \\ &+ \sum_{1\leqslant i_1<i_2<i_3\leqslant n} P(E_{i_1} \cap E_{i_2} \cap E_{i_3}) + \cdots \\ &+ (-1)^{(k+1)} \sum_{1\leqslant i_1<\cdots<i_k\leqslant n} P(E_{i_1} \cap \cdots \cap E_{i_k}) + \cdots \\ &+ (-1)^{(n+1)} P(E_1 \cap \cdots \cap E_n) \end{aligned}$$

Demonstração. Por indução em n, aplicando a Proposição 2.8. □

Em seguida, apresentaremos exemplos de cálculo de probabilidade.

Exemplo 7. Duas pessoas participam de um sorteio para dois prêmios. A probabilidade da primeira pessoa ser sorteada é p_1, enquanto que a probabilidade da segunda pessoa ser sorteada é p_2. Por outro lado, a probabilidade de que ambas sejam sorteadas é p_3. Qual a probabilidade de que nenhuma das duas sejam sorteadas?

Para responder a esta questão, utilizamos a Proposição 2.8. Sejam E_1 o evento em que a primeira pessoa é sorteada e E_2 aquele em que a segunda pessoa é sorteada. A probabilidade de que pelo menos uma das pessoas seja sorteada é

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2) = p_1 + p_2 - p_3$$

Por outro lado, o evento em que ambas as pessoas não são sorteadas é complemento ao evento em que pelo menos uma é sorteada. Logo, a resposta é $1 - p_1 - p_2 + p_3$.

Exemplo 8. (Problema do Aniversário) Em um grupo de n pessoas, qual a probabilidade de que duas pessoas façam aniversário na mesma data? Para a solução, supomos que o ano tenha 365 dias exatos e que cada pessoa tenha a mesma probabilidade de ter nascido em qualquer dia do ano. O espaço amostral $\mathcal{A}$ consiste de tuplas de n datas de nascimento e, portanto, possui tamanho 365^n. O evento

$$D = \{(d_1, \ldots, d_n) \subseteq \mathcal{A} : d_1, \ldots, d_n \text{ são distintos}\}$$

é aquele em que todas as pessoas nasceram em um dia distinto. Para que as datas de aniversário fossem todas distintas, a primeira pessoa teria 365 possibilidades, a segunda teria 364, e assim por diante, até a n-ésima pessoa que teria $365 - n + 1$. Isto é, o total de possibilidades correspondente ao arranjo é

$$365 \times 364 \times \cdots \times (365 - n + 1)$$

e, portanto,

$$P(D) = \frac{365 \times 364 \times \cdots \times (365 - n + 1)}{365^n}$$

Como a probabilidade do evento em que duas pessoas façam aniversário na mesma data é complemento de D, a resposta é

$$1 - \frac{365 \times 364 \times \cdots \times (365 - n + 1)}{365^n}$$

Pode parecer de certa forma surpreendente que, para valores relativamente baixos de n, esta probabilidade é alta. Para $n \geqslant 23$, esta probabilidade já é superior a 50%. Por este motivo, este problema é também conhecido como o *Paradoxo do Aniversário*.

2.5 Probabilidade Condicional

Sejam E, F dois eventos de um espaço amostral $\mathcal{A}$. A *probabilidade de E condicional a F* é a probabilidade de ocorrência de E restringindo o espaço amostral aos elementos que satisfazem a F. Isto é, com o conhecimento prévio de ocorrência de F, a probabilidade de E é relativa somente aos casos em que F ocorre. A probabilidade de E condicional a F é representada por $P(E \mid F)$.

Para ilustrar este conceito, seja o experimento em que uma moeda foi lançada 3 vezes consecutivamente. Seja E o evento em que o resultado desses lançamentos foi (C, C, C). Isto é, 3 caras consecutivas. Similarmente, seja F o evento em que o resultado dos dois primeiros lançamentos foi (C, C). Qual é a probabilidade de ocorrência de E, com o conhecimento prévio de que F ocorreu, isto é, qual o valor de $P(E \mid F)$?

Observamos que, dada a ocorrência de F, o espaço amostral para E se reduz para $\{(C, C, C), (C, C, c)\}$. O resultado em que E ocorre é (C, C, C) e, portanto, $P(E \mid F) = 1/2$. Por outro lado, uma vez que F ocorreu, a probabilidade dos demais resultados não iniciados por duas caras torna-se igual a 0. Observamos ainda que a probabilidade de E, sem o conhecimento prévio de F, é igual a $1/8$, isto é, $P(E) = 1/8$.

Sejam E, F eventos de um espaço amostral $\mathcal{A}$, $P(F) \neq 0$. A probabilidade condicional $P(E \mid F)$ de ocorrência do evento E, relativo aquela de F, é dada por

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)}$$

No exemplo anterior, em que E, F são os eventos em que os resultados dos lançamentos de uma moeda por 3 vezes consecutivas são respectivamente 3 caras e iniciados por 2 caras, observamos que $E \subseteq F$. Assim, $E \cap F = E$. Por outro lado, $P(E) = 1/8$ e $P(F) = 2/8 = 1/4$. Logo,

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)} = \frac{1}{2}$$

A expressão de probabilidade condicional apresentada está restrita aos casos em que $P(F) \neq 0$. Quando $P(F) = 0$, a ocorrência do evento E independe da ocorrência de F. Assim,

$$P(E \mid F) = P(E), \text{para } P(F) = 0$$

.

A Proposição 2.9 fornece uma expressão para o cálculo da probabilidade de uma união de eventos. A Proposição 2.10 descreve a expressão da probabilidade de interseção de eventos.

Proposição 2.10. *Se $E_1, \ldots, E_n$ são eventos de um espaço amostral $\mathcal{A}$, então*

$$\begin{aligned} P(E_1 \cap \cdots \cap E_n) = & P(E_1)P(E_2 \mid E_1)P(E_3 \mid E_2 \cap E_1)\cdots \\ & P(E_n \mid E_{n-1} \cap \cdots \cap E_1) \end{aligned}$$

Demonstração. Para mostrar a validade da expressão anterior, basta substituir os valores das probabilidades condicionais $P(E_2 \mid E_1)$, $P(E_3 \mid E_2 \cap E_1)$, ..., $P(E_n \mid E_{n-1} \cap \cdots \cap E_1)$ que aparecem do lado direito da expressão anterior pelos valores das probabilidades condicionais. O valor assim obtido corresponderá ao lado esquerdo da expressão, isto é, ao valor de $P(E_1 \cap \cdots \cap E_n)$. □

Teorema 2.11 (Teorema da Probabilidade Total)**.** *Sejam $E_1, \ldots, E_n$ eventos de um espaço amostral $\mathcal{A}$, os quais constituem uma partição de $\mathcal{A}$. Seja F um evento qualquer de $\mathcal{A}$. Então,*

$$P(F) = \sum_{i=1}^{n} P(E_i)P(F \mid E_i)$$

Demonstração. Para cada evento E_i, considerando o par de eventos F, E_i e o evento $F \cap E_i$, segue que $P(E_i)P(F \mid E_i) = P(F \cap E_i)$. Como os eventos $E_1, \ldots, E_n$ formam uma partição de $\mathcal{A}$, o que implica que E_i e E_j são disjuntos para $i \neq j$ e, portanto, $F \cap E_i$ e $F \cap E_j$ são também disjuntos. Portanto, $\sum_{i=1}^{n} P(E_i)P(F \mid E_i) = \sum_{i=1}^{n} P(F \cap E_i) = P(\cup_{i=1}^{n}(F \cap E_i)) = P(F \cap (\cup_{i=1}^{n} E_i)) = P(F \cap \mathcal{A}) = P(F)$. □

O teorema acima permite decompor a probabilidade de um evento qualquer E em relação a um conjunto de eventos $\{E_1, \ldots, E_n\}$, que constitui uma partição de $\mathcal{A}$, de tal modo que $P(F)$ seja a soma de cada probabilidade $P(E_i)$ ponderada pela probabilidade condicional $P(F \mid E_i)$.

Corolário 2.12 (Lei de Bayes). *Sejam $E_1, \ldots, E_n$ eventos do espaço amostral $\mathcal{A}$, os quais constituem uma partição de $\mathcal{A}$. Seja $F \subseteq \mathcal{A}$ um evento qualquer. Sendo $P(E_i), P(F) > 0$, para todo $i = 1, \ldots, n$,*

$$P(E_i \mid F) = \frac{P(F \mid E_i)P(E_i)}{\sum_{i=1}^{n} P(F \mid E_i)P(E_i)}$$

Demonstração. Considerando o numerador e o denominador da expressão acima, temos que $P(F \mid E_i)P(E_i) = P(F \cap E_i)$ e, pelo Teorema 2.11,

$$\sum_{i=1}^{n} P(F \mid E_i)P(E_i) = P(F)$$

Logo, $P(F \cap E_i)/P(F) = P(E_i \mid F)$, o que demonstra o Corolário. □

Como exemplo de aplicação das expressões de probabilidade total e da Lei de Bayes, considere o seguinte exemplo. Uma indústria automobilística produz modelos de 3 tipos distintos, I, II e III, de automóveis. O modelo I corresponde a 20% de produção da indústria, o modelo II a 50% e o modelo III a 30%. Foi observada um falha na suspensão dos automóveis produzidos, que corresponde a 2% de produção dos modelos I, 3% dos modelos II e 1% dos modelos III.

(a) Qual é a probabilidade de algum veículo produzido pela indústria apresente falha na suspensão?

Seja o espaço amostral correspondendo à produção dos veículos. Represente por E_1, E_2, E_3 os eventos de que um veículo produzido seja respectivamente dos tipos I, II, III. Além disso, seja F o evento de que um veículo apresente falha na suspensão. As seguintes probabilidades são dados do problema:

$$P(E_1) = 0{,}2;\ P(E_2) = 0{,}5;\ P(E_3) = 0{,}3$$

$$P(F \mid E_1) = 0,02\ ;\ P(F \mid E_2) = 0,03\ ;\ P(F \mid E_3) = 0,01$$

A resposta ao item (a) é a probabilidade $P(F)$. Pelo Teorema 2.11,

$$\begin{aligned} P(F) &= \sum_{i=1}^{3} P(E_i)P(F \mid E_i) \\ &= 0,2 \times 0,02 + 0,5 \times 0,03 + 0,3 \times 0,01 \\ &= 0,022 \end{aligned}$$

(b) Entre todos os veículos produzidos pela indústria que apresentam falha na suspensão, qual a probabilidade de que seja do modelo I?

A resposta do item (b) é a probabilidade $P(E_1 \mid F)$. Aplicando a Lei de Bayes,

$$\begin{aligned} P(E_1 \mid F) &= \frac{P(F \mid E_1)P(E_1)}{P(F)} \\ &= \frac{0{,}02 \times 0{,}2}{0{,}022} \\ &\approx 0{,}182 \end{aligned}$$

2.6 Eventos Independentes

Considere os eventos E, F de um espaço amostral $\mathcal{A}$. A expressão para a probabilidade de E condicionada à ocorrência de F, é dada por $P(E \mid F) = P(E \cap F)/P(F)$, com $P(F) \neq 0$. Nesta seção, tratamos da situação em que a probabilidade de ocorrência de E independe do evento F. Nesses casos, para $E, F \in \mathcal{A}$, vale a expressão

$$P(E \mid F) = P(E)$$

O evento E é dito *independente* de F.

Invertendo os papéis de E e F na expressão da probabilidade condicional e nas equações anteriores, concluímos que E independente de F implica F independente de E. Então dizemos simplesmente que E e F são *independentes* entre si. Nesse caso, obtemos que

$$P(E \cap F) = P(E)P(F)$$

Essa expressão pode ser generalizada para o caso de n eventos, dois a dois independentes. Ou seja, se $E_1, \ldots, E_n$ são n eventos independentes dois a dois, então

$$P(E_1 \cap \ldots \cap E_n) = P(E_1) \ldots P(E_n)$$

De um modo geral, para verificar se dois eventos dados, E, F são independentes, pode-se aplicar a expressão anterior. Isto é, determinar o evento $E \cap F$ e verificar se $P(E \cap F) = P(E)P(F)$.

Por exemplo, seja o experimento do lançamento de uma moeda por duas vezes consecutivas. Seja E o evento em que ambos os lançamentos produziram o

mesmo resultado, isto é, ambos caras ou ambos coroas. Seja F o experimento em que o resultado do segundo lançamento é coroa. O espaço amostral é, portanto, $\Omega = \{(C,C),(C,c),(c,C),(c,c)\}$. Assim, $E = \{(C,C),(c,c)\}$, $F = \{(C,c),(c,c)\}$ e $E \cap F = \{(c,c)\}$,

$$P(E) = 1/2, \quad P(F) = 1/2 \text{ e } P(E \cap F) = 1/4 = P(E)P(F)$$

Logo, E, F são independentes.

2.7 Variáveis Aleatórias

Seja $\mathcal{A}$ o espaço amostral relativo a um certo experimento. Uma *variável aleatória* é uma função do espaço amostral no conjunto dos reais $\mathbb{R}$. De um modo geral, uma variável aleatória X será utilizada para refletir um resultado desejado x, e denotamos essa possibilidade por $P[X = x]$.

O conceito de independência de eventos se estende para variáveis aleatórias. Assim, duas variáveis aleatórias X, Y são *independentes* quando

$$P[X = x, Y = y] = P[X = x]P[Y = y]$$

Nesse caso, os resultados refletidos pela variável X independem daqueles obtidos por Y, e vice-versa.

Uma *variável aleatória discreta* é aquela que pode assumir um conjunto enumerável de valores possíveis. Caso contrário, a variável aleatória é dita *contínua*. Se X é uma variável aleatória discreta, definimos a *função discreta de probabilidade* $p(x)$ de X, ou *densidade*, como

$$p(x) = P[X = x]$$

isto é, $p(x)$ é igual à probabilidade (discreta) de que X seja o resultado x.

Como primeiro exemplo, seja o caso do lançamento de um dado. Denote por X a variável aleatória que reflete os possíveis resultados obtidos no lançamento. Os valores possíveis de x da variável aleatória X são, portanto, iguais a $1, 2, 3, 4, 5, 6$.

Como exemplo adicional, lançamos um dado por duas vezes consecutivas, e estamos interessados nas diferenças dos lançamentos. Seja X a variável aleatória que reflete essas diferenças, e x os valores resultados possíveis obtidos. Os valores possíveis de x são, portanto, iguais a $0, 1, -1, 2, -2, 3, -3, 4, -4, 5, -5$, e as

probabilidades correspondentes são iguais a:

$$
\begin{aligned}
p(0) &= P[X = 0] = 6/36 = 1/6 \\
p(1) &= P[X = 1] = p(-1) = P[X = -1] = 5/36 \\
p(2) &= P[X = 2] = p(-2) = P[X = -2] = 4/36 = 1/9 \\
p(3) &= P[X = 3] = p(-3) = P[X = -3] = 3/36 = 1/12 \\
p(4) &= P[X = 4] = p(-4) = P[X = -4] = 2/36 = 1/18 \\
p(5) &= P[X = 5] = p(-5) = P[X = -5] = 1/36
\end{aligned}
$$

Em seguida, descrevemos um conceito largamente utilizado na teoria das probabilidades. Sejam X uma variável aleatória e $p(x)$ sua função de probabilidade. A *esperança* ou *valor esperado* de X, denotado por $E[X]$, é definida como

$$E[X] = \sum_x x p(x)$$

se X é discreta, ou

$$E[X] = \int_{-\infty}^{+\infty} x f(x) dx$$

caso contrário, onde $f(x)$ é chamada de *função de densidade*, definida como a função tal que

$$P(X \leqslant x) = \int_{-\infty}^{x} f(t) dt$$

ou, equivalentemente,

$$f(x) = \frac{d}{dx} P(X \leqslant x)$$

No exemplo anterior, em que a variável aleatória X representa as diferenças dos dois lançamentos consecutivos de um dado, os valores possíveis para x, com suas probabilidades, são

$$
\begin{aligned}
p(0) &= 1/6 \\
p(1) &= p(-1) = 5/36 \\
p(2) &= p(-2) = 1/9 \\
p(3) &= p(-3) = 1/12 \\
p(4) &= p(-4) = 1/18 \\
p(5) &= p(-5) = 1/36
\end{aligned}
$$

Para o cálculo da esperança $E[X]$, consideramos que $p(x) = p(-x)$ para $x \neq 0$. Concluimos que cada par de valores $xp(x)$ e $-xp(-x)$ se anularão dois a dois, para $x \neq 0$. Assim,

$$E[X] = 0 \times p(0) = 0 \times 1/6 = 0$$

Por diversas vezes, o interesse é calcular a esperança de uma função da variável aleatória X, ao invés da esperança de X, propriamente dita. Assim, considere novamente o exemplo anterior, em que X representa as diferenças de dois lançamentos consecutivos de um dado. Seja determinar a esperança da função X^2. Definimos uma variável aleatória $Y = X^2$, e a nossa questão agora consiste em determinar a esperança da variável Y. Denotemos por y os possíveis resultados obtidos para Y. Os valores que y pode assumir são $0, 1, 4, 9, 16, 25$. Os valores das possibilidades correspondentes são

$$\begin{aligned}
p(0) &= P[Y = 0] = P[X = 0] = 6/36 = 1/6 \\
p(1) &= P[Y = 1] = P[X = 1] + P[X = -1] = 5/36 + 5/36 = 5/18 \\
p(4) &= P[Y = 4] = P[X = 2] + P[X = -2] = 1/9 + 1/9 = 2/9 \\
p(9) &= P[Y = 9] = P[X = 3] + P[X = -3] = 1/12 + 1/12 = 1/6 \\
p(16) &= P[Y = 16] = P[X = 4] + P[X = -4] = 1/18 + 1/18 = 1/9 \\
p(25) &= P[Y = 25] = P[X = 5] + P[X = -5] = 1/36 + 1/36 = 1/18
\end{aligned}$$

A esperança de $Y = X^2$ é

$$E[Y] = 0 \times 1/6 + 1 \times 5/18 + 4 \times 2/9 + 9 \times 1/6 + 16 \times 1/9 + 25 \times 1/18 = 35/6$$

O valor esperado de uma variável aleatória, de certa forma, representa uma média ponderada entre os valores que essa variável pode assumir. Frequentemente, é importante o conhecimento de como esses valores estão distribuídos em relação à média. Por exemplo, se estão concentrados em torno da média, ou estão dispersos, afastando-se da média. Para quantificar esses valores de dispersão, são usadas as medidas de variância e desvio padrão.

Seja X uma variável aleatória, com valor esperado $E[X]$. A *variância* de X, denotada por $\mathrm{Var}(X)$, é definida como

$$\mathrm{Var}(X) = E[(X - E[X])^2]$$

Para determinar a variância de uma variável aleatória, frequentemente é utilizada a expressão seguinte:

Proposição 2.13. *Seja X uma variável aleatória, com valor esperado $E[X]$. Então sua variância é*

$$\text{Var}(X) = E[X^2] - (E[X])^2$$

Demonstração. Seja x um resultado do experimento aleatório relativo a X, e $p(x)$ a sua probabilidade de ocorrência. Sabemos que $E[X] = \sum_x xp(x)$. Então

$$\begin{aligned}
\text{Var(X)} &= E[(X - E[X])^2] = \\
&= \sum_x (x - E[X])^2 p(x) \\
&= \sum_x x^2 p(x) - 2E[X] \sum_x xp(x) + E[X]^2 \sum_x p(x) \\
&= E[X^2] - 2E[X]E[X] + (E[X]^2) \times 1 \\
&= E[X^2] - (E[X])^2
\end{aligned}$$

□

Como exemplo, seja calcular a variância do lançamento de dois dados, consecutivamente, cuja variável aleatória X representa a diferença dos valores entre o primeiro e o segundo lançamentos. Vimos, anteriormente, que $E[X] = 0$ e $E[X^2] = 35/6$. Portanto, a variância de X pode ser calculada por

$$\text{Var(X)} = E[X^2] - (E[X])^2 = 35/6 - (0)^2 = 35/6$$

Um outro critério para avaliar a dispersão dos valores de uma variável aleatória é o *desvio padrão*, representado por $\sigma(X)$,

$$\sigma(X) = \sqrt{\text{Var(X)}}$$

O desvio padrão se presta para o afastamento dos dados individuais, em relação ao valor esperado. Assim, no exemplo em que X representa a diferença entre os valores de dois lançamentos consecutivos dos dados, o desvio padrão é

$$\sigma(X) = \sqrt{\text{Var(X)}} = \sqrt{35/6} \approx 2,415$$

Assim, os valores individuais, em média, estão afastados de $2,415$ do valor esperado.

2.7.1 Linearidade da Esperança e da Variância

Como vimos, uma variável aleatória pode ser definida em função de outras. Um caso particular bastante comum é uma variável aleatória ser definida como a combinação linear de outras variáveis. Neste caso, é possível determinar o valor esperado e a variância dessa variável em função dos valores esperados e variâncias das variáveis aleatórias das quais ela depende.

Consideramos um espaço amostral $\mathcal{A}$ enumerável, correspondendo a um experimento e $s \in \mathcal{A}$ um resultado deste experimento. Sejam X uma variável aleatória correspondente e $X(s)$ o seu valor para s. Seja $p(s)$ a probabilidade de que o resultado do experimento seja igual a s. Primeiramente, mostraremos que o valor esperado de X é equivalente à média de $X(s)$ ponderada pela probabilidade $p(s)$ para todo $s \in \mathcal{A}$.

Proposição 2.14. *Seja X uma variável aleatória correspondente ao espaço amostral $\mathcal{A}$. Temos que*

$$E[X] = \sum_{s \in \mathcal{A}} X(s)p(s)$$

Demonstração. Seja $F_k \subseteq \mathcal{A}$ o evento associado a todos os eventos elementares s tais que $X(s) = k$, isto é, $F_k = \{s \in \mathcal{A} \mid X(s) = k\}$. Assim, note que as probabilidades $P(X = k)$ e $P(F_k)$ são a mesma. Portanto, temos que

$$\begin{aligned}
E[X] &= \sum_k kP(X = k) \\
&= \sum_k kP(F_k) \\
&= \sum_k k \sum_{s \in F_k} p(s) \\
&= \sum_k \sum_{s \in F_k} kp(s) \\
&= \sum_k \sum_{s \in F_k} X(s)p(s) \\
&= \sum_{s \in \mathcal{A}} X(s)p(s)
\end{aligned}$$

□

Corolário 2.15. *Sejam $X_1, \ldots, X_n$ variáveis aleatórias quaisquer correspondentes ao espaço amostral $\mathcal{A}$. Para quaisquer constantes $a_1, a_2, \ldots, a_n$, temos que*

$$E\left[\sum_{i=1}^{n} a_i X_i\right] = \sum_{i=1}^{n} a_i E[X_i]$$

Demonstração. Seja Z a variável aleatória definida por

$$Z = \sum_{i=1}^{n} a_i X_i$$

De acordo com a Proposição 2.14, temos que

$$\begin{aligned}
E[Z] &= \sum_{s\in\mathcal{A}} Z(s)p(s) \\
&= \sum_{s\in\mathcal{A}} \left(\sum_{i=1}^{n} a_i X_i(s)\right) p(s) \\
&= \sum_{s\in\mathcal{A}} \sum_{i=1}^{n} a_i X_i(s)p(s) \\
&= \sum_{i=1}^{n} \sum_{s\in\mathcal{A}} a_i X_i(s)p(s) \\
&= \sum_{i=1}^{n} a_i \sum_{s\in\mathcal{A}} X_i(s)p(s) \\
&= \sum_{i=1}^{n} a_i E[X_i]
\end{aligned}$$

□

Como exemplo, seja determinar o valor esperado da variável aleatória X que representa soma de faces quando um dado é lançado duas vezes. Seja X_i a face obtida no i-ésimo lançamento, $i = 1, 2$. Logo, $X = X_1 + X_2$. Como

$$E[X_i] = \sum_{k=1}^{6} k \cdot \frac{1}{6} = \frac{7}{2}$$

temos que, aplicando-se a linearidade da esperança,

$$E[X] = E[X_1] + E[X_2] = \frac{7}{2} + \frac{7}{2} = 7$$

A variância não apresenta, em geral, linearidade como a esperança. No entanto, para o caso de soma de variáveis aleatórias independentes, pode-se estabelecer a variância da seguinte forma.

Proposição 2.16. *Sejam $X_1, X_2, \ldots, X_n$ variáveis aleatórias independentes correspondentes ao espaço amostral $\mathcal{A}$. Para quaisquer constantes $a_1, a_2, \ldots, a_n$, temos que*

$$\text{Var}\left(\sum_{i=1}^{n} a_i X_i\right) = \sum_{i=1}^{n} a_i^2 \text{Var}(X_i)$$

Portanto, a linearidade da variância ocorre quando $X_1, \ldots, X_n$ são variáveis aleatórias independentes e $a_1 = a_2 = \cdots = a_n = 1$.

Em seguida, descreveremos algumas distribuições de variáveis aleatórias discretas comumente utilizadas.

2.7.2 Variável Aleatória de Bernoulli

A *variável aleatória de Bernoulli X com parâmetro p* pode assumir apenas dois valores, $X = 0$ ou $X = 1$. Geralmente, o experimento cuja variável aleatória assume o valor 1 é dito *sucesso*, enquanto que o de valor 0 é o *fracasso*. Assim, a função de probabilidade $p(x)$ relativa à variável aleatória X de Bernoulli pode ser escrita como

$$p(0) = 1 - p$$
$$p(1) = p$$

onde $0 \leqslant p \leqslant 1$ é a probabilidade de que o resultado do experimento tenha sido sucesso.

Por exemplo, no lançamento de um dado por duas vezes consecutivas, se sucesso corresponde a obter a soma 12 dos dois valores dos lançamentos, a probabilidade de sucesso é, portanto, 1/36, e a probabilidade associada à variável de Bernoulli é

$$p(0) = 35/36$$
$$p(1) = 1/36$$

A esperança e a variância de uma variável aleatória de Bernoulli são dadas a seguir.

Proposição 2.17. *Seja X uma variável aleatória de Bernoulli com parâmetro p. O valor esperado e a variância de X são obtidos por*

$$E[X] = p$$
$$\text{Var}(X) = p(1-p)$$

Demonstração. Temos que a esperança é obtida por

$$\begin{aligned} E[X] &= \sum_{k=0,1} kp(k) = 0 \cdot p(0) + 1 \cdot p(1) \\ &= p \end{aligned}$$

Como

$$\begin{aligned} E[X^2] &= \sum_{k=0,1} k^2 p(k) = 0 \cdot p(0) + 1 \cdot p(1) \\ &= p \end{aligned}$$

temos que sua variância é dada por

$$\begin{aligned} \text{Var}(X) &= E[X^2] - E^2[X] = p - p^2 \\ &= p(1-p) \end{aligned}$$

□

2.7.3 Variável Aleatória Binomial

A variável aleatória binomial X é uma generalização da variável de Bernoulli, onde, ao invés de considerar o sucesso ou o fracasso de um experimento aleatório, consideramos n experimentos. Nesse caso, são realizados n experimentos independentes, cada qual com a mesma probabilidade de sucesso p, e probabilidade de fracasso $1-p$. O objetivo é determinar a quantidade total de sucessos obtidos nesses n experimentos.

Assim, são dados um inteiro $n \geqslant 1$ e um valor p, $0 \leqslant p \leqslant 1$. São realizados n experimentos, cada um com probabilidade de sucesso p e fracasso $1-p$. A *variável aleatória binomial X com parâmetros* (n, p) representa a quantidade total de sucessos após a realização de todos os experimentos. Assim, ele se constitui em uma generalização da variável aleatória de Bernoulli. Essa última, um caso especial de variável binomial, cujos parâmetros são $(1, p)$.

A proposição seguinte determina a função de probabilidade da variável aleatória binomial.

Proposição 2.18. *Seja X uma variável aleatória binomial de parâmetros (n, p). A função de probabilidade de X, associada à quantidade de sucessos do experimento aleatório correspondente a X é*

$$p(i) = \binom{n}{i} p^i (1-p)^{n-i}$$

Demonstração. Suponha que os experimentos aleatórios resultaram em i sucessos e $n - i$ fracassos. Como os experimentos são independentes, a probabilidade de uma sequência de n resultados apresentar i sucessos é $p^i(1-p)^{n-i}$. Como são n experimentos, dos quais i resultaram em sucesso, há um total de $\binom{n}{i}$ diferentes sequências de resultados sucessos-fracassos, que conduzem aos i sucessos. Portanto,

$$p(i) = \binom{n}{i} p^i (1-p)^{n-i}$$

□

A proposição seguinte determina o valor esperado $E[X]$ e a variância de uma variável aleatória binomial X.

Proposição 2.19. *Seja X uma variável aleatória binomial de parâmetros (n, p). O valor esperado e a variância de X são obtidos por*

$$E[X] = np$$

$$\text{Var}(X) = np(1-p)$$

Demonstração. Seja X_i a variável aleatória de Bernoulli que representa o sucesso do i-ésimo experimento. Portanto,

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

Note que as variáveis X_i, X_j, com $i \neq j$, são independentes. Assim, usando a linearidade da esperança e da variância, temos que

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = np$$

$$\text{Var}(X) = \text{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \text{Var}(X_i) = np(1-p)$$

□

2.7.4 Variável Aleatória Geométrica

A variável aleatória geométrica é uma variação da binomial. Na variável de Bernoulli, há um experimento aleatório com probabilidade de sucesso p, e a variável representa a ocorrência de sucesso. Na variável binomial, há n experimentos independentes, cada qual com probabilidade de sucesso igual a p, e a variável representa o número total de sucessos ocorridos após a realização de n experimentos. Para a variável aleatória geométrica X, são realizados experimentos aleatórios independentes, cada qual com probabilidade de sucesso p, repetidamente, até que ocorra um sucesso. O objetivo é determinar o número de experimentos que resultam em fracasso até a obtenção do primeiro sucesso. Nesse caso, X representa o número total de fracassos.

A função de probabilidade da variável aleatória geométrica é dada pela proposição seguinte.

Proposição 2.20. *São realizados experimentos independentes aleatórios, consecutivamente, cada qual com probabilidade de sucesso p, onde $0 \leqslant p \leqslant 1$, até a ocorrência do primeiro sucesso, que se dá no n-ésimo experimento. Seja X a variável aleatória geométrica correspondente a esses experimentos. Então a função de probabilidade associada é*

$$p(n) = (1-p)^{n-1} p$$

Demonstração. Sabemos que houve n experimentos independentes, $n-1$ dos quais com probabilidade $(1-p)$, correspondentes aos fracassos, e o n-ésimo experimento de sucesso, com probabilidade p. Então

$$p(n) = (1-p)^{n-1} p$$

□

Proposição 2.21. *Seja X uma variável geométrica de parâmetro p. O valor esperado e a variância de X são obtidos por*

$$E[X] = \frac{1}{p}$$

$$\text{Var}(X) = \frac{1-p}{p^2}$$

Demonstração. Pela definição de valor esperado, temos que

$$\begin{aligned} E[X] &= \sum_{k=1,2,\ldots} kp(k) \\ &= \sum_{k=1,2,\ldots} k(1-p)^{1-k}p \\ &= p\sum_{k=1,2,\ldots} k(1-p)^{1-k} \end{aligned}$$

Por outro lado, temos que

$$\begin{aligned} \sum_{k=1,2,\ldots} kq^{1-k} &= \sum_{k=1,2,\ldots} \frac{d}{dq}q^k \\ &= \frac{d}{dq}\sum_{k=1,2,\ldots} q^k \\ &= \frac{d}{dq}\frac{q}{1-q} \\ &= \frac{1}{(1-q)^2} \end{aligned}$$

Assim, no lugar de $q = 1-p$, temos que

$$\begin{aligned} E[X] &= p\sum_{k=1,2,\ldots} k(1-p)^{1-k} \\ &= p\frac{1}{p^2} = \frac{1}{p} \end{aligned}$$

O cálculo da variância pode ser feito da seguinte maneira. Primeiro, notamos que como

$$\frac{d}{dq}(k-1)q^k = k(k-1)q^{k-1}$$

temos que

$$\begin{aligned}\sum_{k=1,2,\ldots} k(k-1)q^{k-1} &= \sum_{k=1,2,\ldots} \frac{d}{dq}(k-1)q^k \\ &= \frac{d}{dq}\sum_{k=1,2,\ldots}(k-1)q^k \\ &= \frac{d}{dq}\sum_{k=1,2,\ldots} kq^{k+1} \\ &= \frac{d}{dq}q^2\sum_{k=1,2,\ldots} kq^{k-1} \\ &= \frac{d}{dq}\frac{q^2}{(1-q)^2} = \frac{2q}{(1-q)^3}\end{aligned}$$

Usando-se a expressão anterior e fazendo-se $q = 1 - p$, temos que

$$\begin{aligned}E[X(X-1)] &= \sum_{k=1,2,\ldots} k(k-1)p(k) \\ &= \sum_{k=1,2,\ldots} k(k-1)(1-p)^{1-k}p \\ &= p\sum_{k=1,2,\ldots} k(k-1)(1-p)^{1-k} \\ &= p\left(\frac{2(1-p)}{p^3}\right) = \frac{2(1-p)}{p^2}\end{aligned}$$

Finalmente,

$$\begin{aligned}\mathrm{Var}(X) &= E[X^2] - E^2[X] \\ &= E[X(X-1)+X] - E^2[X] \\ &= E[X(X-1)] + E[X] - E^2[X] \\ &= \frac{2(1-p)}{p^2} + \frac{1}{p} - \frac{1}{p^2} = \frac{1-p}{p^2}\end{aligned}$$

□

2.7.5 Variável Aleatória de Poisson

Uma variável aleatória X é *de Poisson com parâmetro* λ quando assumir um dos valores $i = 0, 1, 2, \ldots$ com função de probabilidade

$$p(i) = \frac{e^{-\lambda}\lambda^i}{i!}, \text{ para } i = 0, 1, 2, \ldots$$

A variável aleatória de Poisson encontra aplicação em diversas situações, como controle de qualidade e várias outras. Esses casos incluem aqueles em que se deseja avaliar a ocorrência de eventos de baixa probabilidade.

Em particular, a variável aleatória de Poisson pode ser utilizada como aproximação para a variável aleatória binomial, cujos parâmetros são (n, p). Nesses casos, os valores de n e p, que conduzem a uma boa aproximação, correspondem a situações em que n é muito grande e p é muito pequeno. Com essas condições, o produto np pode ser um valor mediano e utiliza-se a variável de Poisson com $\lambda = np$ para aproximar tal binomial.

Em relação à determinação do valor esperado e da variância da variável aleatória de Poisson, consideremos inicialmente a nossa intuição a respeito, onde a variável aleatória binomial de parâmetros (n, p) é aproximada pela variável de Poisson com $\lambda = np$. O valor esperado da variável binomial, que também será daquela de Poisson, é $np = \lambda$ e a variância é $np(1-p) = \lambda(1-p) \approx \lambda$, considerando p bem pequeno. Assim, podemos supor que tanto o valor esperado quanto a variância da variável aleatória de Poisson sejam ambos iguais a λ.

Proposição 2.22. *Seja X uma variável aleatória de Poisson com parâmetro λ, obtida como aproximação do valor esperado de uma variável aleatória binomial de parâmetros (n, p). Então o valor esperado e a variância da variável aleatória de Poisson são iguais, respectivamente, a*

$$E[X] = \lambda$$

$$\text{Var}(X) = \lambda$$

Demonstração. Para o que se segue, é necessário observar a conhecida igualdade

$$e^x = \sum_{i=0,1,\ldots} \frac{x^i}{k!}$$

Assim, temos que o valor esperado é dado por

$$\begin{aligned} E[X] &= \sum_{k=1,2,\ldots} kp(k) = \sum_{k=1,2,\ldots} k\left(\frac{e^{-\lambda}\lambda^k}{k!}\right) \\ &= \lambda e^{-\lambda} \sum_{k=1,2,\ldots} \frac{\lambda^{k-1}}{(k-1)!} \\ &= \lambda e^{-\lambda} \sum_{k=0,1,\ldots} \frac{\lambda^k}{k!} = \lambda e^{-\lambda} e^{\lambda} = \lambda \end{aligned}$$

Para a variância, primeiramente determinamos

$$\begin{aligned} E[X(X-1)] &= \sum_{k=1,2,\ldots} k(k-1)p(k) = \sum_{k=1,2,\ldots} k(k-1)\left(\frac{e^{-\lambda}\lambda^k}{k!}\right) \\ &= \lambda^2 e^{-\lambda} \sum_{k=2,3,\ldots} \frac{\lambda^{k-2}}{(k-2)!} \\ &= \lambda^2 e^{-\lambda} \sum_{k=0,1,\ldots} \frac{\lambda^k}{k!} = \lambda^2 e^{-\lambda} e^{\lambda} = \lambda^2 \end{aligned}$$

e, assim, temos finalmente que

$$\begin{aligned} \mathrm{Var}(X) &= E[X^2] - E^2[X] \\ &= E[X(X-1) + X] - E^2[X] \\ &= E[X(X-1)] + E[X] - E^2[X] \\ &= \lambda^2 + \lambda - \lambda^2 = \lambda \end{aligned}$$

□

Por diversas vezes, a variável aleatória de Poisson é utilizada quando o interesse é a determinação da probabilidade de ocorrência de eventos dentro de certo intervalo. Esse intervalo pode ser um período especificado de tempo, espaço ou alguma outra unidade. Por exemplo, determinar a probabilidade de um telefone celular receber um determinado número de chamadas erradas por dia. Nesse caso, o intervalo considerado é de um dia.

Como exemplo, suponha que um certo aparelho de telefone celular recebe, em média, 6 ligações erradas por dia. Pergunta-se:

a) Qual a probabilidade do celular receber exatamente 2 ligações em um dia?

b) Qual a probabilidade do celular receber até 2 ligações em um dia?

Para a solução, utilizaremos a variável aleatória de Poisson.

$$p(i) = \frac{e^{-\lambda}\lambda^i}{i!}$$

onde a variável λ representa a média, no caso igual a 6, e a variável i representa a quantidade de resultados, cuja probabilidade se deseja determinar.

a) Deseja-se determinar $p(2)$:

$$p(2) = \frac{e^{-6} \cdot 6^2}{2!} \approx \frac{0{,}00248 \cdot 36}{2} = 0{,}04464$$

b) Deseja-se determinar $\sum_{i=0}^{2} p(i)$:

$$p(0) = \frac{e^{-6} \cdot 6^0}{0!} \approx \frac{0{,}00248 \cdot 1}{1} = 0{,}00248$$

$$p(1) = \frac{e^{-6} \cdot 6^1}{1!} \approx \frac{0{,}00248 \cdot 6}{1} = 0{,}01488$$

$$p(2) = \frac{e^{-6} \cdot 6^2}{2!} \approx \frac{0{,}00248 \cdot 36}{2} = 0{,}04464$$

$$p(0) + p(1) + p(2) \approx 0{,}04464 + 0{,}01488 + 0{,}00248 \approx 0{,}062$$

Portanto, a probabilidade do aparelho receber até 2 chamadas erradas por dia é 0,062.

2.7.6 Desvio do Valor Esperado

Nas seções anteriores, introduzimos o conceito de valor esperado de uma variável aleatória como uma medida do seu valor médio ponderado pelas respectivas probabilidades. Além disso, mostramos que a variância é uma medida do desvio médio do valor da variável aleatória e seu valor esperado. No entanto, ainda é necessário mensurar o quão provável é o valor de uma variável aleatória se desviar de seu valor esperado.

A seguinte proposição fornece uma primeira medida desta natureza.

Teorema 2.23 (Desigualdade de Markov)**.** *Seja X uma variável aleatória não negativa. Para qualquer $a > 0$, temos que*

$$P(X \geqslant a) \leqslant \frac{E[X]}{a}$$

Deste resultado, é possível derivar um limite que utiliza a variância da variável aleatória, além de seu valor esperado:

Teorema 2.24 (Desigualdade de Chebyshev)**.** *Seja X uma variável aleatória. Para qualquer valor real a, temos que*

$$P(|X - E[X]| \geqslant a) \leqslant \frac{\mathrm{Var}(X)}{a^2}$$

Ambos limites anteriores são justos em geral, isto é, existem variáveis aleatórias para as quais os limites fornecidos pelos Teoremas 2.23 e 2.24 são verificados na igualdade. No entanto, é possível derivar desigualdades mais fortes para variáveis aleatórias mais restritas que são ainda interessantes na prática. Há diversos limites considerados na literatura e, como exemplo, apresentamos um frequentemente empregado.

Teorema 2.25 (Desigualdade de Chernoff)**.** *Seja $X = \sum_{i=1}^{n} X_i$ uma variável aleatória, onde X_i representa uma variável aleatória de Bernoulli de parâmetro p_i, $i = 1, 2, \ldots, n$. Portanto, $E[X] = \sum_{i=1}^{n} p_i$. Além disso, considere que X_i é independente de X_j para todo $i \neq j$. Para qualquer $\delta > 0$, temos que*

$$P(X - E[X] > (1 + \delta)E[X]) < \left(\frac{e^{\delta}}{(1 + \delta)^{(1+\delta)}} \right)^{E[X]}$$

e para qualquer $0 < \delta < 1$, temos que

$$P(X < (1 - \delta)E[X]) < e^{-\delta^2 E[X]/2}$$

Como exemplo, considere o lançamento de uma moeda honesta n vezes e X é a variável aleatória que determina o número ocorrido de caras. Portanto, X é uma variável aleatória binomial com $E[X] = n/2$ e $\mathrm{Var}(X) = n/4$. Queremos determinar a probabilidade de que pelo menos $3/4$ dos lançamentos resultem em cara. Intuitivamente, essa probabilidade deveria diminuir conforme n aumenta. Pela desigualdade de Markov, podemos concluir que

$$P\left(X \geqslant \frac{3n}{4}\right) \leqslant \frac{E[X]}{3n/4} = \frac{n/2}{3n/4} = \frac{2}{3}$$

A desigualdade de Chebyshev produz um limite melhor. Com efeito, como X se distribui simetricamente em torno do valor esperado, temos que

$$\begin{aligned} P\left(X \geqslant \frac{3n}{4}\right) &= \frac{1}{2} P\left(|X - E[X]| \geqslant \frac{n}{4}\right) \\ &\leqslant \frac{1}{2}\frac{\mathrm{Var}(X)}{(n/4)^2} = \frac{1}{2}\frac{n/4}{(n/4)^2} = \frac{2}{n} \end{aligned}$$

Note que o novo limite agora diminui com o aumento de n, enquanto o anterior é constante. Apliquemos, agora, o limite dado por Chernoff.

$$\begin{aligned} P\left(X > \frac{3n}{4}\right) &= P\left(X < \frac{n}{4}\right) \\ &= P\left(X < \left(1 - \frac{1}{2}\right)\frac{n}{2}\right) \\ &< e^{-(1/2)^2(n/2)/2} = \frac{1}{e^{n/16}} \end{aligned}$$

o que resulta em um limite ainda melhor que o anterior, pois o denominador da fração cresce exponencialmente mais rápido que o denominador da anterior.

2.8 Exercícios

2.1 Utilizando os axiomas de probabilidade, mostrar que o evento vazio possui probabilidade nula.

2.2 POS-COMP 2013
Considere um único lance de um dado não viciado. Assinale a alternativa que apresenta,corretamente, a probabilidade de insucesso em obter um 2 ou um 5.
a) 1/36 b) 1/12 c) 1/6 d) 1/3 e) 2/3

2.3 POS-COMP 2014
Um equipamento eletrônico tem dois componentes de armazenamento, A e B que são independentes. Trabalha-se com a probabilidade de falha no componente A de 20% e de falha no componente B de 15%. A probabilidade de ocorrer falha, simultaneamente nos dois componentes é:
a) 35% b) 30% c) 27% d) 12% e) 3%

2.4 ENEM 2013
Numa escola com 1.200 alunos foi realizada uma pesquisa sobre o conhecimento de duas línguas estrangeiras: inglês e espanhol. Nessa pesquisa constatou-se que 600 alunos falam inglês, 500 falam espanhol e 300 não falam qualquer um desses idiomas. Escolhendo-se um aluno dessa escola ao acaso e sabendo-se que ele não fala inglês, qual a probabilidade de que esse aluno fale espanhol?
a) 1/2 b) 5/8 c) 1/4 d) 5/6 e) 5/14

2.5 Se E, F são dois eventos de um espaço amostral, mostrar que

$$P(E \setminus F) = P(E) - P(E \cap F)$$

2.6 Mostrar que uma probabilidade condicional é, de fato, uma probabilidade, isto é, satisfaz os Axiomas 2.1, 2.2 e 2.3 de probabilidade.

2.7 UVA Online Judge 10056
Escreva um programa para o seguinte problema:
Um jogo com N jogadores consiste em lançar um dado com muitas faces numeradas (incluindo os números de 1 a N). Vence aquele jogador i que obtém o lançamento igual a i, sendo que nenhum jogador anterior obteve um lançamento bem sucedido. Se, ao final de N tentativas ninguém acertou, o jogo reinicia até alguém acertar. A probabilidade de um jogador i obter o lançamento igual a i é p. Dados N, p e i, determinar a probabilidade do jogador i vencer o jogo.

2.8 UVA Online Judge 11061
Escreva um programa para o seguinte problema:
No popular jogo War, os países do mundo são divididos entre os jogadores e cada jogador coloca um certo número de exércitos nos países de sua propriedade. A conquista de um país vizinho é feita com uma disputa com sucessivos lançamentos de dados. O atacante lança 3 dados se tiver mais de 3 exércitos, 2 se tiver 3 e 1 se tiver apenas 2 exércitos. O defensor lança 3 dados se tiver 3 ou mais exércitos, 2 se tiver 2 e 1 se tiver 1. Os lançamentos dos jogadores são ordenados inversamente e cada par das ordenações é comparado estabelecendo a disputa entre um exército de cada jogador. O atacante só ganha se o seu dado for maior que o do adversário. Um ataque termina quando o atacante ganha todos os exércitos do adversário, quando fica apenas com 1 exército ou se desistir do ataque. Dado o número de

exércitos N do defensor, determinar o número mínimo de exércitos que o atacante deve usar para ter uma probabilidade maior que 50% de conquistar o país do adversário.

2.9 UVA Online Judge 12461
Escreva um programa para o seguinte problema:
n pessoas embarcam em um avião com n lugares. O primeiro passageiro perdeu seu cartão de embarque. Então ele senta em um lugar aleatório. Cada passageiro subsequente senta no seu próprio lugar, se ele estiver disponível, ou em um lugar aleatório. Dado n, quer-se saber qual a probabilidade do n-ésimo passageiro encontrar seu lugar ocupado.

2.10 Escolhem-se ao acaso, e sem reposição, 8 números do conjunto $\{1, 2, \ldots, 25\}$. Qual o valor esperado da soma dos valores escolhidos?

2.11 Uma confeitaria oferece 20 tipos de trufas diferentes. São escolhidas aleatoriamente 8 trufas, sendo todas as escolhas com a mesma probabilidade. Qual o número esperado de tipos de trufas escolhidos?

2.12 Cinco pares diferentes de meias são colocados em uma lavadora, mas só 7 meias retornam. Qual o número esperado de pares de meias que retornam?

2.13 POS-COMP 2014
Suponha que numa empresa uma de suas máquinas de manufatura esteja sob avaliação de performance. Na produção de 8 lotes de peças, a máquina apresentou a seguinte sequência de peças defeituosas por lote: 9, 3, 8, 8, 9, 8, 9, 18. Nessas condições, assinale a alternativa que apresenta corretamente o desvio padrão de peças defeituosas em relação à média, ($S = \sqrt{\frac{\sum_x (x-\overline{x})^2}{N}}$), onde S é o desvio padrão, N é o número de elementos da amostra, x é o elemento da amostra, e $\overline{x}$ a média aritmética.
a) 0 b) $\sqrt{120}$ c) $\sqrt{15}$ d) 9 e) 72

2.14 Seja E o experimento relativo ao lançamento de um dado viciado tal que a probabilidade de obtenção de cada face é diretamente proporcional ao quadrado do valor de cada face. Seja X a variável aleatória igual à face obtida no lançamento desse dado.
a) Obtenha a função de probabilidade de X.
b) Calcule $E[X]$, $Var(X)$ e $\sigma(X)$.

2.15 (Ross 2010) Um dado honesto é lançado repetidamente, de modo indepen-

dente. Sejam X e Y o número de lançamentos necessários para obter um 6 e um 5, respectivamente. Determine:
a) $E[X]$
b) $E[X|Y = 1]$
c) $E[X|Y = 5]$

2.16 Sejam X uma variável aleatória binomial com parâmetros (n, p) e $p(i)$ a sua função de probabilidade. Mostrar que $\sum_i p(i) = 1$.

2.17 Um estudante preenche, por adivinhação, uma prova de múltipla escolha com 8 questões e 4 respostas possíveis em cada questão, uma das quais certa. Pergunta-se:
a) Qual a distribuição do número de respostas certas?
b) Qual a probabilidade de que o estudante obtenha 6 ou mais respostas certas?
c) Qual a probabilidade de que acerte pelo menos 2 questões?

2.18 Provar a Proposição 2.9.

2.19 Sejam X uma variável aleatória geométrica e $p(n)$ a sua função de probabilidade. Mostre que $\sum_1^\infty p(n) = 1$.

2.20 Mostrar que o valor esperado $E[X]$ de uma variável aleatória geométrica X é igual a $1/p$, onde p é a probabilidade de sucesso em cada experimento aleatório realizado.

2.21 (Ross 2010) Sejam X e Y variáveis aleatórias independentes, com a distribuição de Poisson com parâmetros respectivos λ_1 e λ_2. Considere $Z = X + Y$. Determine a distribuição condicional de X, dado que $Z = z$.

2.22 O número de falhas por dia na conexão com a Internet em dado local é uma variável aleatória de Poisson com parâmetro $1/3$.
a) Encontre a probabilidade de que haja 3 ou mais falhas em determinado dia.
b) Calcule essa probabilidade dado que já tenha ocorrido pelo menos uma falha nesse dia.

2.9 Notas Bibliográficas

A história da teoria da probabilidade remonta a tempos antigos. Na Grécia Antiga, Platão e Aristóteles utilizaram o termo "chance", no seu sentido não quantitativo. Um dos primeiros registros da aplicação de métodos matemáticos para resolver questões envolvendo probabilidades foi encontrado em correspondências mantidas entre Gerolamo Cardano, Pierre de Fermat e Blaise Pascal, nos séculos XVI e XVII. Naquela época, a aplicação residia basicamente no estudo de jogos, como de cartas, de moedas tipo cara-e-coroa, entre outros. Em particular, Pascal e Fermat mantiveram correspondência ativa sobre o tema. O primeiro tratado sobre probabilidades foi escrito pelo matemático holandês Christian Huygens no século XVII. Ainda naquele século, matemáticos da família de Bernoulli, em particular Jacob Bernoulli, realizaram contribuições importantes, que permanecem até hoje. Entre essas, a verificação de experimentos independentes, com a mesma probabilidade de sucesso p, questão abordada na Seção 2.7 do presente capítulo, ver (Bernoulli 1713). Durante este período, além de Pascal, Fermat e Bernoulli, deve ser mencionado Laplace, quem publicou o livro "Théorie Analytique de Probabilités", em 1812. Nessa época, já eram considerados conceitos como média, valor esperado e variáveis aleatórias. A motivação da variável aleatória de Poisson, provém da distribuição de probabilidades introduzida por Poisson (1837). Os axiomas de probabilidade, que constituem as bases da teoria moderna de probabilidade, foram estabelecidas por Kolmogorov (1933). A literatura corrente de probabilidades é vasta. No caso específico do tema do presente texto, isto é, com ênfase em algoritmos, o livro de probabilidades de Mitzenmacher e Upfal (2005) é indicado. Ele contém a utilização de técnicas probabilísticas, aplicadas a algoritmos randomizados. O livro de autoria de Ross (2010) é bastante utilizado como texto para um curso de probabilidades. Veja também o livro de Allan (2005). Em língua portuguesa, são conhecidos os livros de Fernandez (2005), bem como o escrito por Moreira e Kohayakawa (2010), o primeiro sobre probabilidades em geral, e o segundo com ênfase em combinatória. Mencionamos também o livro de Magalhães (2006) e o de Cunha et al. (2012). Também em língua portuguesa, próximo ao tema do presente texto, há o livro de Figueiredo et al. (2007), com ênfase em algoritmos randomizados, apresentado no Colóquio Brasileiro de Matemática de 2007.

3 Algoritmos Randomizados

3.1 Introdução

Neste capítulo, são estudados os algoritmos randomizados. Este estudo constitui uma das inúmeras aplicações do conceito de probabilidade, cujas propriedades básicas foram descritas no capítulo anterior. Trata-se de um tópico da área de algoritmos, ao qual tem sido atribuído espaço e importância crescentes nos últimos anos. Esse fato só tende a se acelerar no futuro próximo.

Os termos algoritmo randomizado, algoritmo aleatorizado e algoritmo probabilístico poderiam ser considerados como sinônimos, mas, em geral, há uma diferença no contexto em que são empregados. Algoritmo randomizado é o termo mais utilizado para designar o que se imagina ser um algoritmo desse tipo, isto é, aquele em que algumas de suas ações dependem da realização prévia de algum evento probabilístico. Por exemplo, quando alguma ação do algoritmo é determinada pelo resultado da geração de um número aleatório, que poderia ser representado pelo resultado do lançamento de um moeda. Há inclusive o termo algoritmo derandomizado, o qual designa um algoritmo determinístico obtido a partir de um algoritmo randomizado, por meio da transformação de cada ação probabilística em uma ação determinística. O termo algoritmo aleatorizado é pouco utilizado. Por outro lado, o termo “aleatorizado” suplantou o “randomizado” em algumas outras

ocorrências, por exemplo, para designar "passeios aleatórios" em um grafo, ao invés de "passeios randomizados". Finalmente, o termo algoritmo probabilístico, apesar de também empregado, é menos comum do que algoritmo randomizado. Atualmente, encontra-se com frequência a sua utilização para designar o tipo de análise realizada em algum algoritmo, por exemplo, na "análise probabilística de algoritmos", que menos frequentemente é chamada de "análise randomizada de algoritmos".

O capítulo se inicia com uma descrição do que significa exatamente *randomização*, em termos algorítmicos. Tratamos de justificar a utilização e vantagens do emprego de técnicas de randomização em relação aos tradicionais algoritmos determinísticos, para certos tipos de problemas. Em seguida, estudamos os dois principais tipos de algoritmos randomizados: algoritmos de Monte Carlo e de Las Vegas. Para cada um deles, o estudo é ilustrado com aplicações. Para o algoritmo de Monte Carlo, são examinados os problemas de identidades de polinômios, elemento majoritário de um conjunto, corte mínimo de arestas de um grafo e a descrição de um teste randomizado de primalidade. Para o algoritmo de Las Vegas, são descritas as aplicações de determinação de elementos unitários de um conjunto e a formulação do método de ordenação Quicksort randomizado. Finalmente, é discutida a possível conversão entre os tipos de algoritmos Monte Carlo e Las Vegas.

3.2 O Significado da Randomização

De um modo geral, um algoritmo pode ser compreendido como um processo, o qual, a partir de um certo conjunto de dados, computa um resultado. O conjunto de dados e a saída constituem a *entrada* e *saída* do algoritmo, respectivamente. Durante o seu processamento, o algoritmo executa as instruções das declarações especificadas em sua formulação. Se a computação do algoritmo depende unicamente de suas instruções aplicadas sobre o conjunto fixo de dados, independente de fatores externos variáveis, o algoritmo é dito *determinístico*. Por exemplo, suponha que desejamos avaliar um polinômio $P(x)$, por meio do cálculo do valor numérico $P(x_0)$, para um certo valor x_0. Se x_0 é um dado do algoritmo, conhecido para a computação, então o algoritmo que avalia $P(x)$, nesses termos, é determinístico.

Por outro lado, se a computação do algoritmo depende de fatores aleatórios ou da realização de experimentos aleatórios, o algoritmo é dito *aleatório* ou *randomizado*. Esses fatores ou experimentos podem ser de naturezas diversas, tais como decisões que dependem de um valor a ser sorteado, de um valor obtido do

lançamento de um dado ou moeda ou de maneira aleatória. Por exemplo, no caso da avaliação de um polinômio $P(x)$, por meio do cálculo do valor numérico de $P(x_0)$, o valor x_0 pode ser obtido através da leitura de dígitos obtidos do relógio do computador ou gerado aleatoriamente. Do ponto de vista computacional, naturalmente, um computador não dispõe de moedas ou dados para efetuar sorteios. Então, geralmente, utilizam-se programas especiais denominados geradores de números aleatórios, que, por sua vez, são constituídos de algoritmos determinísticos. Assim, na realidade, esses algoritmos geram números pseudoaleatórios.

À primeira vista, pode parecer estranho que a estratégia de usar aleatoriedade para o exame de certos passos do algoritmo possa ser de algum benefício. Pois, intuitivamente, o poder de escolher de forma determinística os rumos de um algoritmo deveria conduzir a melhores resultados do que deixá-los ao acaso. Contudo, são inúmeros os exemplos em que a randomização ocupa um papel de destaque na compreensão de certos fenômenos. Por exemplo, a física moderna utiliza largamente a randomização e leis da estatística. Da mesma forma, no estudo de genética, dos sistemas biológicos, em economia, de comportamento de mercados etc. A importância da randomização é um fenômeno recente, e na computação, mais recente ainda.

As principais vantagens da utilização de algoritmos randomizados sobre os determinísticos são de conseguir, em muitos casos, algoritmos mais rápidos ou de maior simplicidade. Mas, por outro lado, a randomização introduz incertezas na computação de um algoritmo. Essas incertezas podem se manifestar na correção dos resultados obtidos ou no tempo de processamento. Um algoritmo determinístico, em princípio, produz uma resposta sempre correta, mas pode requerer um tempo excessivo para obtê-la. Por outro lado, utilizando um algoritmo randomizado para o mesmo problema, seria possível obter resposta mais rápida, mas sem garantias de que esteja correta. Entretanto, outros tipos de algoritmos randomizados podem garantir a correção da resposta, mas sem garantia do tempo de processamento.

As duas alternativas acima constituem os tipos principais de algoritmos randomizados. Aqueles que garantem a obtenção rápida da resposta, mas sem garantia de correção, são denominados algoritmos de *Monte Carlo*, enquanto os que obtêm sempre respostas corretas, mas sem garantia do tempo, são algoritmos de *Las Vegas*.

Um dos fatores principais que atribui importância aos algoritmos randomizados é a análise realizada no sentido de quantificar a incerteza. Assim, utilizam-se métodos probabilísticos para determinar a probabilidade de que um algoritmo de Monte Carlo obtenha a resposta correta. Da mesma forma, por meio desses mé-

todos, o objetivo é obter a probabilidade de que um algoritmo de Las Vegas seja eficiente, em tempo. Conforme será verificado ao longo do texto, frequentemente essas probabilidades podem se aproximar de 1. Esse fato, sem dúvida, constitui a principal razão da importância da randomização.

3.3 Algoritmos de Monte Carlo

Conforme mencionado na seção anterior, um algoritmo de Monte Carlo é um algoritmo randomizado que produz uma resposta rápida, mas cuja correção não é garantida. Resposta rápida, no caso, significa um tempo polinomial no tamanho da entrada do algoritmo.

Inicialmente, iremos examinar os algoritmos de Monte Carlo aplicados a problemas de decisão. Um *problema de decisão* é aquele cuja solução consiste simplesmente em responder "sim" ou "não" a alguma questão. Naturalmente, existem problemas de decisão cujas soluções podem apresentar alto grau de dificuldade, ou mesmo não existir. Em particular, a classe dos problemas NP-Completos, bastante conhecida na literatura, é constituída de problemas de decisão para os quais não são conhecidos algoritmos eficientes, isto é, de complexidade polinomial. Os algoritmos de Monte Carlo aplicados a problemas de decisão podem apresentar uma correção de garantia parcial, para certos problemas. Assim, há algoritmos de Monte Carlo que garantem a correção da resposta "sim", mas não a correção da resposta "não". Esses são denominados *algoritmos de Monte Carlo com garantia no "sim"*. Analogamente, há algoritmos de Monte Carlo que podem garantir a correção do "não", mas não do "sim". Esses últimos são denominados *algoritmos de Monte Carlo com garantia no "não"*. De modo geral, ambos são denominados algoritmos de Monte Carlo com erro controlado. Finalmente, mencionamos que existem algoritmos de Monte Carlo para problemas de decisão que não garantem a correção de ambos, "sim" ou "não".

Como primeiro exemplo de algoritmo de Monte Carlo, apresentamos um algoritmo de Monte Carlo para o qual descrevemos um algoritmo com garantia no "não".

3.3.1 Problema: Identidade de polinômios

Uma forma canônica de representar polinômios é por meio de uma soma de monômios. Por exemplo, o polinômio $F(x) = x^2 - 7x + 12$ representa um polinômio de forma canônica, cujo grau é 2. Contudo, é bastante comum representar um polinômio de grau $k \geqslant 1$ sob a forma de produto de k polinômios de grau 1. Assim,

o polinômio $H(x) = (x-3)(x-4)$ representa um polinômio idêntico ao $F(x)$ anterior, mas apresentado sob a forma de produto de polinômios de grau 1.

Em seguida, trataremos do problema de decisão de verificar se dois polinômios dados, $F(x)$ e $H(x)$, de mesmo grau n, são idênticos ou não. Naturalmente, se ambos $F(x)$ e $H(x)$ forem dados sob a mesma forma de representação, seja a canônica, seja o produto de n polinômios de grau 1, o problema pode ser resolvido em tempo $O(n)$. Consideremos, agora, o caso em que um dos polinômios, $F(x)$, encontra-se na forma canônica e $H(x)$ é dado como um produto de n polinômios de grau 1. Naturalmente, o problema pode ser facilmente solucionado, por meio da conversão de $H(x)$ para a forma canônica, mediante a computação de $n-1$ produtos de polinômios. Contudo, essa conversão de representação requer $O(n^2)$ passos. Em seguida, descrevemos um algoritmo de Monte Carlo para esse problema, cuja complexidade é $O(n)$. O algoritmo retorna "sim", caso os polinômios sejam idênticos, e "não" caso contrário.

Algoritmo 3.1 Identidade de Polinômios

Dados: polinômios $F(x)$, $H(x)$, ambos de grau n
  Escolher, de maneira aleatória e uniforme, um inteiro k, $1 \leqslant k \leqslant 100n$
  a , $b \leftarrow F(k)$, $H(k)$
  se $a = b$ **então**
    retornar "sim"
  senão
    retornar "não"

Para a determinação da complexidade do algoritmo, supomos que a escolha aleatória e uniforme do inteiro k, entre 1 e $100n$, pode ser realizada em tempo constante, por meio de um gerador de números aleatórios adequado. A determinação de $F(k)$ e $H(k)$ requer $O(n)$ passos. Assim, a complexidade de tempo é $O(n)$. O espaço adicional necessário, além da entrada, é apenas $O(1)$.

O Algoritmo 3.1, contudo, pode apresentar respostas incorretas. O teorema seguinte discute a questão da correção do algoritmo.

Teorema 3.1. *A resposta retornada pelo Algoritmo 3.1 é necessariamente correta, caso seja "não". Se a resposta for "sim", ela estará incorreta com probabilidade* $\leqslant 1/100$.

Demonstração. O algoritmo retorna "não" quando $F(k) \neq H(k)$, o que significa que $F(x) \neq H(x)$. Isto é, a resposta "não" é sempre correta. Se o algoritmo retorna "sim", considere as seguintes alternativas. Se $F(x) = H(x)$, então a

resposta é novamente correta. Contudo, se $F(x) \neq H(x)$, a resposta é incorreta. Mas, para que esse erro possa ocorrer, é necessário que $F(x) \neq H(x)$ e $F(k) = H(k)$, isto é, $F(k) - H(k) = 0$. Ou seja, k é a raiz do polinômio $F(x) - H(x)$. Esse último possui grau $\leqslant n$. Mas $F(x) - H(x)$ possui n raízes, no máximo. Isto é, existem no máximo k valores entre 1 e $100n$, para os quais $F(k) = H(k)$. Como existem $100n$ números entre 1 e $100n$, a probabilidade de que o valor k escolhido satisfaça $F(k) = H(k)$ é $\leqslant n/100n = 1/100$. Logo, a probabilidade de que o Algoritmo 3.1 retorne a resposta incorreta para o "sim" é $\leqslant 1/100$. □

De acordo com o teorema anterior, a resposta "não" dada pelo Algoritmo 3.1 é sempre correta. Logo, trata-se de um algoritmo de Monte Carlo com garantia no "não".

Uma maneira simples de reduzir a probabilidade de erro do algoritmo é repetir o experimento, isto é, executar o algoritmo novamente. Neste caso, poderíamos repetir a execução do Algoritmo 3.1.

Seja k_2 o inteiro escolhido neste segundo experimento. Se $F(k_2) \neq H(k_2)$ o algoritmo retorna "não", e a resposta está correta. Se $F(k_2) = H(k_2)$, o algoritmo retorna "sim", e a probabilidade de que esteja incorreta é $\leqslant 1/10.000$. O método pode ser repetido iterativamente, e a probabilidade de que a resposta esteja incorreta decresce exponencialmente. Contudo, deve-se levar em conta que, em tempo $O(n^2)$, um algoritmo determinístico simples obtém a resposta sempre correta.

Em seguida, apresentamos um outro exemplo simples de utilização de algoritmos de Monte Carlo.

3.3.2 Problema: Elemento majoritário

Dado um conjunto S de tamanho n, formado de elementos arbitrários, possivelmente repetidos, um certo elemento é dito *majoritário* se ele ocorrer em S mais do que $n/2$ vezes. O problema consiste em descrever um algoritmo simples para decidir se S contém ou não um elemento majoritário. Cada elemento $s \in S$ satisfaz $s \leqslant N$, onde $n << N$.

O algoritmo randomizado de Monte Carlo, a seguir, obtém a solução, respondendo "sim" caso exista um elemento majoritário e "não", caso contrário.

A determinação da complexidade é imediata. A escolha aleatória e uniforme do índice k do vetor S requer tempo $O(1)$, assim como a determinação do valor $s = S[k]$. A determinação da frequência de s requer tempo $O(n)$. Logo, a complexidade do algoritmo é $O(n)$. Novamente, o espaço adicional à entrada do algoritmo é de apenas $O(1)$.

Algoritmo 3.2 Elemento majoritário

Dados: Multiconjunto S, $|S| = n$, cada $s \in S$ tal que $s \leqslant N$, e $n << N$.
  Escolher, de maneira aleatória e uniforme, um inteiro k, $1 \leqslant k \leqslant n$
  frequência $\leftarrow 0$
  para $i \leftarrow 1..n$ **:**
    se $S[i] = S[k]$ **então**
      frequência $\leftarrow$ *frequência* $+1$
  se *frequência* $> n/2$ **então**
    retornar "sim"
  senão
    retornar "não"

Naturalmente, o Algoritmo 3.2 pode retornar resposta incorreta. O teorema seguinte analisa a correção do algoritmo.

Teorema 3.2. *A resposta retornada pelo Algoritmo 3.2 é necessariamente correta, caso seja "sim". Se a resposta for "não", ela estará incorreta com probabilidade* $< 1/2$.

Demonstração. O algoritmo retorna "sim" somente quando $S[k]$ ocorre mais que $n/2$ vezes em S, isto é, $S[k]$ é majoritário em S. Neste caso, a resposta "sim" estará sempre correta. Se a frequência de $S[k]$ é $\leqslant n/2$, o algoritmo retorna "não", e considere as seguintes alternativas. Se, de fato, S não contém um elemento majoritário, a resposta dada pelo algoritmo está correta. Contudo, se S contém um elemento majoritário $S[k'] \neq S[k]$, então a resposta dada pelo algoritmo estará incorreta. Mas, nesse caso, a frequência de $S[k'] > n/2$. Portanto, a probabilidade de que o índice k, escolhido arbitrariamente, satisfaça $S[k] \neq S[k']$ é $< 1/2$. Logo, o algoritmo estará incorreto com probabilidade $< 1/2$. □

Como decorrência do Teorema 3.2, concluímos que o Algoritmo 3.2 é um algoritmo de Monte Carlo com garantia no "sim".

Da mesma forma como no Algoritmo 3.1, múltiplas execuções do Algoritmo 3.2 produzem decrescimento exponencial de sua probabilidade de erro. Assim, repetindo duas vezes a execução do algoritmo, é possível garantir uma probabilidade de erro $< 1/4$.

Descrevemos, agora, um exemplo de algoritmo de Monte Carlo, aplicado a um problema da teoria de grafos.

3.3.3 Problema: Corte mínimo de arestas

Seja $G(V, E)$ um grafo conexo, com conjunto de vértices V, e conjunto de arestas E, e $n = |V|, m = |E|$. Um *corte de arestas* de G é um subconjunto $C \subseteq E$, tal que o grafo $G - C$, obtido pela remoção das arestas de C, é desconexo. Diz-se que o corte C é *mínimo* quando a sua cardinalidade é a menor possível, dentre todos os cortes de arestas de G. Apresentaremos um algoritmo randomizado de Monte Carlo, para determinar um corte mínimo de G.

O algoritmo utiliza a seguinte operação aplicada sobre as arestas de G. Uma *contração* da aresta $(v, w) \in E$ é a operação que identifica os vértices v e w no grafo. Isto é, a contração de (v, w) obtém um novo grafo em que os vértices v e w são substituídos por um único vértice z, cujos vizinhos são os vizinhos de v, juntamente com os vizinhos de w. Observe que se v e w possuem um vizinho comum em G, a operação de contração produz arestas paralelas no novo grafo. Além disso, a contração produz sempre um laço.

Veja um exemplo de contração na Figura 3.1. A Figura 3.1 (b) representa o grafo da Figura 3.1 (a), após a contração da aresta $(3, 5)$, enquanto a Figura 3.1 (c) é o grafo da Figura 3.1 (a) após a contração da aresta $(7, 3)$. Note que essa última contração produz arestas paralelas.

A ideia básica do algoritmo é efetuar repetidas operações de contração de arestas, cada qual realizada sobre o grafo obtido como resultado da contração anterior. A cada contração realizada, o número de vértices é reduzido de uma unidade. Se o grafo inicial possui n vértices, ao final de $n - 2$ contrações, o grafo se reduz a dois vértices. Entre esses dois vértices finais há, possivelmente, múltiplas arestas paralelas. Esse conjunto de arestas paralelas corresponde, no grafo inicial, a um corte de arestas. A questão é determinar a probabilidade de que esse corte seja mínimo. Em todas as contrações realizadas, os laços criados são removidos.

A descrição anterior corresponde à formulação do Algoritmo 3.3.

Algoritmo 3.3 Corte Mínimo

Dados: Grafo $G(V, E)$, conexo, $|V| = n$.
$G_n(V_n, E_n) \leftarrow G$
para $i \leftarrow n, (n-1), \ldots, 3$ **:**
Escolher aresta $e \in E_i$, de forma aleatória e uniforme.
$G_{i-1}(V_{i-1}, E_{i-1})$ é o grafo resultante da contração de $e \in E_i$, removendo os possíveis laços formados.
retornar corte $C \leftarrow E_2$

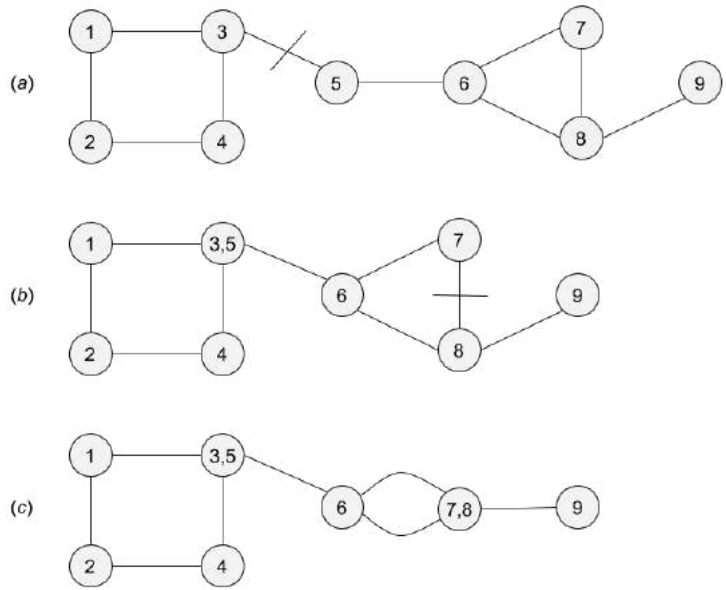


Figura 3.1: Contração de arestas

A Figura 3.2 ilustra um exemplo da aplicação do algoritmo. O grafo da Figura 3.2(a) contém 6 vértices, portanto o algoritmo efetua as contrações das quatro seguintes arestas, $(1,2)$, $(4,6)$, $(3,5)$, $(1,3)$, respectivamente, até transformá-lo no grafo da Figura 3.2(e). Esse último possui dois vértices e três arestas. Então a resposta obtida pelo algoritmo será um corte de tamanho 3. Observe que esse corte não é mínimo, visto que o grafo possui cortes de tamanho 2.

Um limite superior para a complexidade do algoritmo pode ser facilmente obtido. O algoritmo efetua $n-2$ contrações de arestas. Sem utilizar estruturas de dados especiais, cada contração pode ser realizada em tempo $O(n)$. Assim, no máximo $O(n^2)$ operações seriam suficientes para obter o corte.

A justificativa do algoritmo pode ser realizada pela seguinte proposição.

Lema 3.3. *Seja $G(V,E)$ um grafo conexo, no qual foram realizadas $n-2$ operações de contração de arestas, de forma iterativa. Sejam $G_n(V_n,E_n) = G(V,E)$, e $G_{n-1}(V_{n-1},E_{n-1})$, ..., $G_2(V_2,E_2)$ os grafos resultantes dessas contrações, respectivamente, em que os laços foram todos removidos. Então*

(i) G_2 é um multigrafo com dois vértices, e E_2 é um corte de G_2,

(ii) Se C é um corte de G_i, $2 < i \leqslant n$, e G_{i-1} obtido pela contração de

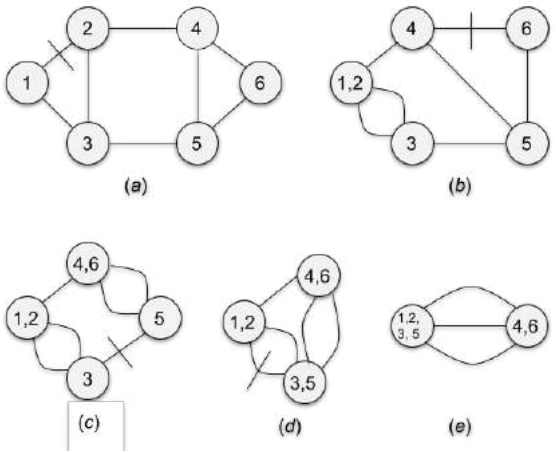


Figura 3.2: Contração de arestas

alguma aresta de $e \in E_i \setminus C$*, então* C *permanece como corte de* G_{i-1}.

Demonstração.

(i) Cada contração reduz o número de vértices de uma unidade, logo G_2 possui 2 vértices, e E_2 é um conjunto de arestas incidentes a ambos, portanto um corte de G_2.

(ii) A única possibilidade de uma contração de aresta de G_i eliminar um corte C é que essa aresta seja incidente, respectivamente, a vértices de dois conjuntos separados pelo corte C. Mas, nesse caso, a aresta contraída pertence a C, uma contradição. Então C permanece como corte de G_{i-1}. □

Corolário 3.4. *Seja* G *um grafo no qual foi aplicado o Algoritmo 3.3. Seja* C *um corte mínimo de* G*. Então o algoritmo retorna esse corte mínimo se e somente se nenhuma das arestas de* C *foi eliminada pelas* $n - 2$ *contrações realizadas.*

O Algoritmo 3.3 é, portanto, um algoritmo de Monte Carlo. Resta, ainda, determinar a probabilidade de que a resposta obtida esteja correta, o que será realizado a seguir.

Teorema 3.5. *Seja* C *um dado corte mínimo de um grafo conexo* $G(V, E)$*. Então o Algoritmo 3.3 encontra o corte* C *com probabilidade* $> 2/n^2$.

Demonstração. Inicialmente, obtemos uma relação entre o número de vértices n, número de arestas m, e o tamanho $c = |C|$. Seja $grau(v)$ o grau de um vértice

$v \in V$. Sabemos que $\sum grau(v) = 2m$. Por outro lado, o grau de um vértice qualquer é um limite superior para o corte mínimo do grafo, pois o grafo se torna desconexo ao remover todas as arestas incidentes a um mesmo vértice. Logo,

$$c \leqslant grau(v)$$

Considerando os n vértices, obtemos

$$nc \leqslant \sum grau(v) = 2m$$

isto é

$$m \geqslant \frac{nc}{2}$$

Em seguida, consideramos as $n - 2$ iterações realizadas pelo algoritmo. Sejam $G_{n-1}(V_{n-1}, E_{n-1}), \ldots, G_2(V_2, E_2)$, respectivamente, os grafos resultantes das operações de contração efetuadas. Sabemos que o corte obtido pelo algoritmo é o conjunto E_2. De acordo com o Corolário 3.4, para que se obtenha $E_2 = C$, é necessário que nenhuma aresta de C tenha sido contraída em qualquer das $n - 2$ contrações de arestas realizadas pelo algoritmo.

Desejamos computar a probabilidade $P(E_2 = C)$. Denote por ϵ_i o evento em que nenhuma aresta de C é contraída pelo algoritmo, na computação de G_{i-1} a partir de G_i. Logo,

$$P(E_2 = C) = P(\epsilon_n \cap \epsilon_{n-1} \cdots \cap \epsilon_3)$$

Sabemos que o evento ϵ_i ocorre condicionalmente à ocorrência dos eventos ϵ_{i+1}, ϵ_{i+2}, ..., ϵ_n. Logo,

$$P(\epsilon_n \cap \epsilon_{n-1} \cdots \cap \epsilon_3) = P(\epsilon_n) P(\epsilon_{n-1} \mid \epsilon_n) P(\epsilon_{n-2} \mid \epsilon_n, \epsilon_{n-1}) \cdots$$

Em cada grafo G_i, seja $n_i = |V_i|$ e $m_i = |E_i|$. Consideremos, agora, o evento ϵ_i. A probabilidade de que pelo menos uma das arestas de C seja escolhida nessa ocasião é c/m_i, tendo em vista que as escolhas são uniformes e arbitrárias. Como nenhuma aresta de C é escolhida, obtemos

$$P(\epsilon_i \mid \epsilon_n, \epsilon_{n-1}, \ldots, \epsilon_{i+1}) = 1 - \frac{c}{m_i}$$

Conforme determinado anteriormente,

$$m_i \geqslant \frac{n_i c}{2}$$

Logo,

$$1 - \frac{c}{m_i} \geqslant 1 - \frac{c}{n_i c/2} = 1 - \frac{2}{n_i}$$

e

$$P(\epsilon_i \mid \epsilon_n, \epsilon_{n-1}, \ldots, \epsilon_{i+1}) \geqslant 1 - \frac{2}{n_i}$$

Logo,

$$\begin{aligned} P(E_2 = C) &= P(\epsilon_n) P(\epsilon_{n-1} \mid \epsilon_n) P(\epsilon_{n-2} \mid \epsilon_n, \epsilon_{n-1}) \ldots P(\epsilon_2 \mid \epsilon_n, \cdots, \epsilon_3) \\ &\geqslant \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{3}\right) = \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} = \\ &= \frac{2 \cdot 1}{n(n-1)} = \frac{2}{n(n-1)} > \frac{2}{n^2} \end{aligned}$$

Portanto,

$$P(E_2 = C) > \frac{2}{n^2}$$

□

A exemplo de outros algoritmos de Monte Carlo, uma maneira de aumentar a probabilidade de obter uma resposta correta consiste em repetir a execução do algoritmo várias vezes. No caso específico do Algoritmo 3.3, o Teorema 3.5 informa que a probabilidade de que a resposta de uma execução do algoritmo esteja correta é $> \frac{2}{n^2}$. Isto é, a probabilidade de que a resposta esteja incorreta é $\leqslant 1 - \frac{2}{n^2}$. Repetindo a execução k vezes e escolhendo o menor valor de corte obtido nessas execuções, a probabilidade de erro é $\leqslant (1 - \frac{2}{n^2})^k$.

3.3.4 Problema: Teste de Primalidade

O principal fundamento para a maioria dos testes de primalidade utilizados atualmente é o resultado conhecido como *Pequeno Teorema de Fermat*, descrito a seguir.

Teorema 3.6. *(Fermat 1640). Seja p um inteiro primo. Então, para qualquer inteiro a satisfazendo $1 \leqslant a < p$, temos*

$$a^{p-1} \mod p = 1$$

O Teorema 3.6 pode então ser empregado para verificar se um dado número p é primo. Tal teorema garante que, quando p é primo, então, para qualquer inteiro a, usualmente denominado *base*, escolhido entre 2 e $p-1$, se elevarmos a à potência $p-1$ e tomarmos o resto da divisão desse resultado por p, o resultado será sempre igual a 1. O teste é conclusivo para a resposta "não" e, nesse caso, o número p é, com certeza, um número composto. Entretanto, se a resposta for "sim", pode estar errada em alguns casos. Por exemplo, $2^{560} \mod 561 = 1$, sugerindo que o número 561 possa ser primo. Entretanto, o resultado $3^{560} \mod 561 = 375$ garante que o número não é primo, sendo 3 dito uma *testemunha* do número 561 ser composto.

O Algoritmo 3.4 implementa o teste, sendo, portanto, um algoritmo de Monte Carlo com garantia no "não".

Algoritmo 3.4 Teste de Primalidade de Fermat

```
função PotênciaModular(a, b, p):
    se b = 0 então
        retornar 1
    senão
        m ← PotênciaModular(a, ⌊b/2⌋, p)
        se b é par então
            retornar (m * m) mod p
        senão
            retornar (m * m * a) mod p
Dados: inteiro p.
Escolher, de maneira aleatória e uniforme, um inteiro a, 2 ⩽ a ⩽ p − 1
se PotênciaModular(a, p − 1, p) = 1 então
    retornar "sim"
senão
    retornar "não"
```

Há inúmeros aspectos envolvendo o teste descrito. Inicialmente comentaremos o algoritmo e, em seguida, a probabilidade de acerto para a resposta "sim".

O Algoritmo 3.4 é uma apresentação didática das ideias relacionadas ao teste de primalidade baseado no Teorema 3.6. A função básica PotênciaModular está descrita como uma recursão que, sucessivamente, obtém o resultado em módulo p da potência de a pela metade do expoente $p-1$ e estende o resultado obtido. Na prática, ela é implementada de forma iterativa, por operações relacionadas aos bits dos inteiros envolvidos, com complexidade $O(\log^3 p)$. Isso porque as aplicações

práticas estão relacionadas à criptografia, onde usam-se inteiros muito grandes, representados com alguns milhares de bits e é necessária grande eficiência nos testes.

Outro aspecto importante é saber qual a probabilidade da resposta "sim" do algoritmo estar correta. Os *números de Carmichael* são inteiros compostos n, tais que, para todas as bases a entre 2 e $n-1$, relativamente primas a n, o teste de Fermat é positivo, indicando falsamente que n é primo. Para os demais compostos, a seguinte proposição é válida.

Teorema 3.7. *Seja n um inteiro composto que não é um número de Carmichael. Então, no máximo, a metade das bases entre 1 e $n-1$ satisfazem o teste de Fermat.*

O Teorema 3.7 permite concluir que a probabilidade de falsos positivos para um inteiro n composto que não seja um número de Carmichael é, no máximo $\frac{1}{2}$, quando se utiliza apenas uma base no teste. Essa probabilidade de erro pode ser diminuída até um limite desejado, pela repetição do teste com outras bases, tal como feito para outros algoritmos de Monte Carlo.

O fato de que os números de Carmichael influenciam a probabilidade mencionada é irrelevante, pois, embora existam infinitos desses números, a sua ocorrência é muito rara, conforme atesta o limite superior, dado pelo teorema a seguir.

Teorema 3.8. *(Pinch 1993.) Seja $C(n)$ a quantidade de números de Carmichael inferiores a n. Então*

$$C(n) \leqslant ne^{-(\ln n)(\ln\ln\ln n)/(\ln\ln n)}$$

O limite superior pelo teorema acima não é justo, conforme pode-se ver pelo seguinte exemplo. Para $n = 2,5 \cdot 10^{10}$ a fórmula indica um limite superior de 4115019 (0,016%) números de Carmichael, quando, na realidade, há apenas 2163 (0,000009%). Desta forma, a adoção da probabilidade de erro de $\frac{1}{2}$ no teste de primalidade de Fermat é bastante razoável.

O teste, como descrito, é usado em alguns sistemas práticos. Entretanto, em sistemas criptográficos que exigem grande robustez dos resultados de primalidade, tais como o RSA, ele foi ampliado com outros mecanismos capazes de detectar melhor os pseudoprimos, ou seja, as situações de falso positivo. O mais conhecido é o teste Miller–Rabin. Resta dizer que, embora haja um algoritmo polinomial para o teste de primalidade, o algoritmo AKS, seu desempenho na prática, ainda é muito inferior aos testes probabilísticos.

3.4 Algoritmos de Las Vegas

Nesta seção, examinaremos os algoritmos de Las Vegas. Relembrando os conceitos, os algoritmos de Las Vegas são algoritmos randomizados, cuja resposta está sempre correta, porém o tempo de execução, em princípio, não está determinado. Na realidade, esse poderá ser tratado como uma variável aleatória. Ou seja, a incerteza reside no tempo e não na correção.

Na Seção 3.3, foram abordados os algoritmos de Monte Carlo. Nestes últimos, os papéis desempenhados pela correção e tempo são exatamente os opostos dos algoritmos de Las Vegas. O tempo de execução de um algoritmo de Monte Carlo é determinístico, em geral polinomial, mas sua resposta nem sempre está correta. Ou seja, a incerteza do algoritmo de Monte Carlo está na correção.

Como primeiro exemplo, iremos considerar um caso simples. Aproveitaremos para ilustrar as diferenças entre as três abordagens de algoritmos: determinístico, Monte Carlo e Las Vegas.

3.4.1 Problema: Elemento unitário

Seja C um conjunto de tamanho n, arbitrariamente grande. Os elementos de C são todos binários, isto é, iguais a 0 ou 1. Supondo n par, exatamente a metade dos elementos são iguais a 0, e a outra metade, 1. A questão é determinar a posição (índice em C) de um elemento qualquer unitário. Sabemos também que os elementos estão uniformemente distribuídos em C, nas posições $C[1], \ldots, C[n]$.

Descrevemos, inicialmente, o Algoritmo 3.5 determinístico para resolver o problema.

Algoritmo 3.5 Elemento unitário (determinístico)

Dados: Conjunto binário C, $|C| = n$ par, $C[i] = 0$ para $n/2$ elementos.
  $i \leftarrow 1$
  enquanto $C[i] = 0$ **:**
    $i \leftarrow i + 1$
  retornar i

O número de passos realizados pelo algoritmo, no pior caso, é $\frac{n}{2} + 1$.

Para efeito de comparação, seja a mesma questão, resolvida através do Algoritmo 3.6 de Monte Carlo.

O número de passos realizado pelo Algoritmo 3.6 é $O(1)$. Mas sua resposta pode não estar correta. Como os elementos 0 e 1 estão uniformemente distribuídos

Algoritmo 3.6 Elemento unitário (Monte Carlo)

Dados: Conjunto binário C, $|C| = n$ par, $C[i] = 0$ para $n/2$ elementos.
Escolher, aleatoriamente, um índice i, $1 \leqslant i \leqslant n$
retornar i

em C, com a metade deles igual a 0, e a outra metade igual a 1, a probabilidade de que a resposta esteja correta é $\frac{1}{2}$. Repetindo-se a execução do algoritmo k vezes, a complexidade torna-se igual a $O(k)$, mas a probabilidade de obter uma resposta correta cresce para $1 - \left(\frac{1}{2}\right)^k$.

Seja, agora, a mesma questão resolvida através de um algoritmo de Las Vegas, pela formulação do Algoritmo 3.7.

Algoritmo 3.7 Elemento unitário (Las Vegas)

Dados: Conjunto binário C, $|C| = n$ par, $C[i] = 0$ para $n/2$ elementos.
repetir
Escolher, aleatoriamente, um índice i, $1 \leqslant i \leqslant n$
até que $C[i] = 1$
retornar i

A resposta dada pelo Algoritmo 3.7 está sempre correta, pois o algoritmo segue escolhendo, aleatoriamente, um índice i até que a condição $C[i] = 1$ seja satisfeita. Quanto ao número de passos efetuados até que seja encontrado $C[i] = 1$, a resposta evidentemente, será dada em termos probabilísticos. Sabemos que a metade dos elementos de C satisfaz $C[i] = 0$, e a outra metade, $C[i] = 1$. Tratando o número de passos como uma variável aleatória, e levando em consideração que seus valores refletem uma média ponderada de suas respectivas probabilidades, chegamos à conclusão que o valor esperado do número de passos efetuados pelo algoritmo de Las Vegas é igual a 2.

Observe as diferenças de comportamento das três soluções apresentadas: determinística, Monte Carlo e Las Vegas.

3.4.2 Problema: Quicksort

Examinamos, a seguir, um outro exemplo para algoritmos de Las Vegas, no caso, o algoritmo de ordenação conhecido como Quicksort.

Novamente, para efeito de comparação, descrevemos, inicialmente, um algoritmo determinístico e, em seguida, um algoritmo randomizado.

Dado um conjunto C, com valores $C(1), \ldots, C(n)$, permutar seus elementos de modo a colocá-los em ordem não decrescente $C(1) \leqslant C(2) \ldots \leqslant C(n)$.

O método do Quicksort para o problema de ordenação consiste basicamente, nos seguintes passos:

1. Escolher um elemento $C(p) \in C$, denominado *pivô*.

2. Particionar C em três subconjuntos $C' \cup \{C(p)\} \cup C'' = C$. O subconjunto C' contém aqueles elementos de $C \setminus \{C(p)\}$ que são menores ou iguais a $C(p)$, enquanto o subconjunto C'' contém aqueles maiores que $C(p)$.

3. Supondo que os conjuntos C' e C'' possam ser ordenados através da execução recursiva desse método, a ordenação final de C será $C' \| \{C(p)\} \| C''$, onde o símbolo $\|$ significa "concatenação".

O algoritmo a ser descrito é recursivo. O particionamento do conjunto C, mencionado anteriormente, é realizado em blocos de elementos consecutivos $C(i), \ldots, C(j)$, onde $1 \leqslant i \leqslant j \leqslant n$. Para um dado pivô $C(p)$, $i \leqslant p \leqslant j$, os subconjuntos C' e C'' podem ser facilmente obtidos, transferindo-se os elementos assim obtidos em tempo linear.

Uma primeira alternativa de particionar o conjunto C, mais elegante e eficiente em termos de espaço, consiste em computar os subconjuntos C' e C'', sobre o próprio espaço C. A configuração de C, ao final do processo de partição, conterá os elementos de $C \setminus C(p)$ que são $\leqslant C(p)$, seguido do pivô $C(p)$ e dos elementos de C maiores que $C(p)$. Para esse particionamento, utilizamos os ponteiros q e t. O ponteiro q somente é incrementado quando algum elemento $\leqslant C(p)$ é detectado, enquanto que o ponteiro t percorre todos os elementos de $C(i), \ldots, C(j-1)$. Neste esquema, o pivô $C(p)$ escolhido é sempre o último elemento do conjunto C corrente, isto é, $p = j$. O conjunto C' é acomodado nas primeiras posições, mais à esquerda em C, enquanto que são atribuídas as posições mais à direita aos elementos de C''.

Suponha que o elemento de C em exame seja $C(t)$, para algum t, $i \leqslant t \leqslant j$. Se $C(t) > C(p)$, nenhuma ação é realizada. Caso contrário, $C(t) \leqslant C(p)$ e, nesse caso, o ponteiro q é incrementado e os elementos de índices q e t trocam posições em C. Ao final do processo, após t atingir a posição j, o pivô $C(p) = C(j)$ troca de posição com $C(q+1)$. Com isso, os elementos de C' ocupam as posições $C(i), \ldots, C(q)$, o pivô se encontra na posição $C(q+1)$ e os elementos de C'' correspondem aos das posições $C(q+2), \ldots, C(j)$. Ver Figura 3.3. O processo é realizado por meio de uma função denominada PARTIÇÃO(C, i, j), que retorna

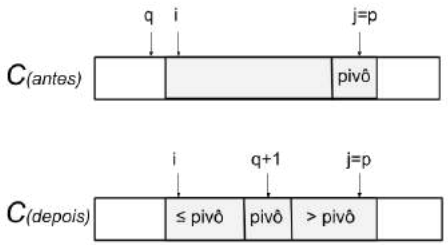


Figura 3.3: Esquema da partição do intervalo $[i, j]$ usada no Quicksort

a posição l tal que os elementos do conjunto $C(i), \ldots, C(l)$ são todos $\leqslant C(l)$ e os elementos do conjunto $C(l+1), \ldots, C(j)$ são $> C(l)$. A função é aplicada recursivamente sobre conjuntos $C(i), \ldots, C(j)$. Essa chamada é realizada, por sua vez, dentro do procedimento recursivo QUICKSORT(C, i, j). Ao final de PARTIÇÃO(C, i, j), o conjunto C estará particionado nos conjuntos $C' \cup C(p) \cup C''$, enquanto que, ao final de QUICKSORT(C, i, j), o conjunto C estará ordenado entre os índices i e j. A chamada externa é QUICKSORT$(C, 1, n)$. O primeiro pivô escolhido é $C(p) = C(n)$.

O algoritmo formal é descrito no Algoritmo 3.8.

Como exemplo da aplicação do algoritmo, seja ordenar os elementos do conjunto

	1	2	3	4	5	6	7
C	3	2	8	1	9	6	4

contendo 7 elementos. A primeira chamada recursiva é QUICKSORT$(C, 1, 7)$. O primeiro pivô escolhido é $C(7) = 4$, o qual produz a chamada PARTIÇÃO$(C, 1, 7)$, que retorna o valor $q = 4$. O conjunto C obtido ao final da computação de PARTIÇÃO$(C, 1, 7)$ é

	1	2	3	4	5	6	7
C	3	2	1	4	9	6	8

Em seguida, a chamada recursiva QUICKSORT$(C, 1, 3)$ produzirá o conjunto

	1	2	3	4	5	6	7
C	1	2	3	4	9	6	8

Algoritmo 3.8 Quicksort determinístico

função PARTIÇÃO(C, i, j):
  $q \leftarrow i - 1$
  para $t \leftarrow i, \ldots, j - 1$ **:**
    se $C(t) \leqslant C(j)$ **então**
      $q \leftarrow q + 1$
      $C(q), C(t) \leftarrow C(t), C(q)$
  $C(q+1), C(j) \leftarrow C(j), C(q+1)$
  retornar $q + 1$

procedimento QUICKSORT(C, i, j):
  se $i < j$ **então**
    $l \leftarrow$ PARTIÇÃO(C, i, j)
    QUICKSORT($C, i, l - 1$)
    QUICKSORT($C, l + 1, j$)

Dados: Conjunto C, de elementos $C(1), \ldots, C(n)$
QUICKSORT($C, 1, n$)

e a chamada QUICKSORT($C, 5, 7$) modificará C para

	1	2	3	4	5	6	7
C	1	2	3	4	6	8	9

que corresponderá ao final do processo.

Para determinar a complexidade, observemos que o método divide o conjunto C em três subconjuntos C', $\{C(p)\}$ e C'' por meio do procedimento PARTIÇÃO. Sabemos que $\{C(p)\}$ é unitário, e que os tamanhos de C' e C'' dependerão do pivô $C(p)$ escolhido. Suponha que $|C'| = c'$. Então $|C''| = n - c' - 1$. Denotando por $T(n)$, o número de comparações efetuadas entre elementos de C durante a ordenação, e considerando que a função PARTIÇÃO efetua $O(n)$ comparações dessa natureza, obtemos:

$$T(n) = \begin{cases} 0, \text{ se } n = 0 \\ T(c') + T(n - c' - 1) + O(n), \text{ caso contrário} \end{cases}$$

O pior caso corresponde à situação em que uma das partes é vazia, isto é, quando

$c' = 0$ ou $n - c' - 1 = 0$, o que conduz a

$$T(n) = T(0) + T(n-1) + O(n) = O(n^2),$$

enquanto que o melhor caso corresponde a $c' = \lfloor \frac{n}{2} \rfloor$ ou $c' = \lceil \frac{n}{2} \rceil$, o que conduz a

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + O(n) = O(n \log n).$$

No melhor caso, os conjuntos C' e C'' são balanceados, isto é, são de cardinalidades aproximadamente iguais. Observemos que este objetivo pode ser alcançado pela escolha da mediana de C para pivô. Como determinar a mediana de um conjunto pode ser realizado em tempo linear, segue-se que o algoritmo Quicksort determinístico pode ser sempre implementado na complexidade acima, isto é, $O(n \log n)$.

Descrevemos, em seguida, o algoritmo randomizado para Quicksort. O algoritmo randomizado segue, em linhas gerais, a mesma estratégia do algoritmo determinístico. A diferença básica consiste no método de escolha do pivô. Enquanto que o determinístico escolhe sempre o último do conjunto, no randomizado a escolha do pivô é aleatória, na qual qualquer elemento do conjunto pode assumir o papel de pivô. Embora a estratégia dos dois algoritmos seja similar, o método de escolha do pivô introduz diferenças na formulação do algoritmo e, principalmente, na sua análise. Mas, uma vez escolhido o pivô, o algoritmo randomizado utiliza a mesma ideia de particionar o conjunto dado, de acordo com o valor do pivô, conjugado a processos recursivos.

Utilizamos a mesma notação empregada anteriormente. Ou seja, dado um conjunto C, com elementos $C(1), \ldots, C(n)$, a serem ordenados, representamos por PARTIÇÃO-RAND(C, i, j) o procedimento que particiona os elementos do conjunto $\{C(i), C(i+1), \ldots, C(j)\} \setminus \{C(r)\}$, de acordo com o pivô $C(r)$, onde $i \leqslant r \leqslant j$, escolhido aleatoriamente. O procedimento PARTIÇÃO-RAND troca de posição $C(r)$ com $C(j)$, isto é, coloca o pivô aleatório na última posição de C, de modo que a estratégia empregada no procedimento PARTIÇÃO(C, i, j) determinístico possa ser utilizada. À exceção dessa última questão, os demais passos do algoritmo são similares, conforme as formulações seguintes. O procedimento principal é denotado por QUICKSORT-RAND(C, i, j), e a chamada externa é QUICKSORT-RAND$(C, 1, n)$.

Em seguida, realizamos a análise do Quicksort randomizado. Para simplificar a análise, assumimos que todos os elementos de C sejam distintos.

Algoritmo 3.9 Quicksort randomizado

função PARTIÇÃO(C, i, j):
  $q \leftarrow i - 1$
  para $t \leftarrow i, \ldots, j - 1$ **:**
    se $C(t) \leqslant C(j)$ **então**
      $q \leftarrow q + 1$
      $C(q), C(t) \leftarrow C(t), C(q)$
  $C(q+1), C(j) \leftarrow C(j), C(q+1)$
  retornar $q + 1$

função PARTIÇÃO-RAND(C, i, j):
  $r \leftarrow$ índice entre i e j escolhido aleatoriamente
  $C(r), C(j) \leftarrow C(j), C(r)$
  retornar PARTIÇÃO(C, i, j)

procedimento QUICKSORT-RAND(C, i, j):
  se $i < j$ **então**
    $l \leftarrow$ PARTIÇÃO-RAND(C, i, j)
    QUICKSORT-RAND($C, i, l - 1$)
    QUICKSORT-RAND($C, l + 1, J$)

Dados: Conjunto C, de elementos $C(1), \ldots, C(n)$.
QUICKSORT-RAND($C, 1, n$)

O objetivo é determinar o valor esperado do número de passos utilizados pelo algoritmo, isto é, pelo procedimento QUICKSORT-RAND(C, i, j) para efetuar a ordenação desejada. O progresso da computação se dá através da realização de operações de comparação entre elementos de C, as quais são dominantes no algoritmo. Assim, o objetivo consiste em determinar o valor esperado do número de comparações efetuadas.

Denote por X a variável aleatória representando o número total de comparações efetuadas pelo algoritmo. Para avaliar o valor de X, observe, inicialmente, que cada par de elementos $C(a), C(b) \in C$, onde $a < b$, é comparado entre si no máximo uma vez. A pergunta de interesse é conhecer em que condições $C(a), C(b)$ são comparados no decorrer do processo. A resposta a essa questão é simples: $C(a), C(b)$ são comparados quando as seguintes condições são ambas satisfeitas:

(i) Algum subproblema de QUICKSORT-RAND contém ambos $C(a), C(b)$, isto é, uma chamada de QUICKSORT-RAND(C, i, j) é realizada, onde $i \leqslant a < b \leqslant j$,

(ii) $C(a)$ ou $C(b)$ são escolhidos como pivô, em PARTIÇÃO-RAND(C, i, j), isto é, esse último procedimento atribui $r \leftarrow a$ ou $r \leftarrow b$.

Concluímos que, se uma das condições acima não for satisfeita, o algoritmo não tenta comparar $C(a)$ com $C(b)$.

Represente por $X_{a,b}$ o indicador do evento em que $C(a), C(b)$ foram comparados. Isto é

$$X_{a,b} = \begin{cases} 1, & \text{se } C(a), C(b) \text{ foram comparados no algoritmo} \\ 0, & \text{caso contrário} \end{cases}$$

Então X representa o somatório dos valores X_{ab}, isto é,

$$X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{a,b}$$

Seja $E[X]$ o valor esperado do número de comparações. Então

$$E[X] = E\left[\sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{a,b}\right]$$

Pela linearidade do valor esperado,

$$E[X] = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[X_{a,b}]$$

Como $X_{a,b}$ é um indicador de variável randômica de Bernoulli, que assume somente os valores 0 e 1, $E[X_{a,b}]$ é igual à probabilidade de que $X_{a,b}$ seja igual a 1. Nesse caso, para determinar $E[X]$, necessitamos determinar a probabilidade de que dois elementos do conjunto sejam comparados no decorrer do algoritmo. Denote por $C' = \{C'(1), \ldots, C'(n)\}$ a sequência ordenada dos elementos de C. Isto é, C' contém os mesmos elementos de C, porém na ordenação crescente.

O lema seguinte responde a essa questão.

Lema 3.9. *Sejam $C'(a), C'(b)$, dois elementos de C', $1 \leqslant a < b \leqslant n$. Então a probabilidade de que $C'(a), C'(b)$ sejam comparados no algoritmo é $\frac{2}{b-a+1}$.*

Demonstração. Inicialmente, determinamos as condições para que $C'(a), C'(b)$ sejam comparados pelo algoritmo. Mostramos que esta comparação ocorre se e somente se $C'(a)$ ou $C'(b)$ é o *primeiro* elemento a ser escolhido como pivô, entre todos os contidos em $C'(a,b) = \{C'(a), C'(a+1), C'(a+2), \ldots, C'(b)\}$, no decorrer do algoritmo. Para tal, observamos que, se um elemento $C'(c) \notin C'(a,b)$ for escolhido como pivô pelo algoritmo, antes que qualquer elemento de $C'(a,b)$ ou seja, o subconjunto ordenado $C'(a,b)$ se manterá indivisível, sendo incluído integralmente em algum dos dois subconjuntos da partição determinada pelo pivô $C'(c)$. Isto decorre do fato de que todos elementos de $C'(a,b)$ são menores do que $C'(c)$, ou todos são maiores do que este pivô. Por outro lado, se $C'(c) \in C'(a,b)$ e $C'(c) \neq C'(a), C'(b)$, então $C'(a)$ e $C'(b)$ serão alocados em partes distintas da partição determinada por $C'(c)$. Isto impedirá uma futura comparação entre $C'(a)$ e $C'(b)$. Finalmente, se $C'(a)$ ou $C'(b)$ for escolhido como primeiro pivô contido em $C'(a,b)$, então ocorre a comparação entre $C'(a)$ e $C'(b)$ pois todos elementos de $C'(a,b)$ serão comparados com o pivô escolhido.

Consequentemente, tendo em vista que a escolha do pivô é aleatória e que todos os elementos possuem probabilidades idênticas de serem escolhidos,

$$
\begin{aligned}
& P(C'(a), C'(b) \text{ serem comparados}) \\
&= P(C'(a) \text{ ou } C'(b) \text{ é o pivô escolhido em PARTIÇÃO-RAND(C', a, b))} \\
&= P(C'(a) \text{ é o pivô escolhido em PARTIÇÃO-RAND(C', a, b))} + \\
&+ P(C'(b) \text{ é o pivô escolhido em PARTIÇÃO-RAND(C', a, b))} \\
&= \frac{1}{b-a+1} + \frac{1}{b-a+1} = \frac{2}{b-a+1}
\end{aligned}
$$

pois $\{C'(a), C'(a+1), \ldots, C'(b)\}$ contém $b-a+1$ elementos. □

Com este resultado, podemos concluir o cálculo de $E[X]$, como a seguir:

$$
\begin{aligned}
E[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} P(C'(a), C'(b) \text{ serem comparados }) \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b-a+1}
\end{aligned}
$$

Definindo $k = b - a$, obtemos

$$E[X] = \sum_{a=1}^{n-1} \sum_{k=1}^{n-a} \frac{2}{k+1}$$

Isto é,

$$E[X] < \sum_{a=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

Como $1 + \frac{1}{2} + \ldots + \frac{1}{n} = O(\log n)$, obtemos

$$E[X] = \sum_{a=1}^{n-1} O(\log n) = O(n \log n)$$

Portanto, o valor esperado do número total de comparações efetuado pelo algoritmo randomizado é $O(n \log n)$.

Observamos ainda que o algoritmo Quicksort goza de uma propriedade interessante decorrente do aspecto relativo ao conceito de aleatoriedade nos algoritmos.

Conforme foi discutido, uma diferença básica entre algoritmos determinísticos e randomizados é que esses últimos requerem a realização de alguma escolha aleatória em alguns de seus passos. Isto é, a aleatoriedade é intrínseca ao algoritmo, e parte do mesmo. Alternativamente, poderíamos supor um algoritmo determinístico (sem aleatoriedade), mas cujos dados fossem aleatórios. Não se trata, pois, de um algoritmo randomizado. Contudo, considerando que o desempenho do algoritmo possa depender de seus dados, caberia então uma análise probabilística de seu desempenho. Nesse caso, trata-se de *análise probabilística de algoritmos determinísticos*, que será abordada com maior detalhe, no Capítulo 4.

No caso especifico do algoritmo Quicksort, transferindo a aleatoriedade da escolha do pivô para os dados de entrada, ao invés de considerar a entrada como uma sequência fixa S de elementos a serem ordenados, a entrada é interpretada como uma permutação aleatória dos elementos de S. Interessante observar que, no caso do algoritmo Quicksort, a análise do algoritmo randomizado é similar à análise probabilística do algoritmo randomizado, obtendo-se resultados similares. Isto constitui uma propriedade especial do Quicksort e não se estende, de um modo geral, a outros algoritmos.

Exemplo 3.6 Problema das Damas

Descrevemos, agora, um outro exemplo para ilustrar os algoritmos de Las Vegas. Trata-se do assim denominado *Problema das Damas* ou *Problema das Damas Pacíficas*. A aplicação utiliza um tabuleiro do tipo jogo de xadrez, de dimensões $n \times n$ e n damas, que se movimentam no tabuleiro como damas de um jogo de xadrez. Isto é, cada dama alcança toda a linha, a coluna e as diagonais que contêm a sua posição. Nesse caso, qualquer outra peça que esteja em alguma dessas posições do tabuleiro pode ser atacada pela dama, conforme ilustra a Figura 3.4. O Problema das Damas consiste em dispor as n damas no tabuleiro, de modo que nenhuma delas possa ser atacada por qualquer outra. Veja uma das soluções na Figura 3.4.

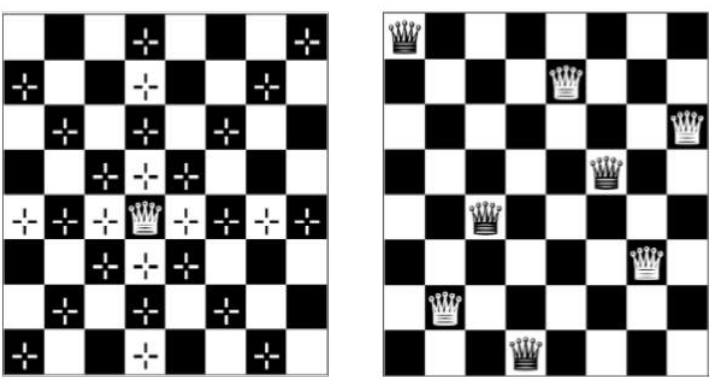


Figura 3.4: Damas Pacíficas: (a) movimentos possíveis e (b) uma solução.

Para efeito de computação descrevemos, inicialmente, um algoritmo determinístico e, posteriormente, um algoritmo randomizado. O algoritmo determinístico utiliza a técnica denominada *retrocesso* (do inglês, *backtracking*), descrita a seguir. O tabuleiro é associado a uma matriz M de dimensões $n \times n$, na qual cada elemento $(i, j), 1 \leqslant i \leqslant j \leqslant n$, representa uma posição do tabuleiro. Supondo algumas damas posicionadas no tabuleiro, a posição $(i, j), 1 \leqslant i, j \leqslant n$ é denominada *dominada* se alguma das damas do tabuleiro contém a linha, coluna ou diagonal da posição (i, j). Caso contrário, a posição é denominada *livre*.

O algoritmo determinístico inicia na linha $i = 1$ e, iterativamente, procura alocar a dama i na coluna j da linha i, que esteja livre. Caso não haja coluna j, da linha i, que esteja livre, o algoritmo retrocede para a linha $i - 1$, remove a dama

$i-1$ da coluna j em que esteja alocada, e procura a posição $(i-1, j')$, $j' > j$, livre, para alocar a dama. Se não houverem posições $(i-1, j')$, o algoritmo retrocede para a linha anterior, removendo a dama posicionada nessa linha, e assim iterativamente. O algoritmo termina quando a dama n for posicionada.

Utilizamos as seguintes estruturas de dados. Cada elemento (i, j) da matriz que representa o tabuleiro corresponde a uma variável booleana, onde (i, j) estará no estado *livre* ou, caso contrário, (i, j) estará *dominado*. As posições nas quais as damas são alocadas em uma solução são representadas pelo vetor SOLUÇÃO(i), $1 \leqslant i \leqslant n$. Nesse caso, SOLUÇÃO$(i) = j$ significa que a dama da linha i se encontra na coluna j.

O algoritmo emprega o procedimento DAMA-R, que procura, para cada linha i, posicionar a dama dessa linha em uma coluna j tal que a posição (i, j) esteja livre.

Para uma utilização mais eficiente de espaço, o algoritmo não considera todas as posições da matriz, simultaneamente. Ao invés disso, o algoritmo armazena, a cada instante, unicamente as informações referentes à linha i corrente. Isto permite reduzir a matriz M a um vetor SOLUÇÃO com n elementos. Ou seja, ocupação de espaço $O(n)$ ao invés de $O(n^2)$.

O procedimento DAMA-R é o seguinte:

Algoritmo 3.10 Problema das Damas - Determinístico (Retrocesso)

```
função DAMA-R(n):
    i, j ← 1, 1
    enquanto i ⩽ n :
        enquanto j ⩽ n e LIVRE(i, j) = F :
            j ← j + 1
        se j ⩽ n então
            SOLUÇÃO(i) ← j
            i, j ← i + 1, 1
        senão
            i, j ← i − 1, SOLUÇÃO(i − 1) + 1
    retornar SOLUÇÃO
```

O procedimento LIVRE(i, j) irá determinar se o elemento (i, j) de M está livre ou não. Para tal, basta verificar se alguma das damas colocadas em uma linha $k < i$ é tal que ela se encontra na coluna j ou em alguma diagonal que passa pelo ponto (i, j). Isto é, o elemento (i, j) estará livre se SOLUÇÃO $\neq j$,

se $k + \text{SOLUÇÃO}(k) \neq i + j$ ou se $k - \text{SOLUÇÃO}(k) \neq i - j$, para todo $k = 1, \ldots, i-1$, conforme o lema seguinte.

Lema 3.10. *Seja o Problema das Damas com uma solução parcial na qual já foram corretamente alocadas as damas nas linhas* $1, \ldots, i-1$. *Então a posição* (i, j) *está livre se e somente se, para todo* $k = 1, \ldots, i-1$,

(i) $SOLUÇÃO(k) \neq j$;

(ii) $k + SOLUÇÃO(k) \neq i + j$;

(iii) $k - SOLUÇÃO(k) \neq i - j$.

A função LIVRE(i, j) está formulado no Algoritmo 3.11.

Algoritmo 3.11 Problema das Damas - Determinação se uma dada posição é livre

função LIVRE(i, j):
  se $i = 1$ **então retornar** V
  se $j > n$ **então retornar** F
  $k \leftarrow i - 1$
  enquanto $k \geqslant 1$ **:**
    se

SOLUÇÃO$(k) = j$ ou
$k + \text{SOLUÇÃO}(k) = i + j$ ou
$k - \text{SOLUÇÃO}(k) = i - j$ **então**
      retornar F
    $k \leftarrow k - 1$
  retornar V

Para avaliar a complexidade do algoritmo, é necessário determinar o número de passos envolvidos para a computação de uma certa linha i, supondo que não haja retrocessos gerados nessa computação. Para a tentativa de colocação da dama no elemento (i, j), é necessário determinar se LIVRE$(i, j) = V$ ou F, isto é, chamar a função LIVRE(i, j), que poderá consumir $O(n)$ passos. Como todas as colunas j da linha i podem ser consideradas, o tempo total para colocar a dama da linha i poderá alcançar até $O(n^2)$ passos. Contudo, a quantidade de retrocessos necessária pode ser alta, o que conduziria esse algoritmo a requerer um número exponencial de passos. É imediato verificar que a complexidade de espaço do algoritmo é $O(n)$.

A correção do método se baseia na observação de que uma tentativa de solução somente é descartada se ela contiver alguma dama em posição que não seja livre.

O algoritmo pode ser modificado, sem dificuldade, de modo a obter todas as soluções do problema e não somente a primeira. Nesse caso, as soluções seriam geradas em ordem lexicográfica crescente.

Um fato interessante a mencionar é o crescimento exponencial do número total de soluções do Problema das Damas. A Tabela 3.1 apresenta as quantidades de soluções totais e únicas (isto é, não obtidas umas das outras por operações de rotação, simetria etc.) em tabuleiros $n \times n$, no intervalo de $1 \leqslant n \leqslant 26$.

n	Total de Soluções	Soluções Únicas
1	1	1
2	0	0
3	0	0
4	2	1
5	10	2
6	4	1
7	40	6
8	92	12
9	352	46
10	724	92
11	2.680	341
12	14.200	1.787
13	73.712	9.233
14	365.596	45.752
15	2.279.184	285.053
16	14.772.512	1.846.955
17	95.815.104	11.977.939
18	666.090.624	83.263.591
19	4.968.057.848	621.012.754
20	39.029.188.884	4.878.666.808
21	314.666.222.712	39.333.324.973
22	2.691.008.701.644	336.376.244.042
23	24.233.937.684.440	3.029.242.658.210
24	227.514.171.973.736	28.439.272.956.934
25	2.207.893.435.808.350	275.986.683.743.434
26	22.317.699.616.364.000	2.789.712.466.510.280

Tabela 3.1: Número de soluções totais e únicas para tabuleiros $n \times n$.

Observe que das 92 soluções para $n = 8$, apenas 12 são únicas. Além disso,

note que, para $n = 26$, o número de soluções é

$$22.317.699.616.364.000$$

enquanto que a quantidade de soluções para $n = 27$ ainda não foi determinada.

Em seguida, apresentamos um algoritmo randomizado de Monte Carlo para o Problema das Damas. O algoritmo randomizado segue a estratégia básica do algoritmo determinístico. Em cada linha i, iniciando por $i = 1$ e progredindo em sequência, o algoritmo escolhe uma posição (i, j) da matriz que esteja livre, e aloca a dama em (i, j). A posição (i, j) é escolhida de forma aleatória, entre todas aquelas que estejam livres na linha i. Caso não haja posição livre nessa linha, devido ao posicionamento das $i - 1$ damas alocadas anteriormente, o algoritmo de Monte Carlo termina e reporta *insucesso*. Assim, não há retrocesso. Ao invés disso, o algoritmo encerra o processo sem obter uma solução.

Desta forma, o algoritmo de Monte Carlo é mais simples do que o determinístico de retrocesso.

Para determinar o conjunto das posições livres de uma certa linha i, utiliza-se a função booleana LIVRE(i, j), formulada no algoritmo determinístico. Como no Algoritmo 3.10 a solução, no algoritmo a seguir, quando encontrada, acha-se armazenada no vetor SOLUÇÃO. As posições livres de cada linha, identificadas por suas colunas, são armazenadas no conjunto CANDIDATOS. A escolha aleatória da posição (i, j), para alocar a dama da linha i, será efetuada por sorteio dentre os pertencentes ao conjunto CANDIDATOS.

O algoritmo consiste do procedimento DAMA-MC, o qual utiliza o procedimento LIVRE(i, j), anteriormente descrito.

O algoritmo descrito retorna uma solução, após a execução bem sucedida de n iterações consecutivas do procedimento DAMA-MC. Em cada uma dessas iterações, a complexidade de tempo é $O(n^2)$, visto que é necessário executar o procedimento LIVRE(i, j), e as demais operações requerem tempo constante. Assim, a complexidade de tempo consumida para a geração de uma solução é $O(n^3)$. A complexidade de espaço é $O(n)$.

Apresentaremos, em seguida, um outro algoritmo randomizado de Monte Carlo, o qual utiliza a mesma ideia de terminar a execução em caso de insucesso sem retrocesso. A diferença básica entre esse novo algoritmo e o anterior, é que utilizamos uma outra maneira de determinar se a posição (i, j) do tabuleiro está livre ou não. Não será empregada a função LIVRE(i, j). De modo alternativo, essa informação será obtida durante a execução do próprio algoritmo randomizado, que resultará em um método mais eficiente.

Algoritmo 3.12 Problema das Damas - Randomizado 1 (Monte Carlo)

```
função DAMA-MC(n):
    para i ← 1,...,n :
        CANDIDATOS ← ∅
        para j ← 1,...,n :
            se LIVRE(i, j) então
                CANDIDATOS ← CANDIDATOS ∪ {j}
        se CANDIDATOS = ∅ então
            retornar INSUCESSO
        senão
            Escolher arbitrariamente j ∈ CANDIDATOS
            SOLUÇÃO(i) ← j
    retornar SOLUÇÃO
```

Assim como no procedimento anterior, o novo método determina se LIVRE$(i, j) = V$ ou F, para $j = 1, 2, \ldots, i$, onde i é a linha corrente do tabuleiro. Por hipótese, todos os valores LIVRE(i, j) já foram determinados para valores menores do que i. Contudo, para calcular se LIVRE$(i, j) = V$ ou F, não será necessário percorrer as linhas do tabuleiro, desde $i - 1$ até 1, como no procedimento anterior. Apenas informações da linha $i - 1$ serão utilizadas.

Definimos 3 vetores P_e, P_c, P_d booleanos, cada qual de dimensão n, descritos a seguir:

$P_e(j) = V$, se há uma dama posicionada na diagonal que sobe à esquerda da posição (i, j) do tabuleiro (isto é, nas posições $(i - k, j - k)$, $k = 1, 2, \ldots$)

$P_c(j) = V$, se há uma dama posicionada na coluna j do tabuleiro

$P_d(j) = V$, se há uma dama posicionada na diagonal que sobe à direita da posição (i, j) do tabuleiro (isto é, nas posições $(i + k, j + k)$, $k = 1, 2, \ldots$)

O valor de cada variável acima será F em caso contrário. Além disso, empregamos também as variáveis booleanas P_e^-, P_c^-, P_d^- as quais correspondem aos valores de P_e, P_c, P_d correspondentes à linha $i - 1$, imediatamente anterior à linha i.

É imediato verificar as seguintes relações entre as variáveis:

Lema 3.11. *Para* $1 < i < n$,

(i) $j > 1$ *e* $P_e^-(j-1) = V \implies P_e(j) = V$
$P_c^-(j) = V \implies P_c(j) = V$
$j < n$ *e* $P_d^-(j+1) = V \implies P_d(j) = V$

(ii) $\textsc{Livre}(i,j) = V \iff (j > 1 \implies P_e^-(j-1) = F)$ *e*
$(P_c^-(j) = F)$ *e*
$(j < n \implies P_d^-(j+1) = F)$

A aplicação iterativa desse lema conduz ao seguinte algoritmo randomizado. Utilizamos novamente o vetor CANDIDATOS para representar os índices das colunas livres para posicionar a dama da linha i.

Algoritmo 3.13 Problema das Damas - Randomizado 2 (Monte Carlo)

```
função DAMA-MC2(n):
    para j ← 1,...,n :
        P_e(j), P_c(j), P_d(j) ← F
    para i ← 1,...,n :
        se i > 1 então
            P_e^-(1..n) ← P_e(1..n)
            P_c^-(1..n) ← P_c(1..n)
            P_d^-(1..n) ← P_d(1..n)
            para j ← 1,...,n :
                se j > 1 então P_e(j) ← P_e^-(j-1)
                P_c(j) ← P_c^-(j)
                se j < n então P_d(j) ← P_d^-(j+1)
        CANDIDATOS ← {j | 1 ⩽ j ⩽ n e P_e(j) = P_c(j) = P_d(j) = F}
        se CANDIDATOS = ∅ então
            retornar INSUCESSO
        senão
            Escolher arbitrariamente j ∈ CANDIDATOS
            SOLUÇÃO(i) ← j
            P_e(j), P_c(j), P_d(j) ← V
    retornar SOLUÇÃO
```

Assim como no algoritmo randomizado anterior, o Algoritmo 3.13 retorna uma solução após realizar n iterações do bloco $i = 1, 2, \ldots, n$. Cada iteração requer $\Theta(n)$ passos. Logo, no caso de sucesso, a complexidade para obter a solução é $O(n^2)$. É também imediato constatar que o espaço necessário é $O(n)$.

As seguintes questões são primordiais para avaliação do Algoritmo 3.13:

1. Qual é a probabilidade de sucesso?
2. Qual é a quantidade esperada de insucessos até ocorrer um sucesso?

A ideia é determinar a probabilidade por meio de um algoritmo que percorre a trajetória do algoritmo randomizado que se deseja avaliar e possa computar a quantidade de terminações bem sucedidas do algoritmo, em relação ao total de tentativas. O algoritmo do cálculo utiliza o procedimento

DAMA-PROB(i, *prob*)

o qual determina a probabilidade do algoritmo de Monte Carlo de gerar uma solução parcial, com damas colocadas em posições específicas nas linhas $1, 2, \ldots, i-1$. O parâmetro *prob* corresponde à probabilidade de geração dessa solução parcial. Essas posições específicas correspondem a escolhas aleatórias realizadas pelo algoritmo, uma para cada linha $1, 2, \ldots, i-1$. Para o cálculo da probabilidade devem ser consideradas as trajetórias resultantes de cada escolha $j \in$ CANDIDATOS durante o algoritmo, e não somente uma escolha específica, como no algoritmo randomizado. Assim, a chamada da função DAMA-PROB($n+1$, *prob*) corresponde à situação em que n damas foram colocadas em posições específicas e *prob* corresponde à probabilidade de que o algoritmo randomizado encontre tal solução. Por outro lado, a probabilidade p de que o algoritmo randomizado termine com sucesso é a soma dos valores de *prob* em chamadas DAMA-PROB($n+1$, *prob*) sobre todas as possíveis configurações de damas nas n linhas.

Será determinado pelo Algoritmo 3.14 a probabilidade p do Algoritmo 3.13 terminar com sucesso. O cálculo para o Algoritmo 3.12 é similar.

O número de passos efetuados pelo Algoritmo 3.14 é elevado para valores moderados de n, pois o algoritmo percorre todas as possíveis soluções. Na realidade, para cada linha i, o algoritmo considera todos os valores possíveis de coluna j que podem conduzir efetivamente a uma solução, isto é, valores de j para os quais a colocação de uma dama na posição (i, j) não conflita com as damas já posicionadas nas linhas $1, 2, \ldots, i-1$. Os demais valores de j são desprezados e não afetam o cálculo da probabilidade de sucesso p. Vale ainda ressalvar que a complexidade de espaço permanece $O(n)$.

A Tabela 3.2 apresenta os resultados obtidos da execução do Algoritmo 3.14 para calcular a probabilidade de sucesso do algoritmo randomizado para n no intervalo $1 \leqslant n \leqslant 14$. Observe que, para $n = 8$, a probabilidade de sucesso é $\approx 0{,}129$.

Algoritmo 3.14 Problema das Damas - Cálculo da Probabilidade de Sucesso

```
função DAMA-CALC-PROB(n):
    p ← 0
    DAMA-PROB(1, 1)
    retornar p

procedimento DAMA-PROB(i, prob):
    se i = n + 1 então
        p ← p + prob
    senão
        se i = 1 então
            P_e(1..n), P_c(1..n), P_d(1..n) ← 0
        senão
            P_e^-(1..n) ← P_e(1..n); P_c^-(1..n) ← P_c(1..n)
            P_d^-(1..n) ← P_d(1..n)
            para j ← 1, ..., n :
                se j > 1 então P_e(j) ← P_e^-(j − 1)
                P_c(j) ← P_c^-(j)
                se j < n então P_d(j) ← P_d^-(j + 1)
        CANDIDATOS ← {j | 1 ⩽ j ⩽ n e P_e(j) = P_c(j) = P_d(j) = 0}
        para todo j ∈ CANDIDATOS :
            P_e(j), P_c(j), P_d(j) ← i
            DAMA-PROB(i + 1, prob/|CANDIDATOS|)
            P_e(j), P_c(j), P_d(j) ← 0
        P_e(1..n) ← P_e^-(1..n); P_c(1..n) ← P_c^-(1..n); P_d(1..n) ← P_d^-(1..n)
        se i > 2 então
            para j ← 1, ..., n :
                se j > 1 então
                    se P_e(j) < i − 1 então P_e^-(j − 1) ← P_e(j)
                    senão  P_e^-(j − 1) ← 0
                se P_c(j) < i − 1 então P_c^-(j) ← P_c(j)
                senão  P_c^-(j) ← 0
                se j < n então
                    se P_d(j) < i − 1 então P_d^-(j + 1) ← P_d(j)
                    senão  P_d^-(j + 1) ← 0
```

n	p
1	1,0
2	0,0
3	0,0
4	0.5
5	0.766666666667
6	0.0694444444444
7	0.249404761905
8	0.129340277778
9	0.130398631932
10	0.0606935074956
11	0.0464757725931
12	0.0465268522376
13	0.0423639236737
14	0.0333462588317

Tabela 3.2: Probabilidade p de sucesso do algoritmo randomizado.

A segunda questão proposta é determinar o número esperado de insucessos do algoritmo randomizado até obter uma solução para o Problema das Damas. Isto é, o valor esperado do número de vezes que o Algoritmo 3.13 deve ser executado até gerar uma solução.

Para responder a essa questão, utilizamos a variável aleatória geométrica. Seja X o número de experimentos realizados até a ocorrência do primeiro sucesso. Sabemos que a probabilidade de sucesso de um dos experimentos é p. O objetivo é determinar o valor esperado $E[X]$ de X. Relembramos do Capítulo 2 que

$$E[X] = \frac{1}{p}$$

Assim, por exemplo, para $n = 8$, o número esperado de execuções do algoritmo randomizado é $\approx 1/0{,}129 \approx 7{,}752$. Isto é, o algoritmo randomizado encontra uma solução por volta da sua 8.ª execução.

Note que, para $n = 14$, a probabilidade de sucesso decresce para ≈ 0.03, que não caracteriza ainda um problema do ponto de vista prático. Nesse caso, o algoritmo randomizado precisaria de apenas cerca de 34 tentativas até obter um sucesso. Naturalmente surge o interesse de consultar os valores de probabilidade para valores a partir de $n = 15$, para verificar como decresce a probabilidade de sucesso em função de n. O tempo de processamento para a computação dessa

n	Tentativas até Sucesso	Damas Colocadas	Tempo Total
8	43	113	0 ms
10	36	102	0 ms
15	480	1359	0 ms
20	72080	199635	144 ms
25	17475	48683	47 ms
30	20275056	56429619	79 s
31	4753166	13186250	19 s
32	31384064	87491425	134 s
33	50704900	139956636	3,86 min
34	826337184	2294605284	1,10 h
35	84888759	235549574	7,133 min
36	8819731671	24531506344	12,97 h
37	364744428	1013013193	33,69 min
38	5724305437	15943342686	9,21 h
39	4091498103	11402835414	6,87 h

Tabela 3.3: Algoritmo de Retrocesso.

probabilidade, no entanto, começa a crescer rapidamente, inviabilizando a tarefa de tabular tais resultados. Isso decorre da natureza do algoritmo de retrocesso, para tal cálculo. Assim, as Tabelas 3.3 e 3.4 ilustram, para diversos valores no intervalo $1 \leqslant n \leqslant 1000$, o número de tentativas necessárias, obtidas empiricamente, até se obter uma solução do problema pelos algoritmos de Retrocesso e de Monte Carlo, respectivamente.

3.5 Conversões entre os algoritmos Las Vegas e Monte Carlo

De um modo geral, é possível transformar um algoritmo de Las Vegas para um certo problema em um algoritmo de Monte Carlo destinado ao mesmo problema, e vice-versa. Isto é, pode-se também transformar um algoritmo de Monte Carlo em Las Vegas. Contudo, no primeiro caso, a determinação da probabilidade de correção do algoritmo de Monte Carlo obtido pode não ser satisfatória. Igualmente, o valor esperado do tempo de execução do algoritmo de Monte Carlo, construído a partir do algoritmo de Las Vegas, pode ser excessivo.

Uma ideia geral para transformar um algoritmo de Las Vegas em Monte Carlo

n	Tentativas até Sucesso	Damas Colocadas	Tempo Total
8	7	49	0,01 ms
20	46	755	0,48 ms
30	91	2285	2,71 ms
50	189	8258	22,52 ms
100	525	47873	0,43 s
200	1504	284344	9,16 s
500	3460	1683633	4,97 min
750	3488	2566623	17,46 min
1000	5317	5237068	1,07 h

Tabela 3.4: Algoritmo de Monte Carlo (média de 1000 experimentos por valor de n).

é estabelecer um tempo máximo limite para a sua execução. Se o algoritmo terminar dentro do limite estabelecido, ele obterá a resposta correta. Caso contrário, a computação será interrompida, ultrapassado o limite, e o algoritmo deve reportar insucesso.

Mais formalmente, considere que $\mathcal{L}$ seja um algoritmo de Las Vegas de decisão com tempo esperado $E[X] = t$, onde X é a variável aleatória correspondendo ao número de passos que o algoritmo requer. Seja $\mathcal{M}$ o algoritmo de Monte Carlo com garantia no "sim" obtido a partir de $\mathcal{L}$ conforme o Algoritmo 3.15. Nele, o algoritmo $\mathcal{L}$ é executado por, no máximo, $t/\epsilon - 1$ passos para alguma constante $0 < \epsilon < 1$.

Algoritmo 3.15 Conversão Las Vegas em Monte Carlo

```
função M(n):
    R ← L(n) (interromper após o limite de t/ε − 1 passos)
    se L(n) foi interrompido então
        retornar "não"
    senão
        retornar R
```

A probabilidade de erro do algoritmo $\mathcal{M}$ de Monte Carlo está limitada a $P(X \geqslant t/\epsilon)$ pois, se $X < t/\epsilon$, o algoritmo fornece a resposta correta, que é aquela de $\mathcal{L}$. Tal probabilidade pode ser limitada usando-se a desigualdade de Markov (Teore-

ma 2.23), como a seguir:

$$P(X \geqslant t/\epsilon) \leqslant \frac{E[X]}{t/\epsilon} = \frac{t}{t/\epsilon} = \epsilon$$

Assim, a probabilidade de erro do algoritmo $\mathcal{M}$ é $\leqslant \epsilon$.

Reciprocamente, uma técnica para transformar um algoritmo de Monte Carlo em Las Vegas é aquela já utilizada no presente capítulo. Isto é, repetir a execução do algoritmo, para um número arbitrário de vezes, até que um sucesso ocorra. Detalharemos a ideia em seguida.

Seja $\mathcal{M}$ um algoritmo de Monte Carlo que produz uma saída correta com probabilidade p. Suponha ainda que exista um algoritmo VERIFICA-SOLUÇÃO que certifica se uma solução fornecida pelo algoritmo de Monte Carlo é, de fato, correta. Seja $\mathcal{L}$ o algoritmo de Las Vegas obtido a partir de $\mathcal{M}$ conforme o Algoritmo 3.16.

Algoritmo 3.16 Conversão Monte Carlo em Las Vegas

```
função L(n):
    repetir
        R ← M(n)
    até que VERIFICA-SOLUÇÃO(R)
    retornar R
```

Sejam $T(n)$ e $t(n)$, respectivamente, as complexidades de tempo dos algoritmos $\mathcal{M}$ e VERIFICA-SOLUÇÃO. Além disso, seja X a variável aleatória que representa o número de iterações do Algoritmo 3.16. Claramente, X é uma variável aleatória geométrica com parâmetro p. Assim, temos que o número esperado de iterações é de

$$E[X] = \frac{1}{p}$$

Em cada iteração, o número de passos gastos é de $T(n) + t(n)$. Assim, o tempo esperado de $\mathcal{L}$ é dado por $(T(n) + t(n))/p$.

3.6 Exercícios

3.1 Dados três polinômios, $F(x)$, $G(x)$ e $H(x)$ escrever um algoritmo de Monte Carlo de erro unilateral para determinar se $F(x)G(x) = H(x)$. Determinar a sua complexidade, bem como a sua probabilidade de erro.

3.2 Dada uma expressão booleana E, na forma normal conjuntiva, com n cláusulas, o problema MAX-SAT consiste em determinar o número máximo de cláusulas de E que podem assumir o valor verdadeiro, dentre todas as atribuições possíveis das variáveis de E. Escrever um algoritmo de Monte Carlo para resolver o problema, de modo que o valor esperado do número máximo fornecido pelo algoritmo seja, pelo menos, a metade do valor máximo obtido, isto é, obtido por um algoritmo exato.

3.3 Descrever um algoritmo determinístico para resolver o problema do Exemplo 3.2 (elemento majoritário), e determinar a sua complexidade.

3.4 Seja C_n um ciclo de n vértices, $n \geqslant 3$. Determinar a probabilidade de que o Algoritmo 3.3, para a determinação do tamanho do corte mínimo de um grafo, obtenha a resposta correta.

3.5 Determinar a probabilidade de que o Algoritmo 3.12 encontre a solução correta do Problema das Damas em um tabuleiro $n \times n$.

3.6 Elabore dois algoritmos para, dado um valor de n, calcular a probabilidade do Algoritmo 3.12 encontrar a solução correta em um tabuleiro $n \times n$. Na elaboração de cada algoritmo, utilize as respectivas ideias:

(i) utilizar técnica análoga àquela empregada no Algoritmo 3.14;

(ii) conduzir execuções do Algoritmo 3.12 e, para cada uma, anotar se a execução foi bem-sucedida.

3.7 Uma *super-dama* em um tabuleiro $n \times n$ possui os movimentos de uma dama, do Problema das Damas, como também os movimentos do cavalo, do jogo de xadrez. Isto é, se uma super-dama está colocada na posição (i, j) do tabuleiro, além das posições alcançadas pela dama convencional, a super-dama também alcança as posições do tabuleiro satisfazendo

$$(i \pm 2, j \pm 1) \text{ e } (i \pm 1, j \pm 2)$$

Em particular, qual é o menor valor de n para o qual existe solução? E para $n = 8$?

3.8 O problema das *Damas Dominantes* consiste em determinar o menor número de damas, em um tabuleiro $n \times n$, tal que todas as posições (i, j) do tabuleiro estejam dominadas, isto é, podem ser alcançadas por uma dama.

Determine o menor número de damas necessárias para que esta condição se verifique. Em particular, quantas damas são necessárias para resolver o problema em um tabuleiro 8×8?

3.9 Provar ou dar contraexemplo:
O número de damas necessárias para resolver o Problema das Damas Dominantes em um tabuleiro $n \times n$ é $\geqslant \lceil n/2 \rceil$.

3.10 O problema das *Damas Agressivas* consiste em determinar o maior número de damas que podem ser posicionadas em um tabuleiro $n \times n$, de modo que qualquer dama possa atacar todas as demais. Em particular, qual o valor para $n = 8$?

3.11 Provar ou dar contraexemplo:
O número de damas em uma solução do Problema das Damas Agressivas se mantém inalterado a partir de certo valor fixo de n.

3.7 Notas Bibliográficas

Os livros de Mitzenmacher e Upfal (2005) e o de Rajeev e Prabhakar (1995) são conhecidos e indicados para a área de algoritmos randomizados. Entre os trabalhos pioneiros de algoritmos randomizados, podemos mencionar os de Berlekamp (1970), Rabin (1976), Gill (1977) e o de Solovay e Strassen (1977). Um artigo que descreve os principais resultados sobre algoritmos randomizados, até então, foi publicado por Karp (1991). Um dos trabalhos considerados pioneiros em algoritmos de Monte Carlo é devido a Curtiss (1953). Deve ser também mencionado o trabalho de Metropolis e Ulam (1949). Um artigo histórico sobre o método de Monte Carlo é o de Metropolis (1987). Os textos em língua portuguesa acerca de algoritmos randomizados, até o momento, são de Figueiredo et al. (2007) e de Martinhon (2002). O trabalho pioneiro em algoritmos de Las Vegas é o de Babai (1979). O método do Quicksort é devido ao Hoare (1962). O algoritmo de corte mínimo é de Karger (1993). Uma versão mais eficiente desse algoritmo é devida a Karger e Stein (1996). O teste de primalidade de Miller–Rabin foi inicialmente proposto por Miller (1976) e posteriormente modificado por Rabin (1980). O Problema das Damas tem sido alvo de interesse desde o século XVIII. Inclusive, Gauss trabalhou na questão da determinação de todas as soluções para o caso de $n = 8$, veja (Campbell 1977). Um método de busca local, capaz de produzir soluções para grandes valores de n, é o do artigo de Sosic e Gu (1994). Outros trabalhos so-

bre o problema, desses mesmos autores, incluem (Sosic e Gu 1990) e (Sosic e Gu 1991). Kalé (1990) desenvolveu uma heurística para resolver o problema. Veja também os artigos de Falkowski e Schmitz (1986) e Reichling (1987). Hoffman, Loessi e R. C. Moore (1969) apresentam uma solução particular para o problema, com complexidade linear. Se forem posicionadas $n' < n$ damas nos tabuleiros $n \times n$, e a questão consistir em decidir se existe uma solução para o problema das n damas, com as n' fixas nas posições originais dadas, então o problema se torna NP-Completo, veja (Gent, Jefferson e Nightingale 2018). O problema da geração de números aleatórios é fundamental para algoritmos randomizados, além de outras questões, envolvendo probabilidades. O livro de Knuth (1997b) é a referência clássica sobre a questão.

4 Análise Probabilística de Algoritmos

4.1 Introdução

Neste capítulo, é estudada a *análise probabilística de algoritmos*. Este estudo é essencial para entender o comportamento de algoritmos que têm o pior caso muito ruim, mas com desempenho prático notável. É também importante para a compreensão da complexidade relativa a partes específicas de alguns algoritmos.

O capítulo se inicia com o conceito formal de *complexidade de caso médio*. Em seguida são apresentadas as análises probabilísticas relativas à complexidade de caso médio para o tratamento de várias estruturas de dados básicas, incluindo Busca Sequencial e Busca Binária em vetores, operações em Tabelas de Dispersão e Árvores Binárias de Busca. É também feita a análise probabilística do Quicksort. Na segunda parte do capítulo, são desenvolvidas análises probabilísticas para aspectos distintos do tempo de execução envolvidos em dois problemas. A primeira análise, relacionada a um problema introduzido sob o nome de portal de preços, visa a determinar o número médio de execuções apenas das partes do algoritmo que envolvem atualizações de dados. A segunda análise está relacionada ao problema clássico de determinar o número de envelopes a serem comprados para completar um álbum de figurinhas, conhecido como o Problema do Colecionador de Figurinhas.

4.2 Limitações da análise de pior caso

Tradicionalmente, analisa-se um algoritmo considerando seu pior caso. Esse enfoque conduziu a resultados primordiais na computação. Em alguns casos, como, por exemplo, o de algoritmos para situações sensíveis, tais como o controle de um voo, é a única abordagem possível.

Entretanto, há um certo número de problemas e algoritmos para os quais a *análise de pior caso* não é uma abordagem prática ou útil. Um dos exemplos é o *método simplex* para programação linear, cuja complexidade de tempo de pior caso é exponencial, mas que, em termos práticos, obtém resultados quase lineares. Outro exemplo de destaque é o Quicksort, conforme apresentado no Capítulo 3, cuja complexidade de pior caso é $O(n^2)$, sendo, na prática, usualmente o algoritmo utilizado, por ser, tipicamente de execução muito rápida, superando os algoritmos de complexidade de pior caso $O(n \log n)$.

Tais situações podem ser explicadas pelo fato de que entradas "ruins" são de ocorrência rara. Os bons resultados, quando existentes, são evidenciados pelo estudo da complexidade "típica" do algoritmo, ou seja, a complexidade de caso médio.

Podemos também utilizar a teoria de probabilidades para analisar outros aspectos de algoritmos que não os tempos de execução. Podemos nos focar apenas na contagem da execução de certas operações básicas realizadas no algoritmo que podem estar associadas a custos específicos. Essas análises são análogas às de caso médio.

4.3 Análises probabilísticas de caso médio

Definindo formalmente, consideremos que uma variável aleatória X possa assumir os valores $\{a_1, a_2, \ldots, a_m\}$, associados aos diferentes tempos de execução de cada entrada possível. A *complexidade de tempo de caso médio* é dada pelo valor esperado:

$$E[X] = \sum_{i=1}^{m} a_i \; P(X = a_i)$$

onde $P(X = a_i)$ é a probabilidade de X assumir o valor a_i.

A seguir, apresentaremos vários exemplos desse enfoque, iniciando com algoritmos para tratamento de estruturas de dados básicas.

4.3.1 Problema: Busca Linear em um vetor com elementos distintos

Seja V um vetor contendo n elementos distintos, sem qualquer organização. O Algoritmo 4.1 mostra uma Busca Linear otimizada nesse vetor.

Algoritmo 4.1 Busca linear em vetor com n elementos distintos

```
função BUSCA(c):
    V[n + 1] ← c
    k ← 1
    enquanto V[k] ≠ c :
        k ← k + 1
    retornar k

Dados: Vetor V com |V| = n, chave x
  t ← BUSCA(x)
```

Observe que o algoritmo utiliza a ideia de colocar o elemento buscado na posição $n+1$. Dessa forma, quando se for buscar um elemento pertencente ao vetor, o retorno da busca será um número no intervalo 1 a n. Em caso contrário, o retorno será $n+1$.

Neste exemplo, mediremos o tempo de execução de forma indireta, considerando a quantidade de vezes que o loop **enquanto** é executado, que corresponde às comparações de elementos de V com a chave procurada c. Para as chaves $V[1]$ a $V[n]$, esses números de comparações são, respectivamente, 1 a n. Para as chaves que não estão em V, o número de comparações é $n+1$. Para o cálculo da complexidade média, temos que supor uma distribuição de probabilidades. Vamos supor que a probabilidade do elemento buscado se encontrar no vetor seja igual a q. Consequentemente, a probabilidade de que a chave buscada não esteja no vetor é $1-q$. Além disso, é razoável supor que as chaves do vetor sejam buscadas de maneira uniforme e, portanto, a probabilidade de que determinada chave seja buscada é igual a q/n. Então, temos:

$$E[X] = \sum_{i=1}^{n} \frac{iq}{n} + (n+1)(1-q) = \frac{(n+1)(2-q)}{2}$$

Essa expressão mostra que, quando $q = 1$, ou seja, num processo em que só se buscam chaves que estão no vetor, $E[X] = (n+1)/2$, o que significa que, em média, o argumento de busca tem que ser comparado com metade do vetor, para ser encontrado. Já se $q = 0$, ou seja, se todas as buscas referem-se a elementos que

não fazem parte do vetor, temos $E[X] = n + 1$. Isso significa que o argumento de pesquisa tem que ser comparado com todo o vetor, para se concluir que houve falha na busca.

4.3.2 Problema: Busca Binária em um vetor ordenado com elementos distintos

Seja V um vetor contendo n elementos distintos, ordenado. O Algoritmo 4.2 mostra uma Busca Binária nesse vetor.

Algoritmo 4.2 Pesquisa binária em vetor ordenado com n elementos distintos

```
função BUSCA(c):
    i ← 1;  j ← n;  k ← 0;
    enquanto i ⩽ j :
        m ← ⌊(i + j)/2⌋;  k ← k + 2;
        se V[m] = c então
            retornar k − 1
        se V[m] < c então
            i ← m + 1
        senão
            j ← m − 1
    retornar k

Dados: Vetor V com |V| = n, chave x
  t ← BUSCA(x)
```

Nesse caso, também contaremos o número de comparações das chaves do vetor com o argumento de pesquisa c, ao invés de tempo de execução, pois eles se relacionam diretamente, como já vimos em outras situações. A variável k conta essas comparações. Enquanto a chave não é encontrada, cada execução do loop **enquanto** conta duas comparações e, quando a chave é encontrada, apenas uma.

Inicialmente, vamos considerar que o processo de busca é tal que somente são buscadas chaves que estejam presentes no vetor. Temos n eventos diferentes, cada um correspondendo à busca de uma das chaves do vetor, com probabilidade $1/n$. A identificação da quantidade de comparações na busca de cada chave será feita com a utilização de uma árvore binária de busca especial, que denominaremos *Árvore de Decisão da Busca Binária*, $T_d(n)$, que ilustra o processo de comparação. A construção de $T_d(n)$ pode ser feita por um processo recursivo espelhado no Algo-

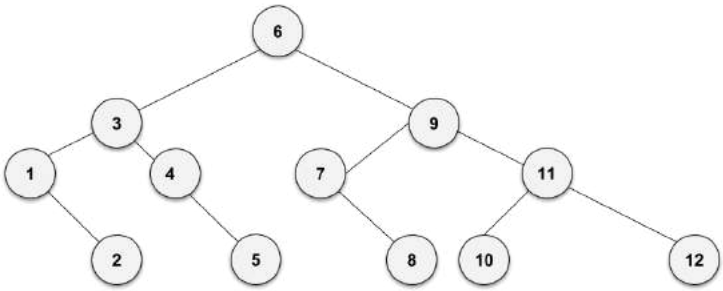


Figura 4.1: Árvore de Decisão da Busca Binária para $n = 12$

ritmo 4.3 cuja ideia é a seguinte: a cada chamada recursiva da função CriaT, que recebe um intervalo $[e, d]$ de índices do vetor V, é criado um nó onde é colocada a chave central desse intervalo, digamos, da posição m. Então recursivamente é criada a subárvore esquerda para o intervalo $[e, m-1]$ e a subárvore direita para o intervalo $[m+1, d]$. O problema trivial é quando o intervalo recebido é nulo, situação em que nada é feito. A chamada inicial passa o intervalo correspondente a todo o vetor. O percurso de busca nessa Árvore Binária de Busca claramente corresponde ao processo da busca binária em um vetor.

Algoritmo 4.3 Criação da Árvore de decisão da pesquisa binária

função CriaT(e, d, p):
  se d > e **então**
    $s \leftarrow$ novo nó; $k \leftarrow \lfloor (e+d)/2 \rfloor$; $s.c \leftarrow V[k]$; $p \leftarrow s$;
    CriaT($e, k-1, s.l$)
    CriaT($k+1, d, s.r$)

Dados: Vetor V com $|V| = n$, árvore T_d
CriaT($1, n, T_d$)

A Figura 4.1 mostra um exemplo da árvore de decisão da pesquisa binária para $n = 12$, supondo que os valores do vetor sejam os inteiros de 1 a 12.

Podemos ver, por exemplo, que a chave 6 é encontrada com 1 comparação, enquanto que as chaves 3 e 9 com 3, ou seja, para encontrarmos o número de comparações para a busca de cada chave, tomamos o dobro do seu nível na árvore e subtraímos 1. Nesse exemplo, a probabilidade de busca de cada chave é $1/12$, e

temos:

$$E[X] = \frac{5+7+3+5+7+1+5+7+3+7+5+7}{12} = \frac{62}{12}$$

O teorema a seguir caracteriza a propriedade básica de $T_d(n)$ ser completa, isto é, suas subárvores nulas estão todas ou no penúltimo ou no último nível da árvore. Além disso, sabe-se que uma árvore binária completa com n nós tem altura $h = \lfloor \log_2 n \rfloor + 1$.

Teorema 4.1. *A Árvore de Decisão da Busca Binária, $T_d(n)$, é uma árvore binária completa.*

Demonstração. A prova é por indução em n. Para $n = 1, 2, 3$ a propriedade é facilmente verificada. Suponhamos $n > 3$. Pela hipótese de indução, as subárvores esquerda e direita de $T_d(n)$ são completas, contendo, respectivamente $\lceil n/2 \rceil - 1$ nós e $\lfloor n/2 \rfloor$ nós. Suas alturas respectivas são $h_e = \lfloor \log_2(\lceil n/2 \rceil - 1) \rfloor + 1$ e $h_d = \lfloor \log_2 \lfloor n/2 \rfloor \rfloor + 1$. Quando n não é potência de 2, satisfaz a $2^k + 1 \leqslant n \leqslant 2^{k+1} - 1$, k inteiro. Temos $2^{k-1} \leqslant \lceil n/2 \rceil - 1, \lfloor n/2 \rfloor < 2^k$. Portanto, $h_e = h_d = k$ e $T_d(n)$ é completa. Se n é da forma $n = 2^k$, k inteiro, $\lceil n/2 \rceil - 1 = 2^{k-1} - 1$ e $\lfloor n/2 \rfloor = 2^{k-1}$. Aqui, temos $h_e = k - 1$ e $h_d = k$, ou seja, alturas diferentes. Mas, nesse caso, a subárvore da esquerda é cheia e todas suas subárvores nulas estão no último nível. Consequentemente, as subárvores nulas de $T_d(n)$ estarão nos seus dois últimos níveis, sendo $T_d(n)$ também completa. □

Como consequência, a Árvore de Decisão da Busca Binária, $T_d(n)$, tem as propriedades dadas pelo seguinte corolário.

Corolário 4.2. *Seja $T_d(n)$ uma árvore de decisão da pesquisa binária em um vetor com n elementos. Então:*

- *A altura de $T_d(n)$ é $h = \lfloor \log_2 n \rfloor + 1$,*
- *O número de nós do nível i é 2^{i-1} se esse não for o último nível,*
- *O número de nós do último nível é $n - (2^{h-1} - 1)$.*

Podemos, agora, calcular o número médio de comparações, para n qualquer, em um processo de Busca Binária, onde os argumentos de busca são sempre valores presentes no vetor. Seja X a variável aleatória que está associada ao número de comparações na busca bem-sucedida dos elementos de V. O número médio de comparações é dado por:

$$
\begin{aligned}
E[X] &= \left(\sum_{i=1}^{h-1} (2i-1)2^{i-1} + (2h-1)(n-(2^{h-1}-1)) \right) \frac{1}{n} \\
&= \left(\sum_{i=1}^{h-1} i\,2^i - \sum_{i=1}^{h-1} 2^{i-1} + 2hn - h2^h + 2h - n + 2^{h-1} - 1 \right) \frac{1}{n} \\
&= \left(\sum_{i=1}^{h-1} \sum_{j=i}^{h-1} 2^i - \sum_{i=1}^{h-1} 2^{i-1} + 2hn - h2^h + 2h - n + 2^{h-1} - 1 \right) \frac{1}{n} \\
&= (h2^h - 2^{h+1} + 2 - 2^{h-1} + 1 + 2hn - h2^h + 2h - n + 2^{h-1} - 1)\frac{1}{n} \\
&= (-2^{h+1} + 2 + 2hn + 2h - n)\frac{1}{n} \\
&= 2h - 1 - \frac{(2^{h+1} - 2h - 2)}{n} \\
&\approx 2\log_2 n
\end{aligned}
$$

pois

$$
2^{h+1} = 2^{\lfloor \log_2 n \rfloor + 2} = 4(2^{\lfloor \log_2 n \rfloor}) \leqslant 4n \Rightarrow \frac{(2^{h+1} - 2h - 2)}{n} < 4
$$

Portanto, no processo de Busca Binária considerado, o número de comparações para a busca de uma chave é aproximadamente o dobro de $\log_2 n$, que é próximo ao de uma busca de pior caso, ou seja, quando a chave procurada estiver em uma posição correspondente a uma folha da árvore mostrada.

Pode ser visto, de forma semelhante à descrita, que, em um processo onde todas as chaves não estão no vetor, o resultado é análogo.

4.3.3 Problema: Tabela de Dispersão com tratamento de colisões por Encadeamento Exterior

Um problema clássico de buscas é o de armazenar, para posterior localização, um conjunto de elementos, cada um identificado por uma chave. Considere U o conjunto de todas as possíveis chaves de interesse. Uma das soluções para esse problema é o uso de Tabelas de Dispersão.

Par criar uma *Tabela de Dispersão*, utilizamos um vetor V, de tamanho m e uma função de dispersão $h(x)$, que transforma cada chave $x \in U$ em um endereço na faixa $0 \ldots m-1$. Essa função atua de maneira uniforme nessa faixa de endereços, isto é, a probabilidade de cada chave x ser associada a cada um dos m endereços é $1/m$. Nesse exemplo, supomos que o tratamento de colisões é por *encadeamento exterior*, significando que, para cada posição r do vetor V, é criada uma lista encadeada L_r onde são colocadas as chaves c_i para as quais $h(c_i) = r$. As chaves de uma mesma lista encadeada são usualmente chamadas de *sinônimos*. Vamos supor que foram inseridas n chaves e que, na inserção de cada chave, ela foi colocada na última posição da sua lista respectiva. O Algoritmo 4.4 ilustra o processo de busca e inserção em uma Tabela de Dispersão, com Encadeamento Exterior. O vetor T contém ponteiros para listas encadeadas, cujo nó contém duas variáveis: chave e prox, um ponteiro da lista encadeada. A função BUSCAINSERÇÃO retorna a quantidade de nós examinados na lista durante a busca de uma chave x.

Algoritmo 4.4 Busca em Tabela de Dispersão com Encadeamento Exterior

```
função BUSCA(c):
    p ← T[h(c)];  k ← 0;
    enquanto p ≠ nulo  e  p.chave ≠ c :
        p ← p.prox
        k ← k + 1
    retornar k
```

Dados: Tabela de Dispersão T com m ponteiros para listas encadeadas L_r
$t \leftarrow$ BUSCA(x)

Definimos como *fator de carga* a razão $\alpha = n/m$. Tanto no processo de inserção quanto no de busca de uma chave c_i, primeiramente calcula-se $h(c_i)$ e, em seguida, percorre-se a lista encadeada correspondente, para inserir ou procurar a chave.

Nesse exemplo, vamos considerar dois problemas:

1. qual o número médio de comparações de chaves em um processo de busca onde todas as chaves da busca não estão na tabela,

2. qual o número médio de comparações de chaves em um processo de busca onde todas as chaves da busca estão na tabela.

A Figura 4.2 mostra um exemplo de uma Tabela de Dispersão com Encadeamento Exterior, com $n = 7, m = 8$, função de espalhamento $h(c) = c \mod 8$ e

um conjunto de chaves $U = \{56, 34, 2, 66, 45, 31, 15\}$.

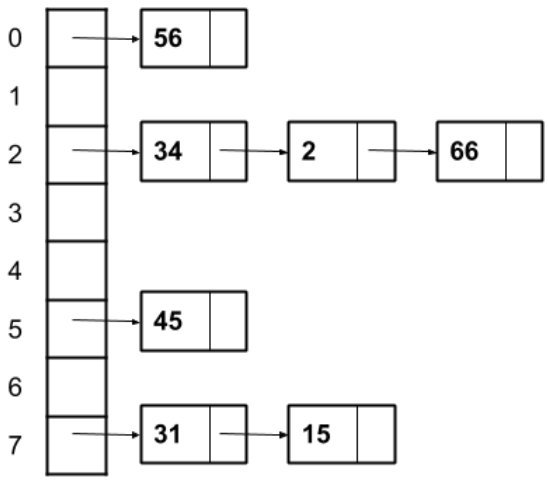


Figura 4.2: Tabela de Dispersão para $n = 7, m = 8$, com Encadeamento Exterior.

Consideremos a solução do primeiro problema para o exemplo, no qual buscamos chaves que não estão na tabela. Neste caso, consideremos 8 eventos, um para cada posição do vetor, com probabilidade uniforme de $1/8$. Para cada uma dessas buscas, temos que comparar a chave buscada com todos os elementos da lista encadeada respectiva. O número esperado de comparações será, então,

$$C = \frac{1+0+3+0+0+1+0+2}{8} = \frac{7}{8}$$

Já para a solução do segundo problema, ou seja, buscas de elementos que estão na tabela, consideremos apenas 7 eventos, um para cada uma das chaves, cada um com probabilidade de $1/7$. Observe que a busca de uma chave que esteja na posição j de sua lista respectiva requer j comparações. Portanto, o valor esperado do número de comparações é:

$$C = \frac{1+2+3+1+1+2}{7} = \frac{10}{7}$$

As generalizações desses casos particulares, que constituem a solução para os problemas 1 e 2, são dadas nos próximos teoremas.

Teorema 4.3. *Numa Tabela de Dispersão com função de dispersão uniforme e encadeamento exterior, o número médio de comparações efetuadas numa busca sem sucesso é igual a* α.

Demonstração. O número de listas encadeadas que podem ser buscadas é m, correspondendo ao tamanho do vetor. Como supomos probabilidades uniformes, a probabilidade da busca em uma determinada lista é $1/m$. Seja L_r a lista encadeada relativa à posição r. A busca mal sucedida relativa a essa posição compara a chave buscada com cada um dos elementos de L_r. Portanto, temos:

$$E[X] = \sum_{r=0}^{m-1} |L_r| \frac{1}{m}$$

Como $\sum_{r=0}^{m-1} |L_r| = n$, segue-se que

$$E[X] = \frac{n}{m} = \alpha$$

□

Teorema 4.4. *Numa Tabela de Dispersão com função de dispersão uniforme e encadeamento exterior, o número médio de comparações efetuadas numa busca com sucesso é igual a*

$$1 + \frac{\alpha}{2} - \frac{\alpha}{2n}$$

Demonstração. Sejam as chaves $c_1, \ldots, c_n$ presentes na Tabela de Dispersão. Temos, por hipótese, que a probabilidade de uma chave estar na lista L_r é $P(h(c_i) = r) = 1/m$, $0 \leqslant r < m$. Dadas as chaves c_i e c_j, então a probabilidade de que ambas estejam na mesma lista L_r $P(h(c_i) = r$ e $h(c_j) = r) = 1/m^2$. Representemos por $X_{i,j}$, $1 \leqslant i < j \leqslant n$ o indicador do evento em que as chaves c_i e c_j estão na mesma lista. Temos:

$$X_{i,j} = \begin{cases} 1, & \text{se } c_i, c_j \text{ estão na mesma lista de sinônimos} \\ 0, & \text{caso contrário} \end{cases}$$

A esperança para esse evento, ou seja, duas chaves estarem em uma mesma lista de sinônimos é

$$E[X_{i,j}] = \sum_{r=0}^{m-1} \frac{1}{m^2} = \frac{1}{m}$$

Sem perda de generalidade, consideremos que a chave c_i foi inserida antes da c_j e que cada chave, ao ser inserida, irá para o final de sua lista de sinônimos.

O número esperado de comparações, na busca de uma chave c_i, é dado pelo número esperado de chaves $c_k, k < i$, que foram incluídas na mesma lista de c_i mais 1, correspondendo à comparação com a própria chave. Dessa forma, para encontrar o número esperado de comparações durante a busca de todas as chaves, tomamos a média sobre as buscas dessas chaves. Portanto, o número de comparações esperado na busca bem-sucedida é dado por:

$$
\begin{aligned}
& E\left[\sum_{i=1}^{n}\left(1+\sum_{j=1}^{i-1} X_{i,j}\right)\frac{1}{n}\right] \\
&= \sum_{i=1}^{n}\left(1+\sum_{j=1}^{i-1} E[X_{i,j}]\right)\frac{1}{n} = \sum_{i=1}^{n}\left(1+\sum_{j=1}^{i-1}\frac{1}{m}\right)\frac{1}{n} \\
&= \sum_{i=1}^{n}\frac{1}{n} + \sum_{i=1}^{n}\sum_{j=1}^{i-1}\frac{1}{mn} = 1 + \sum_{i=1}^{n}\frac{i-1}{mn} \\
&= 1 + \frac{n^2-n}{2mn} = 1 + \frac{n}{2m} - \frac{n}{2mn} \\
&= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}
\end{aligned}
$$

□

Observe que, se tivermos $n = O(m)$, tanto a busca sem sucesso quanto a busca bem-sucedida são realizadas em média, em $O(1)$, ou seja, em tempo constante.

4.3.4 Quicksort

Quando foi apresentado o algoritmo do Quicksort, no Capítulo 3, mostraram-se as análises das complexidades de tempo de pior e melhor caso. Essas complexidades são $O(n^2)$ e $O(n \log n)$, respectivamente. Agora, apresentaremos a análise do comportamento "típico" do algoritmo, ou seja, a complexidade de caso médio. Tal como antes, consideremos como medida o número de comparações de chaves feito durante a ordenação. Em outras palavras, queremos encontrar o número esperado de comparações para a ordenação de um vetor com n elementos, dispostos de forma randômica.

Seja X_n a variável randômica associada ao número de comparações feitas na ordenação de um vetor com n elementos pelo Quicksort. As comparações são todas efetuadas no procedimento PARTIÇÃO, chamado pelo QUICKSORT. Claramente, $X_0 = 0$ e $X_1 = 0$, pois correspondem aos subproblemas triviais e o procedimento QUICKSORT nada faz, nesses casos. Devemos calcular $E[X_n]$ para todo $n \geqslant 2$. Seja L_n o número de comparações na chamada recursiva da ordenação do lado esquerdo da partição e R_n o número relativo do lado direito. Portanto, X_n é igual ao número de comparações para fazer a partição, somado aos valores L_n e R_n, $n \geqslant 2$. É fácil verificar que, quando $n \geqslant 2$, o procedimento PARTIÇÃO efetua $n-1$ comparações entre chaves, pois compara o pivô com as demais chaves.

Para calcularmos o valor esperado de L_n ou R_n, temos que considerar que cada lado da partição do vetor pode ter tamanho de 0 a $n-1$. Observe que, quando um lado da partição tem tamanho i, o outro tem tamanho $n-1-i$. Podemos assumir que cada uma dessas possibilidades ocorre com igual probabilidade, ou seja, $1/n$. Para cada partição, temos que fazer recursivamente a ordenação das duas partes resultantes. Consequentemente,

$$E[L_n] = E[R_n] = \left(\sum_{i=0}^{n-1} E[X_i] \right) \frac{1}{n}$$

Isso nos leva à seguinte recorrência para obter $E[X_n]$:

$$E[X_n] = \begin{cases} 0, \text{ se } n \in \{0, 1\} \\ n - 1 + \left(\sum_{i=0}^{n-1} E[X_i] \right) \frac{2}{n}, \text{ se } n \geqslant 2 \end{cases}$$

A seguir, resolveremos a recorrência. Para maior clareza podemos escrever:

$$E[X_n] = n - 1 + \frac{2(E[X_0] + E[X_1] + E[X_2] + \ldots + E[X_{n-1}])}{n}, \quad n \geqslant 2$$

Multiplicando ambos os lados por n, obtemos:

$$nE[X_n] = n^2 - n + 2(E[X_0] + E[X_1] + \ldots + E[X_{n-1}]), \quad n \geqslant 2 \qquad (1)$$

Aplicando (1) a $n-1$,

$$(n-1)E[X_{n-1}] = (n-1)^2 - (n-1) + 2(E[X_0] + \ldots + E[X_{n-2}]), \quad n \geqslant 3 \qquad (2)$$

Subtraindo (1) − (2) e rearrumando,

$$nE[X_n] = (n+1)E[X_{n-1}] + 2n - 2, \quad n \geqslant 3$$

Dividindo os dois lados por $n(n+1)$,

$$\frac{E[X_n]}{n+1} = \frac{E[X_{n-1}]}{n} + \frac{2}{n+1} - \frac{2}{n(n+1)}, \quad n \geqslant 3$$

Usando iteração para resolver a recorrência:

$$\begin{aligned}\frac{E[X_n]}{n+1} &= \frac{2}{n+1} + \frac{2}{n} - \frac{2}{n(n+1)} - \frac{2}{(n-1)n} + \frac{E[X_{n-2}]}{n-1} \\ &= \ldots \\ &= 2\sum_{i=3}^{n} \frac{1}{i+1} - 2\sum_{i=3}^{n} \frac{1}{i(i+1)} + \frac{E[X_2]}{3}\end{aligned}$$

Como $E[X_0] = E[X_1] = 0$, temos, pela recorrência, $E[X_2] = 1$. Já vimos anteriormente que $\sum_{i=1}^{n} \frac{1}{i} = H_n$. Portanto,

$$\begin{aligned}2\sum_{i=3}^{n} \frac{1}{i+1} &= 2(H_n - H_3 + \frac{1}{n+1}) = 2H_n - 2(1 + \frac{1}{2} + \frac{1}{3}) + \frac{2}{n+1} \\ &= 2H_n - \frac{11}{3} + \frac{2}{n+1}\end{aligned}$$

Por outro lado, a série $\sum_{i=1}^{n} \frac{1}{i(i+1)}$ é uma série telescópica equivalente a $\sum_{i=1}^{n}(\frac{1}{i} - \frac{1}{i+1})$, cujo resultado é $1 - \frac{1}{n+1}$. Então, temos:

$$2\sum_{i=3}^{n} \frac{1}{i(i+1)} = 2(1 - \frac{1}{n+1} - \frac{1}{1\cdot 2} - \frac{1}{2\cdot 3}) = \frac{2}{3} - \frac{2}{n+1}$$

Finalmente,

$$\begin{aligned}E[X_n] &= (n+1)(2H_n - \frac{11}{3} + \frac{2}{n+1} - (\frac{2}{3} - \frac{2}{n+1}) + \frac{1}{3}) \\ &= (n+1)(2H_n + \frac{4}{n+1} - 4) \\ &= 2(n+1)H_n - 4n \\ &\approx 2nH_n \\ &\approx 1.39n \log_2 n\end{aligned}$$

Esse resultado evidencia que o caso médio do QUICKSORT é muito próximo do melhor caso e bem longe do pior. Isso justifica a sua grande utilização na prática.

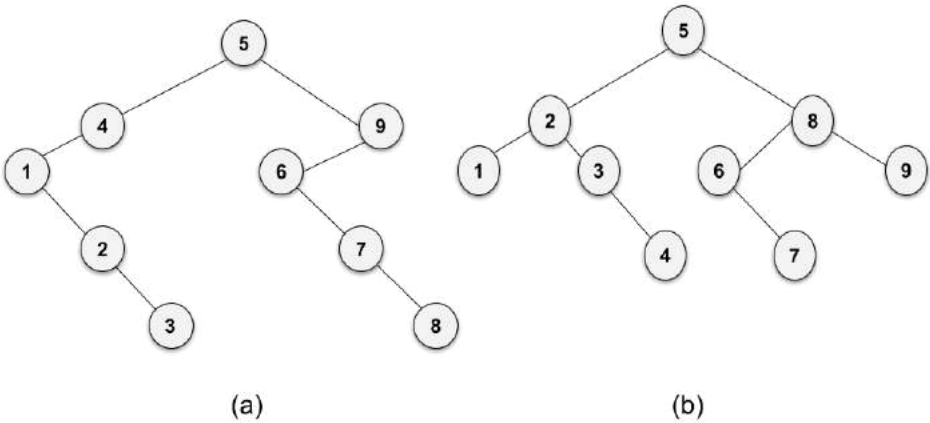


Figura 4.3: Diferentes Árvores Binárias de Busca para as chaves $1, 2, \ldots, 9$

4.3.5 Árvores Binárias de Busca

Uma Árvore Binária de Busca (ABB) utiliza alocação encadeada na sua construção. A estrutura de uma ABB depende da ordem de inserção das chaves na mesma. Por exemplo, se inserirmos as chaves da sequência $\langle\ 5, 4, 9, 6, 1, 2, 7, 8, 3\ \rangle$ nessa ordem descrita, obtemos a árvore mostrada na Figura 4.3(a). Já se as mesmas chaves forem inseridas na ordem da sequência $\langle\ 5, 2, 1, 8, 6, 3, 9, 4, 7\ \rangle$, obteremos a árvore da Figura 4.3(b).

Nesse exemplo, consideremos o processo de busca bem-sucedida na árvore. Estamos interessados no tempo médio de busca, considerando todas as n chaves presentes na ABB. O Algoritmo 4.5 é utilizado na busca de uma chave.

Também usaremos como medição indireta do tempo de busca, o número de comparações feitas entre chaves da árvore e o argumento de busca. Da mesma forma que na Busca Binária, ao se encontrar uma chave a função Busca retorna o dobro do seu nível na árvore subtraído de 1. Na árvore da Figura 4.3(a), o número médio de comparações é:

$$C = \frac{5+7+9+3+1+5+7+9+3}{9} = \frac{49}{9}$$

enquanto que, para a árvore da Figura 4.3(b), temos:

$$C = \frac{5+3+5+7+1+5+7+3+5}{9} = \frac{41}{9}$$

Algoritmo 4.5 Busca em Árvore Binária de Busca

```
função BUSCA(c):
    p ← T;  k ← 1;
    enquanto p ≠ nulo  e  p.chave ≠ c :
        k ← k + 2
        se  p.chave > c  então
            p ← p.le
        senão
            p ← p.ld
    retornar k

Dados: Árvore T com n nós, chave x
  t ← BUSCA(x)
```

Consideremos que o processo de busca é tal que cada uma das n chaves é buscada com a probabilidade uniforme e igual a $1/n$. Além disso, a árvore é criada de maneira aleatória, isto é, usando como ordem de entrada uma permutação qualquer das chaves. Para encontrar o resultado desejado, ou seja, o número médio de comparações nesse processo, associaremos as buscas na árvore com o conceito de *comprimento do caminho interno*, definido como a soma dos tamanhos dos caminhos da raiz até cada um dos nós da árvore. No presente caso, usaremos a variável aleatória Y_n para indicar o comprimento do caminho interno de um ABB com n nós. Por exemplo, para a árvore da Figura 4.3(a), temos $Y_9 = 2+3+4+1+0+2+3+4+1 = 20$.

Associaremos a variável X_n ao número médio de comparações na busca das n chaves da árvore. Pode ser visto que temos a relação $X_n = 2Y_n + n$. Queremos calcular $E[X_n]$ e, para tanto, calcularemos inicialmente, $E[Y_n]$.

Pode ser visto facilmente que $Y_0 = 0$ e $Y_1 = 0$. Seja agora uma árvore T_n com n nós. A raiz dessa árvore pode ser qualquer uma das n chaves. A árvore pode ter como subárvore esquerda uma subárvore de busca com l nós, $0 \leqslant l < n$, e como subárvore direita, uma árvore de busca com $n-l-1$ nós. O comprimento médio do caminho da árvore é a soma dos comprimentos médios das subárvores, somado com $n-1$, pois a distância dos nós da subárvore à raiz principal é acrescido de 1 em relação à distância à raiz da subárvore.

Portanto, $E[Y_n]$ é dado por:

$$E[Y_n] = n - 1 + \left(\sum_{i=0}^{n-1} E[Y_i] + \sum_{i=0}^{n-1} E[Y_{n-i-1}] \right) \frac{1}{n} = n - 1 + 2 \left(\sum_{i=0}^{n-1} E[Y_i] \right) \frac{1}{n}$$

já que as duas somas são iguais. Essa formulação conduz a uma recorrência idêntica àquela feita para o Quicksort e, portanto, tem a mesma solução:

$$E[Y_n] = 2(n+1)H_n - 4n$$

Finalmente,

$$E[X_n] = 2E[Y_n] + n = 4(n+1)H_n - 7n$$

Esse resultado é para a busca de todas as chaves. Para uma chave o valor é

$$\frac{E[X_n]}{n} = \frac{4(n+1)H_n}{n} - 7 \approx 4H_n \approx 2.78 \log_2 n$$

Comparando esse resultado com o da Busca Binária, que corresponde à melhor árvore de busca que podemos construir com n chaves, vemos que o caso médio é muito próximo do melhor caso, sendo da ordem de 1, 4 vezes desse valor.

4.4 Análises probabilísticas especiais

Nesta seção, ilustraremos outros casos de análise probabilística de algoritmos que não sejam a determinação do tempo médio de execução. Iniciaremos por um problema que denominamos *Portal de Preços*.

4.4.1 Portal de Preços

Uma agência de defesa do consumidor mantém, na Internet, um site de preços mínimos para produtos sensíveis. Diariamente essa agência coleta o preço praticado por alguns fornecedores, naquele dia. À medida que as informações são coletadas, elas imediatamente são atualizadas. É mantida uma página especial contendo os preços mínimos do dia. São buscadas informações de n fornecedores e supõem-se que todos os preços são diferentes e aleatórios.

Para cada produto sendo monitorado, o processo de manutenção do preço mínimo pode ser descrito por meio do Algoritmo 4.6. O foco da presente análise está no procedimento ATUALIZA-MÍNIMO, que efetua a atualização do preço mínimo na

página em relação aos preços praticados no dia considerado. Supomos que esse processo seja muito custoso devido a outros procedimentos associados a mudanças de preço. Portanto, deseja-se saber qual o valor esperado do número de vezes que o procedimento ATUALIZA-MÍNIMO será executado.

Observe que a situação do modelo descrito ocorre em muitos algoritmos, nos quais se deseja obter o número médio de execuções de trechos específicos desses algoritmos.

Algoritmo 4.6 Publicação de Preço Mínimo

```
p ← ∞
para i ← 1..n :
    coleta preço f(i) do fornecedor i
    se f(i) < p então
        p ← f(i)
        ATUALIZA-MÍNIMO(p, i)
```

Como estamos supondo que os preços coletados são todos diferentes, podemos associar os diferentes preços aos números de 1 a n, de tal forma que a sequência de preços coletados equivale a uma das $n!$ permutações possíveis, todas com igual probabilidade. Claramente, o pior caso corresponde a todos os preços serem publicados, ou seja, a atualização da página ser realizada n vezes, caso os preços sejam obtidos em ordem crescente.

Associemos a cada preço i o indicador X_i, do evento que indica que o preço i foi publicado, isto é, a função ATUALIZA-MÍNIMO foi executada. Ou seja,

$$X_i = \begin{cases} 1, & \text{se o preço } i \text{ foi publicado} \\ 0, & \text{caso contrário} \end{cases}$$

Associemos a variável aleatória X ao somatório dos indicadores X_i, isto é,

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

Nesses termos, o que estamos procurando calcular é a esperança $E[X]$ que, pela linearidade da esperança, nos leva a:

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

De acordo com a definição de esperança:

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} P(\text{preço } i \text{ ser publicado})$$

O preço i será publicado sempre que ele for menor que todos os $i-1$ precedentes. Como qualquer um dos i primeiros preços pode ser o menor, essa probabilidade é $1/i$. Portanto:

$$E[X] = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} = H_n \approx \ln n$$

Concluímos que o caso médio de atualizações a serem realizadas na página é próximo a $\ln n$, um valor significativamente menor que o pior caso.

4.4.2 Colecionador de Figurinhas

No Problema do *Colecionador de Figurinhas*, um colecionador quer completar um álbum com n figurinhas distintas. A cada compra, o colecionador adquire um envelope contendo uma única figurinha. Há dois problemas de interesse relativos a essa situação.

O primeiro deles está relacionado à questão de saber quantos envelopes o colecionador precisa comprar até completar o álbum. Vários problemas algorítmicos podem ser modelados por esse problema. Por exemplo, em um programa que trata uma Tabela de Dispersão com Encadeamento Exterior, desejamos determinar quantas inserções de dados devem ser feitas para que todas as listas de sinônimos tenham algum elemento. Outro problema que pode ser modelado por um processo idêntico é aquele que envolve a geração de grafos aleatórios por inclusão de arestas, sorteadas iterativamente a partir do grafo vazio. Poderíamos nos interessar pelo número de iterações para se conseguir uma certa densidade de arestas.

A modelagem que usaremos é aquela de preencher um vetor de forma aleatória, por sorteio da posição a ser preenchida. O problema de encontrar o número de sorteios a serem efetuados para preencher completamente o vetor é análogo ao de encontrar o número de pacotes de figurinhas a serem compradas para completar o álbum. O Algoritmo 4.7 esquematiza tal modelagem.

O primeiro problema que trataremos é o de saber qual o número médio de sorteios a serem feitos até que todas as posições do vetor tenham sido escolhidas.

Seja X a variável aleatória que indica o número de sorteios a serem feitos para conseguir escolher todos as posições do vetor. Para todo $1 \leqslant i \leqslant n$, considere que

Algoritmo 4.7 Preenchimento de um vetor por sorteio

```
V[1..n] ← 0
k, t ← 0, 0
repetir
    t ← t + 1
    escolher, aleatoriamente um índice j, 1 ⩽ j ⩽ n
    se V[j] = 0 então
        V[j], k ← 1, k + 1
até que k = n
retornar t
```

há $i-1$ posições ocupadas e seja X_i a variável que indica o número de sorteios para encontrar uma das $n-i+1$ posições ainda não ocupadas. Então

$$X = X_1 + X_2 + \ldots + X_n$$

e, portanto,

$$E[X] = E[X_1] + E[X_2] + \ldots + E[X_n] = \sum_{i=1}^{n} E[X_i]$$

Podemos verificar que as variáveis X_i têm distribuição geométrica com parâmetro de sucesso $p_i = \frac{n-i+1}{n}$ porque, após terem sido sorteadas $i-1$ posições livres do vetor, o sucesso na escolha de uma nova posição livre, em cada sorteio, tem probabilidade igual à fração das posições ainda não sorteadas, que é exatamente $\frac{n-(i-1)}{n}$. A distribuição é geométrica porque a variável está associada à obtenção do primeiro sucesso para a i-ésima escolha. Como se trata de uma distribuição geométrica, temos

$$E[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$$

Consequentemente,

$$\begin{aligned} E[X] &= \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n\left(\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{1}\right) = nH_n \\ &\approx n \ln n \end{aligned}$$

Portanto, a resposta para a primeira questão é que a média procurada é aproximadamente $n \ln n$ tentativas. Para ilustrar esse resultado, para um álbum com 100 figurinhas, o valor esperado do número de pacotes a serem comprados é de aproximadamente 461.

Consideremos, a seguir, uma segunda questão. Suponha que, relativamente ao Problema do Colecionador de Figurinhas, queira-se saber qual o número de figurinhas distintas esperadas, ao se comprar m pacotes. Em termos do problema análogo de preenchimento de um vetor, de forma aleatória, suponha que se queira saber quantas posições distintas do vetor terão sido sorteadas, após m sorteios. O Algoritmo 4.8 ilustra a situação. Nesse algoritmo, a variável k conta as posições distintas escolhidas. Deseja-se saber, após m iterações, qual o valor esperado para essa variável k.

Algoritmo 4.8 Preenchimento limitado de um vetor por sorteio

```
V[1..n] ← 0
k ← 0
para i = 1..m :
    escolher, aleatoriamente um índice j, 1 ⩽ j ⩽ n
    se V[j] = 0 então
        V[j], k ← 1, k + 1
retornar k
```

Embora os dois problemas considerados, nesta seção, estejam relacionados, a análise é de natureza diferente. Para resolver o presente problema vamos associar, à cada posição i do vetor, um indicador do evento de a posição i ter sido escolhida, após m sorteios.

$$X_i = \begin{cases} 1, & \text{se a posição } i \text{ foi escolhida em um dos } m \text{ sorteios} \\ 0, & \text{caso contrário} \end{cases}$$

A probabilidade associada a X_i pode ser obtida como o complemento da probabilidade da posição i não ter sido a escolhida em nenhuma das m tentativas. Em cada sorteio, a probabilidade de cada posição não ser escolhida é uniforme e igual a $\frac{n-1}{n}$. A probabilidade de que a posição i tenha sido escolhida após m iterações é, portanto, $p_i = 1 - (\frac{n-1}{n})^m$, sendo a mesma para todas as posições.

Seja X a variável aleatória que indica o total de posições do vetor sorteadas. Então,

$$X = X_1 + X_2 + \ldots + X_n.$$

Segue-se que

$$E[X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \left(1 - \left(\frac{n-1}{n}\right)^m\right) = n\left(1 - \left(\frac{n-1}{n}\right)^m\right)$$

Para se ter alguma intuição sobre esse resultado, segue um dado numérico. Para $n = m = 1000$, o valor esperado de posições ocupadas no vetor é aproximadamente 632.

4.5 Exercícios

4.1 Considere o seguinte processo de busca em um vetor ordenado V, usando Busca Binária: serão buscadas, com probabilidade uniforme, $n+1$ chaves $x_i, 0 \leqslant i \leqslant n$. As chaves são tais que $x_0 < V[1], V[i] < x_i < V[i+1], 1 \leqslant i < n$ e $x_n > V[n]$. Ou seja, nenhuma chave buscada está no vetor. Calcular o número esperado de comparações nesse processo de busca.

4.2 Verificar a igualdade, para i e h inteiros:

$$\sum_{i=1}^{h-1} i\,2^i = h2^h - 2^{h+1} + 2$$

4.3 Considere a seguinte alteração na busca linear: cada elemento do vetor é uma palavra muito usada de uma língua, onde as palavras mais usadas obedecem à lei de Zipf: a frequência de ocorrência, nessa língua, associada à n-ésima palavra mais usada é k/n onde k é uma constante. As palavras são colocadas no vetor por ordem decrescente de frequência de ocorrência. Suponha que só serão feitas buscas de palavras existentes no vetor e que a probabilidade de que cada palavra seja buscada é proporcional à sua frequência de ocorrência na língua. Calcule o número esperado de comparações para esta situação.

4.4 O algoritmo Algoritmo 4.9 é uma alteração do algoritmo estudado de Busca Binária e pode ser usado para buscas bem-sucedidas. Calcule o número médio de comparações feitas pelo algoritmo.

4.5 O algoritmo Algoritmo 4.10 é usado para ordenação, por inserção, em um vetor V com n elementos distintos. Pergunta-se:

Algoritmo 4.9 Modificação da Pesquisa binária

```
função BUSCA(c):
    i ← 1;  j ← n;  k ← 0;
    enquanto i ⩽ j :
        m ← ⌊(i + j)/2⌋;  k ← k + 2;
        se V[m] < c então
            i ← m + 1
        senão
            se V[m] > c então
                j ← m − 1
            senão
                retornar k
```

Dados: Vetor V com $|V| = n$, chave x
$t \leftarrow$ BUSCA(x)

a) Qual o número esperado de comparações efetuadas na ordenação?

b) Qual o número esperado de trocas efetuadas no vetor durante a ordenação?

Algoritmo 4.10 Ordenação Por Inserção

```
função INSERE( ):
    para i ← 2..n :
        V[0] ← V[i];   j ← i;
        enquanto V[j − 1] > V[j] :
            V[j] ← V[j − 1];   j ← j − 1;
        V[j] ← V[0]
```

Dados: Vetor V com elementos distintos, $|V| = n$
INSERE()

4.6 Recalcule o número médio de comparações em uma busca bem-sucedida numa Tabela com Dispersão com Encadeamento Exterior, se a inserção de uma chave na lista de sinônimos for sempre feita no início da lista.

4.7 Modifique o algoritmo do portal de preços mínimos para atualizar uma página, que contenha preços mínimos e máximos.

4.8 Calcule o número de atualizações esperadas no algoritmo do Exercício 7.

4.9 Considere a seguinte alteração no problema do coletor de figurinhas: suponha que cada pacote contenha k figurinhas aleatórias, ao invés de apenas 1. Recalcule o número médio de pacotes a serem comprados para completar o álbum.

4.10 Considere outra alteração no problema do coletor de figurinhas: suponha que cada pacote contenha k figurinhas aleatórias distintas, ao invés de apenas 1. Recalcule o número médio de pacotes a serem comprados para completar o álbum.

4.11 Escreva um algoritmo randomizado para gerar, de forma aleatória, m arestas para um grafo com n vértices.

4.12 Considere o problema da geração aleatória de arestas para um grafo com n vértices. Suponha que se deseja uma densidade $d = m/n$ de arestas, na qual m é o número de arestas e n o de vértices. Calcule o número de sorteios a serem realizados para conseguir essa densidade.

4.6 Notas Bibliográficas

Os livros de Graham, Knuth e Patashnick (1994), Dobrushkin (2009) e o de Flajolet e Sedgewick (2009) contêm fundamentos matemáticos importantes para a análise probabilística de algoritmos, além daqueles sobre probabilidade citados no Capítulo 2. Os primeiros trabalhos que efetivaram análises mais completas de algoritmos, incluindo estudos de pior caso, melhor caso e caso médio são devidos a Knuth (1972, 1997a,b,c). Nesses trabalhos, especialmente nos três últimos, que constituem a trilogia "The Art of Computer Programming", são analisados probabilisticamente os algoritmos para estruturas fundamentais de dados. Outros livros também desenvolveram tratamento semelhante para essas estruturas de dados, dentre eles, aqueles de Cormen et al. (2009), Gonet e Baeza-Yates (1991) e Flajolet e Sedgewick (2009). Na mesma linha, temos, em português, o livro de Szwarcfiter e Markenzon (2010). O Quicksort é um dos métodos de ordenação mais estudados e foi objeto de inúmeras análises probabilísticas, tais como as feitas por Sedgewick (1977), Mizoi e Osaki (1996) e Alonso et al. (2004). Alguns autores trataram de análises probabilísticas de algoritmos em grafos, tais como Frieze (1990) e Karp, Motwani e Nisan (1993). Inúmeros problemas combinatórios foram objeto de análises como as aqui tratadas. Resultados resumidos para uma ampla gama de

problemas dessa classe são apresentados em (Coffman et al. 1993; Gazmuri 1986; Slominski 1982) e em muitos artigos. Para citar alguns, o Problema do Empacotamento foi tratado por Hofri (1987) e Hong e Leung (1995), o Problema do Caixeiro Viajante, por Karp e Steele (1985) e Frieze e Yukich (2007), o Problema da Mochila por Beier e Vöcking (2004). Em 2001, Spielman e Teng (2001) inventaram a Análise Suavizada de Algoritmos, pela qual ganharam alguns prêmios, inclusive o Prêmio Gödel. Esse tipo de análise combina o melhor da análise de pior caso de algoritmos com a análise de caso médio. Um resultado importante obtido com essa técnica é a explicação da razão pela qual o método simplex, cujo pior caso é exponencial, apresenta excelente desempenho prático. A análise do Problema Portal de Preços feita neste capítulo guarda grande semelhança com aquela feita para o Problema de Recrutamento, por Cormen et al. (2009). O Problema do Colecionador de Figurinhas é clássico em probabilidade, e sua origem é atribuída a de Moivre (1711). Esse problema foi objeto de inúmeros estudos e extensões ao longo dos séculos. Algumas das contribuições recentes são devidas a Holst (1986) e Flajolet, Gardy e Thimonier (1992).

5 Algoritmos de Dados Massivos

5.1 Introdução

Este capítulo dedica-se ao estudo de algoritmos que precisam lidar com uma quantidade muito grande de dados. Para esses algoritmos, a eficiência da complexidade de espaço possui importância crucial.

No estudo clássico da complexidade de algoritmos, um algoritmo é dito *eficiente* se sua complexidade de tempo é $O(n^c)$, onde n representa o tamanho da entrada e c é uma constante não negativa. Como cada célula de memória alocada por um algoritmo é normalmente inicializada, a complexidade de tempo de um algoritmo é maior ou igual àquela de espaço. Assim, se um algoritmo é eficiente com complexidade de tempo $O(n^c)$, sua complexidade de espaço também é $O(n^c)$.

Tal definição de eficiência, na prática, é insuficiente, pois ela permite valores da constante c arbitrariamente grandes. Um algoritmo com complexidade de tempo de $\Theta(n^{100})$ é eficiente pela definição, mas não encontra emprego prático. Para uma aplicação real, espera-se que os valores de c sejam pequenos, não excedendo algumas unidades. Neste capítulo, estudaremos algoritmos para os quais o tamanho da entrada n é muito grande, além dos limites considerados usuais, de modo que as restrições sobre os valores permitidos de c sejam ainda mais severas. A principal dessas restrições será que os dados de entrada não poderão ser

armazenados todos simultaneamente nem na memória principal, nem em memória secundária para posterior consulta. Como consequência, a leitura da entrada poderá ser feita apenas uma única vez, de forma sequencial. Tais algoritmos serão chamados de *algoritmos de dados massivos*.

Estas restrições encontram-se presentes no paradigma de fluxos de dados, apresentado formalmente no início do capítulo. Em seguida, discutimos diversos problemas fundamentais envolvendo a leitura de fluxos de dados, entre eles: a escolha aleatória e uniforme de um elemento, a contagem de elementos especiais do fluxo, determinação do número de elementos distintos, consulta de pertinência de dado valor ao fluxo, frequência dos elementos e semelhança entre conjuntos.

5.2 O Paradigma de Fluxo de Dados

Os algoritmos deste capítulo adotam o chamado *paradigma de fluxo de dados*. Nesse paradigma, pressupõe-se que os dados de entrada são lidos um a um de uma fonte até que não haja novos dados a serem lidos. A sequência de dados $S = a_1, a_2, \ldots, a_n$ produzida por tal fonte é chamada de *fluxo de dados* ou, simplesmente, de *fluxo* (a variável n, daqui em diante, denotará o número de dados em um fluxo, ao invés do tamanho da entrada de um algoritmo como empregado na introdução do capítulo). Cada a_i representa um valor inteiro, real ou de qualquer outro tipo. Concretamente, ele consiste de uma cadeia binária de, no máximo, ℓ bits, onde ℓ é um parâmetro conhecido do fluxo sendo lido. O tamanho exato de cada cadeia binária a_i é denotado por $|a_i|$. Considerando a conversão de números binários em decimais, cada a_i pode também ser visto como um número natural entre 0 e $2^\ell - 1$. Supõe-se que o valor de ℓ seja conhecido antes da leitura dos dados, enquanto o valor de n não necessariamente o seja. Em verdade, o desconhecimento *a priori* do valor de n pelo algoritmo, em alguns casos, é o que torna desafiadora a elaboração do algoritmo sob o paradigma de fluxo de dados, como veremos adiante. Ao término da leitura do fluxo, pressupõe-se que o algoritmo produza a computação de uma saída que dependa dos dados $S = a_1, a_2, \ldots, a_n$, onde a_n foi o último dado lido.

Como exemplo de fluxo, considere o processamento em tempo real da sequência $S = a_1, a_2, \ldots, a_n$ de acessos a um servidor que provê um serviço *online* popular (como, por exemplo, um serviço de vídeos sob demanda), de modo que cada a_i seja a identificação do cliente associado ao i-ésimo acesso. Por identificador do cliente, podemos nos referir ao nome desse usuário na aplicação, seu endereço IP ou outro dado que identifique o cliente naquele contexto. Como exem-

plo de computação ao término da leitura do fluxo, considere aquela de determinar o número distinto de usuários presentes em S. Note que, se um usuário acessou o serviço em dois momentos, então $a_i = a_j$ para algum $i \neq j$. Outro exemplo é determinar o usuário que mais acessou o serviço, entre várias outras perguntas de interesse prático. Embora o momento de realizar a computação desejada possa ser conhecido (por exemplo, na transição de um dia para outro), o valor de n é imprevisível, uma vez que novos acessos podem surgir em um quantidade arbitrária a todo instante. Em outras palavras, o valor de n será desconhecido ao algoritmo. Sendo assim, os dados $a_1, a_2, \ldots, a_n$ devem ser consumidos por um algoritmo cuja finalidade é reportar o resultado tão logo a_n esteja definido, isto é, tão logo esteja caracterizado o fim do fluxo de dados.

O tamanho da entrada de um algoritmo que lê um fluxo $S = a_1, a_2, \ldots, a_n$ com $|a_i| \leqslant \ell$ é $O(\ell n)$. Um *algoritmo para dados massivos* recebe um fluxo como entrada e assume que a quantidade de elementos em tal fluxo seja muito grande. Desse modo, em algoritmos para dados massivos, o valor ℓn é grande o suficiente para tornar inconveniente ou até inviável o armazenamento de todos os elementos de S simultaneamente em memória. No entanto, o algoritmo deve guardar alguma informação relevante a respeito de cada a_i de modo que, ao término da leitura de S, a saída de interesse seja produzida (Figura 5.1). Considera-se que uma complexidade de espaço de

$$O(\ell^c \log^d n) \tag{5.1}$$

para algum par de constantes pequenas não negativas c, d, é viável na prática mesmo para valores grandes de n. Assim, essa será a complexidade esperada dos algoritmos de dados massivos, quando possível.

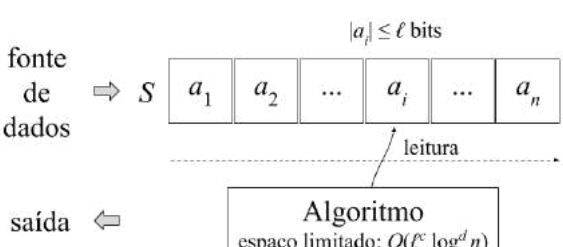


Figura 5.1: Paradigma dos algoritmos para dados massivos.

O Exemplo 1 ilustra um problema para o qual o algoritmo projetado sob o paradigma clássico é adequado também ao paradigma de dados massivos.

Exemplo 1. Dado um fluxo de dados $S = a_1, a_2, \ldots, a_n$ tal que cada $a_i \in \{0, 1, \ldots, 2^\ell - 1\}$, determinar $\sum_{i=1}^{n} a_i$.

Nesse problema, a abordagem clássica é aquela de acumular a soma s dos elementos lidos do fluxo, conforme descrito pelo Algoritmo 5.1. A função *próximo*(S) empregada é uma função especial que retorne o próximo elemento do fluxo caso haja algum ou retorne um elemento especial $\emptyset$ no caso contrário, indicando o fim do fluxo. O algoritmo, tal como apresentado, também é de dados massivos, pois

$$s = \sum_{i=1}^{n} a_i \leqslant \sum_{i=1}^{n} (2^\ell - 1) = 2^\ell n - n = O(2^\ell n)$$

e, portanto, o espaço de memória $|s|$, necessário para armazenar s, é dado por

$$|s| = O(\log\left(2^\ell n\right)) = O(\ell + \log n)$$

o que de fato se encontra dentro do esperado para dados massivos, uma vez que $\ell + \log n = O(\ell \log n)$, conformando-se à expressão (5.1). Adicionalmente, observe também que a hipótese de desconhecimento *a priori* do valor de n não acrescenta dificuldade na solução do problema.

Algoritmo 5.1 Soma de Elementos do Fluxo - Paradigma Clássico e de Dados Massivos

```
função SOMA-ELEMENTOS(S):
    s ← 0; a ← próximo(S)
    enquanto a ≠ ∅ :
        s ← s + a; a ← próximo(S)
    retornar s
```

Em contraste, o Exemplo 2 ilustra um problema de solução trivial sob o paradigma clássico, mas cuja solução no paradigma de dados massivos não é direta. Um algoritmo para esse segundo problema, em conformidade com os requerimentos para ser um algoritmo de dados massivos, será apresentado na Seção 5.3.

Exemplo 2. Dado um fluxo de dados $S = a_1, a_2, \ldots, a_n$, onde n é desconhecido inicialmente, escolher um a_i de forma aleatória e uniforme.

Uma estratégia natural para solucionar o problema seria efetuar uma leitura até o final do fluxo, visando determinar o valor de n. Em seguida, pode-se realizar o

sorteio de um valor i no intervalo $1 \leqslant i \leqslant n$ com distribuição uniforme, retornando o i-ésimo elemento do fluxo como resposta. Para tanto, todos os elementos devem ser armazenados em memória durante a leitura do fluxo. Tal estratégia é empregada na função ESCOLHE-ELEMENTO do Algoritmo 5.2. Observe que a fila nesse algoritmo pode ser implementada por uma lista encadeada, contornando o problema do desconhecimento prévio de n. A complexidade de espaço dessa função é $O(\ell n)$, pois os elementos de fluxo precisam ser armazenados todos em memória.

Algoritmo 5.2 Escolha de um Elemento Aleatória e Uniformemente - Clássico

```
função ESCOLHE-ELEMENTO(S):
    definir fila F
    n ← 0; a ← próximo(S)
    enquanto a ≠ ∅ :
        inserir a em F
        n ← n + 1; a ← próximo(S)
    sortear uniformemente um natural i no intervalo 1 ⩽ i ⩽ n
    para j ← 1 até i :
        remover o próximo elemento de F, guardando-o em a
    retornar a
```

Há uma solução mais eficiente, de espaço $O(\ell + \log n)$, se n for previamente conhecido, conforme mostra a função ESCOLHE-ELEMENTO-N do Algoritmo 5.3. Nela, pode-se efetuar imediatamente o sorteio da posição do elemento escolhido e desprezar o armazenamento de todos os elementos do fluxo exceto aquele de posição i.

Algoritmo 5.3 Escolha de um Elemento Aleatória e Uniformemente - Clássico

```
função ESCOLHE-ELEMENTO-N(n, S):
    sortear uniformemente natural i no intervalo 1 ⩽ i ⩽ n
    para j ← 1 até i :
        a ← próximo(S)
    retornar a
```

Embora a complexidade de ESCOLHE-ELEMENTO-N esteja dentro do esperado

para um algoritmo de dados massivos, esta solução viola o fato de n ser previamente desconhecido. Por outro lado, ESCOLHE-ELEMENTO não comete tal violação, mas não se qualifica para ser de dados massivos por armazenar todos os dados em memória. Há um algoritmo que, sem pressupor o conhecimento de n, resolve o problema com complexidade de espaço também de $O(\ell + \log n)$. Esse algoritmo será detalhado na Seção 5.3.

Nas próximas seções, apresentaremos problemas clássicos de dados massivos. Esses problemas têm caráter fundamental, com diversas aplicações.

5.3 Escolha Aleatória e Uniforme de Elemento

Nesta seção, retornamos à discussão iniciada no Exemplo 2 da Seção 5.1. O problema considerado é: dado um fluxo de dados $S = a_1, a_2, \ldots, a_n$, no qual n é desconhecido, escolher um elemento a_i aleatoriamente e com distribuição uniforme. Vimos que o problema é trivial se n fosse conhecido, com espaço $O(\ell + \log n)$. Porém, deixamos em aberto uma solução de mesma complexidade quando se desconhece n previamente.

A solução encontra-se no Algoritmo 5.4. Para cada a_i lido, procede-se da seguinte maneira. Se $i = 1$, m é inicializado com o valor a_1. Para cada $i > 1$, troque o valor de m para a_i com probabilidade $1/i$ (e, portanto, mantenha o valor corrente de m com probabilidade $(i-1)/i$). Ao fim da leitura de S, o algoritmo reporta o valor de m como o elemento escolhido.

Algoritmo 5.4 Escolha de um Elemento Aleatório e Uniforme - Dados Massivos

```
função ESCOLHE-ELEMENTO(S):
    a ← próximo(S)
    i ← 1; m ← a
    a ← próximo(S)
    enquanto a ≠ ∅ :
        i ← i + 1
        sortear uniformemente real p no intervalo 0 ⩽ p ⩽ 1
        se p ⩽ 1/i então
            m ← a
        a ← próximo(S)
    retornar m
```

A complexidade de espaço do algoritmo é $O(\ell + \log n)$. Resta mostrar que a estratégia acima de fato resolve o problema proposto.

Teorema 5.1. *Seja m o elemento retornado pelo Algoritmo 5.4. Para todo $1 \leqslant i \leqslant n$, a probabilidade que o valor de m seja escolhido como aquele de a_i é de $1/n$.*

Demonstração. Mostraremos por indução em n. Se $n = 1$, é trivial verificar que o algoritmo retorna a_1, elemento que deve ser sorteado de fato. Se $n > 1$, assuma que a estratégia acima esteja correta para todo fluxo de cardinalidade menor que n. Seja S' o fluxo $a_1, a_2, \ldots, a_{n-1}$. Uma execução do algoritmo até a iteração $i = n-1$ no fluxo S pode ser vista como uma execução completa do algoritmo no fluxo S'. Sendo assim, por hipótese de indução, ao fim da iteração na qual $i = n-1$ sob fluxo S, vale que m escolheu qualquer dos elementos de S' com probabilidade $1/(n-1)$. A probabilidade desse elemento ser mantido na iteração $i = n$ é de $\frac{n-1}{n}$. Sendo assim, a probabilidade de que um elemento dentre $a_1, a_2, \ldots, a_{n-1}$ seja escolhido ao final da execução do algoritmo é

$$\left(\frac{1}{n-1}\right)\left(\frac{n-1}{n}\right) = \frac{1}{n}$$

Por outro lado, a probabilidade do elemento a_n ser o escolhido só depende da última iteração, na qual a_n é escolhido com probabilidade $1/n$, completando a prova. □

O Exercício 1 pede a generalização desse algoritmo para o caso em que o conjunto de elementos a serem escolhidos possuir cardinalidade maior do que um.

5.4 Número de Elementos Satisfazendo uma Propriedade

Dado um fluxo $S = a_1, a_2, \ldots, a_n$, considere o problema de contar o número de elementos de S que satisfazem certa propriedade Π. Por exemplo, pode-se desejar a determinação do número k de elementos de S que são pares, ou maior que certa constante, ou que são raízes de certa equação, ou qualquer propriedade arbitrária de interesse. Naturalmente, uma solução é manter uma variável contadora que é incrementada após a leitura de cada a_i se tal elemento satisfizer a propriedade sendo considerada. Veja o Algoritmo 5.5. O espaço utilizado pelo algoritmo consiste no armazenamento da variável k, sob a hipótese de que a verificação se um elemento satisfaz a propriedade Π é feita em espaço constante. Como $k \leqslant n$, o espaço é, portanto, $O(\ell + \log n)$.

Algoritmo 5.5 Contagem de elementos satisfazendo Π (espaço $O(\log n)$)

```
função CONTA-ELEMENTO(S):
    k ← 0; a ← próximo(S)
    enquanto a ≠ ∅ :
        se a satisfaz Π então
            k ← k + 1
        a ← próximo(S)
    retornar k
```

Tal complexidade é aceitável para dados massivos. No entanto, considere o problema em que qualquer uso de espaço $\Omega(\ell + \log n)$ é inaceitável. Tal requerimento pode ser de interesse para aplicações na Internet que servem dezenas de bilhões de vídeos sob demanda por dia. Em um fluxo com todos os acessos a vídeos nesta plataforma, é de interesse a contagem de quantos acessos foram feitos a um determinado vídeo de interesse. Esta contagem é chamada popularmente de número de visualizações daquele vídeo. Ao escolhermos como propriedade Π um determinado acesso corresponder ao vídeo de interesse, a contagem de elementos do fluxo satisfazendo tal propriedade é precisamente o número de visualizações do vídeo em questão. Como tais aplicações mantém o número de visualizações para cada vídeo, temos que uma economia em espaço em cada contador pode representar uma grande economia no total de contadores de visualizações a serem mantidas na aplicação.

Primeiramente, note que o número k de elementos satisfazendo a dada propriedade pode ser qualquer um no intervalo $0 \leqslant k \leqslant n$. Sendo assim, para manter o algoritmo exato, não é possível usar espaço $o(\log n)$ ou, caso contrário, nem todo valor possível de k poderia ser representado. Assim, para atender ao requerimento de usar espaço $o(\ell + \log n)$, devemos relaxar os requerimentos do problema. O Algoritmo 5.6 é uma proposta de solução na qual o número computado de elementos satisfazendo a propriedade é uma estimativa ao invés do valor real. A vantagem é que o espaço empregado pelo algoritmo é de $O(\ell + \log\log n)$, muito menor quando comparado àquele da versão determinística.

De maneira similar ao Algoritmo 5.5, esse algoritmo mantém uma variável contadora z, inicialmente nula. Porém, a cada $a_i \in S$ que satisfaz a propriedade de interesse, z não é incrementada sempre, mas com probabilidade $1/2^z$. Assim, ao final do algoritmo, $z \leqslant k$. Além disso, a igualdade $z = k$ ocorre com baixa probabilidade, pois, para que isto ocorresse, a variável z deveria ser incrementada

com probabilidade 1 a cada novo elemento satisfazendo a propriedade. Ao invés, como foi dito, ocorre com probabilidade de $1/2^z$, que decresce exponencialmente à medida que z incrementa. Intuitivamente, o valor esperado de z ao final do algoritmo parece ser bem menor que k.

O algoritmo retorna como estimativa para k o valor $2^z - 1$, o que a primeira vista parece não razoável. Como qualquer valor de z no intervalo $0 \leqslant z \leqslant k$ é possível, a estimativa $2^z - 1$ é em geral muito maior que k, em particular quando $z = k$. Embora possível, relembramos que $z = k$ ocorre com baixa probabilidade e, portanto, devemos nos preocupar com os valores mais prováveis da estimativa.

Antes de abordarmos a qualidade dessa estimativa formalmente, discutiremos de maneira intuitiva primeiro. Note que, ao final do algoritmo, o retorno $r = 2^z - 1$ é a única saída do algoritmo e toda a informação do fluxo de entrada está perdida. Assim, sejam k o número de elementos de S que satisfazem Π e K a variável aleatória representando o número de elementos satisfazendo Π em fluxos arbitrários. Note que, ao final da execução do algoritmo, é impossível saber o valor exato de k, uma vez que o fluxo está perdido e o seu total não foi computado. Um número infindável de valores são possíveis para k (que é o mesmo que dizer que diversos valores são possíveis para K), embora nem todos ocorram com a mesma probabilidade. Por exemplo, suponha que o retorno do algoritmo tenha sido $r = 1023 = 2^{10} - 1$. Deste modo, sabemos que houve 10 incrementos de z e, por consequência, $k \geqslant 10$, pois cada incremento só é possível pela existência de um elemento associado satisfazendo a propriedade. Por isso, $P(K \leqslant 9 \mid r = 2^{10} - 1) = 0$. Já $K = 10$ é um evento de possível ocorrência considerando o mesmo valor de r, tal que

$$P(K = 10 \mid r = 2^{10} - 1) = \frac{1}{2^0} \cdot \frac{1}{2^1} \cdot \cdots \cdot \frac{1}{2^9} = \frac{1}{2^{0+1+\cdots+9}} = \frac{1}{2^{45}} \qquad (5.2)$$

pois tal evento ocorre se e somente se cada um dos 10 elementos satisfazendo a propriedade produzir um incremento de z. Com este cálculo, fica claro agora o quão baixa é a probabilidade que $z = k$, mesmo para valores pequenos de z. Por outro lado, $P(K = 10^9 \mid r = 2^{10} - 1)$ também parece ter baixa probabilidade, pois, com 10^9 possibilidades de incremento da variável z, seria razoável esperar haver mais incrementos que apenas os 10 que se sucederam. Assim, o valor de maior probabilidade de K é intermediário a tais extremos. A ideia do algoritmo é empregar como estimador para k a esperança de K, isto é, assumir que $k = E[K]$. O Teorema 5.2 prova que $E[K] = r$, justificando o retorno do algoritmo.

Teorema 5.2. *Sejam $S = a_1, a_2, \ldots, a_n$ o fluxo de entrada dado ao Algoritmo 5.6, $r = 2^z - 1$ o valor de retorno do Algoritmo 5.6 e K a variável aleatória represen-*

Algoritmo 5.6 Contagem de elementos satisfazendo Π (espaço esperado $O(\ell + \log\log n)$)

```
função CONTA-ELEMENTO(S):
    z ← 0; a ← próximo(S)
    enquanto a ≠ ∅ :
        se a satisfaz Π então
            sortear uniformemente real p no intervalo 0 ⩽ p ⩽ 1
            se p ⩽ 1/2^z então
                z ← z + 1
        a ← próximo(S)
    retornar 2^z − 1
```

tando o número de elementos que satisfazem Π em fluxos arbitrários. Assumindo $E[K]$ um estimador para k, o número real de elementos satisfazendo a propriedade em S, temos que

$$k = E[K] = r$$

Além disso, a complexidade de espaço esperada do algoritmo é $O(\ell + \log\log n)$.

Demonstração. Seja INCR a variável aleatória de Bernoulli na qual o sucesso corresponde ao incremento de z, quando ocorre a leitura de um a_i que satisfaz a propriedade de interesse Π. Portanto,

$$P(\text{INCR} \mid z = j) = 1/2^j$$

Seja X_j a variável aleatória geométrica que corresponde à ocorrência do primeiro sucesso de INCR a partir do momento em que z se torna igual a j. Em outras palavras, X_j representa o número de elementos que satisfazem a propriedade Π lidos logo após z se tornar igual a j e até que z se torne igual a $j + 1$. Pela expressão do valor esperado de uma variável geométrica,

$$E[X_j] = \frac{1}{P(\text{INCR} \mid z = j)} = \frac{1}{1/2^j} = 2^j$$

Como K é o número de elementos de S satisfazendo a propriedade Π, então

$$K = \sum_{j=0}^{z-1} X_j$$

Logo,

$$\begin{aligned} E[K] &= E\left[\sum_{j=0}^{z-1} X_j\right] \\ &= \sum_{j=0}^{z-1} E[X_j] \\ &= \sum_{j=0}^{z-1} 2^j = 2^z - 1 = r \end{aligned}$$

Pela expressão anterior, temos que $z = \Theta(\log E[K]) = \Theta(\log k) = O(\log n)$. Para armazenar z, é necessário $\Theta(\log z)$ bits, ou seja, espaço $O(\log \log n)$, resultando em um espaço médio de $O(\ell + \log \log n)$. □

5.5 Número de Elementos Distintos

Seja $S = a_1, a_2, \ldots, a_n$ um fluxo, e considere o problema de determinar o número D de elementos distintos de S. Formalmente, o problema é determinar

$$D = |\{a_1, a_2, \ldots, a_n\}|$$

Esse problema é de interesse em muitas aplicações. Como exemplo, se o fluxo consiste do endereço IP de visitantes a determinado *site*, o número de elementos distintos do fluxo corresponde ao número de visitantes únicos ao *site*, uma importante métrica de popularidade do serviço.

O algoritmo exato, que consiste em manter uma estrutura de dados com todos os elementos distintos conhecidos até certo momento, inserindo nessa estrutura de dados o próximo elemento lido do fluxo apenas se ele é distinto daqueles já existentes, seria prático se o número de elementos distintos fosse reduzido. No caso particular em que todos os elementos são distintos, por exemplo, tal abordagem armazenaria todo o fluxo em memória, o que foge do paradigma que estamos considerando para dados massivos. Os algoritmos apresentados a seguir mantêm o espaço sob controle, porém ao custo de produzir uma estimativa aproximada de tal número.

O algoritmo da Contagem Linear

Como vimos, o algoritmo exato de armazenar todos os elementos distintos do fluxo lidos até certo momento não conduz a um algoritmo de dados massivos. Mas prossigamos nessa estratégia, desconsiderando o problema do espaço por enquanto.

A próxima preocupação seria a complexidade de tempo. Um algoritmo que se baseie na ideia de verificar, para cada novo elemento lido do fluxo, se tal elemento pertence ao conjunto de elementos distintos já lidos sendo mantido, pode levar tempo quadrático no tamanho do fluxo caso a estrutura de dados não seja bem escolhida. Uma alternativa seria usar uma tabela de dispersão, da seguinte maneira.

Para algum natural L, defina um vetor de ponteiros para L listas encadeadas $V[1..L]$, inicialmente nulas, e escolha uma função de dispersão

$$h\colon \{0,1,\ldots,2^\ell - 1\} \to \{1,2,\ldots,L\}$$

Para cada elemento a_i lido do fluxo, insira a_i na lista encadeada $V[h(a_i)]$ caso tal elemento não seja encontrado nessa lista. Ao fim da leitura de S, some os tamanhos $|V[i]|$ de todas as listas encadeadas para $1 \leqslant i \leqslant L$. O Algoritmo 5.7 descreve formalmente tal estratégia.

Algoritmo 5.7 Contagem de Elementos Distintos - Paradigma Clássico

função CONTA-ELEMENTOS-DISTINTOS(S):
    seja a função de dispersão $h\colon \{0,1,\ldots,2^\ell - 1\} \to \{1,2,\ldots,L\}$
    defina tabela de dispersão $V[1..L]$ com função de dispersão h
    $a \leftarrow$ *próximo*(S)
    enquanto $a \neq \emptyset$ **:**
        se $V[h(a)]$ não contém a **então**
            insira a em $V[h(a)]$
        $a \leftarrow$ *próximo*(S)
    retornar $\sum_{i=1}^{L} |V[i]|$

Como é conhecido da análise de tabelas de dispersão, a complexidade de tempo de caso médio de cada inserção é de $\Theta(n/L)$, o que se torna tempo constante se $L = \Theta(n)$.

O algoritmo da contagem linear utiliza método análogo, porém sem armazenar os elementos distintos. Ao invés de listas encadeadas, V é um vetor de bits, e a

inserção do elemento a_i apenas marca a posição $V[h(a_i)]$ com um bit 1. Como elementos repetidos marcam sempre a mesma posição, o número de elementos distintos é estimado a partir do número de posições não nulas. Detalharemos essa estratégia em seguida.

Para algum natural L, defina um vetor de bits $V[1..L]$, inicialmente com cada posição de valor nulo e escolha uma função de dispersão $h\colon \{0, 1, \ldots, 2^\ell - 1\} \to \{1, 2, \ldots, L\}$. Para cada a_i lido, defina $V[h(a_i)]$ igual a 1. Ao fim da leitura de S, sejam P o número de posições não nulas de V e D o número de elementos distintos. Se a função h fosse injetiva (isto é, para todo x, y, $h(x) \neq h(y)$ se $x \neq y$), então P seria igual a D. Mas como esse em geral não é o caso, a ideia é obter um estimativa para D em função de P e L.

O Algoritmo 5.8 concretiza tal ideia, estimando D a partir dos valores de P e L. De maneira intermediária, ele computa o valor $\rho = (L-P)/L$, que representa a fração dos elementos nulos de $V[1..L]$. Assim, podemos equivalentemente dizer que D é estimado a partir do valor de ρ.

Algoritmo 5.8 Contagem de Elementos Distintos - Contagem Linear

```
função CONTA-ELEMENTOS-DISTINTOS(S):
    seja a função de dispersão h: {0, 1, ..., 2^ℓ − 1} → {1, 2, ..., L}
    defina vetor de bits V[1..L] associado à função de dispersão h
    V[1..L] ← 0 ; P ← 0
    a ← próximo(S)
    enquanto a ≠ ∅ :
        se V[h(a)] ≠ 1 então
            V[h(a)] ← 1; P ← P + 1
        a ← próximo(S)
    ρ ← (L − P)/L
    retornar  −L ln ρ
```

A ideia desse algoritmo é dada a seguir. Seja X a variável aleatória que representa o número de posições não nulas de $V[1..L]$ ao fim da leitura do fluxo. Tal grandeza é uma variável aleatória, pois depende da função de dispersão h escolhida. Partimos da premissa que a esperança de X possa ser determinada, e tal expressão deverá poder ser escrita como função do número D de elementos distintos do fluxo. De fato, valores maiores para X devem corresponder, com alta probabilidade, a valores maiores de D. Por outro lado, o número P de posições

não nulas é conhecido ao final do algoritmo, pois pode ser medido diretamente a partir da valoração final de $V[1..L]$. Usando como estimador de P a expressão de $E[X]$, podemos obter uma relação entre P e D. Dessa forma, a partir do valor medido de P, determina-se o valor estimado de D. O Teorema 5.3 detalha a estratégia acima, concluindo que $D \approx -L\ln\rho$, o que justifica o retorno do algoritmo.

Teorema 5.3. *Seja X a variável aleatória que representa o número de posições não nulas em $V[1..L]$ ao fim da leitura do fluxo $S = a_1, a_2, \ldots, a_n$ pelo Algoritmo 5.8. Usando como estimador para P o valor de $E[X]$, temos que $D \approx -L\ln\rho$, onde $\rho = (L-P)/L$.*

Demonstração. Seja Y_i a variável de Bernoulli em que $Y_i = 1$ se $V[i] = 1$ ao fim da leitura de S. Como assumimos que cada uma das D chaves distintas pode ser sorteada uniformemente a cada uma das L posições, temos que cada chave pode ser associada a certa posição i com probabilidade $1/L$ e, portanto, de não ser com probabilidade $(L-1)/L$. Logo,

$$P(Y_i = 0) = \left(\frac{L-1}{L}\right)^D$$

Usando o fato que $e^r = \lim_{n\to\infty}(1 + r/n)^n$, temos que

$$\begin{aligned} P(Y_i = 1) &= 1 - P(Y_i = 0) = 1 - \left(\frac{L-1}{L}\right)^D \\ &= 1 - \left(\left(1 + \frac{-1}{L}\right)^L\right)^{D/L} \approx 1 - e^{-D/L} \end{aligned}$$

Como,

$$X = \sum_{i=1}^{L} Y_i$$

temos que

$$E[X] = \sum_{i=1}^{L} E[Y_i] = \sum_{i=1}^{L} P(Y_i = 1) \approx L(1 - e^{-D/L})$$

Estimando P por $E[X]$, resulta em

$$P \approx L(1 - e^{-D/L})$$

Isolando D, obtemos que

$$D \approx L\ln\left(\frac{L}{L-P}\right) = L\ln\rho^{-1} = -L\ln\rho$$

□

A complexidade de espaço do algoritmo é de $\Theta(L)$.

Note que a estimativa de D dada pelo algoritmo é sem utilidade quando $P = L$. Naturalmente, a probabilidade de que P se aproxime de L aumenta conforme n aumenta. Portanto, L deve ser escolhido respeitando alguma relação com n. Mais especificamente, é interessante saber para qual valor de n o valor $E[X]$ será igual a L. Interessantemente, esse é exatamente o problema do colecionador de figurinhas (Seção 4.4.2). Conforme foi visto, se $n \approx L\ln L$, temos que cada posição de V será não nula com alta probabilidade, ou seja, $P = L$ neste caso. Portanto, o algoritmo produzirá seu efeito se o valor de L for convenientemente escolhido de modo que n não se aproxime muito de $L\ln L$.

O algoritmo da Dispersão Mínima

Seja $S = a_1, a_2, \ldots, a_n$ um fluxo, cada $a_i \in S$ tal que $|a_i| \leqslant \ell$, para o qual se deseja computar o número D de elementos distintos. Seja

$$h\colon \{0, 1, \ldots, 2^\ell - 1\} \to [0, 1]$$

uma função de dispersão, isto é, h é uma função de dispersão que mapeia cada valor $a \in \{0, 1, \ldots, 2^\ell - 1\}$ em um real $0 \leqslant h(a) \leqslant 1$. O contradomínio de h é um intervalo contínuo nessa seção por conveniência dos cálculos adiante.

O algoritmo da *dispersão mínima* se baseia na seguinte ideia. Em primeiro lugar, ele computa o valor $h_{\min}$ definido como o mínimo valor da função h sobre todos os elementos de S, isto é,

$$h_{\min} = \min_{1 \leqslant i \leqslant n} \{h(a_i)\}$$

Em seguida, observa-se que o valor de $h_{\min}$ é determinado em função do valor de D, e é independente daquele de n. De fato, como aplicações de h a elementos iguais resultam no mesmo valor de dispersão, a quantidade de resultados distintos da aplicação de h sobre os elementos do fluxo varia com o número de elementos distintos de S. Além disso, quanto maior D, maior a chance de que os valores

resultantes de h sejam variados. Assim, para que $h_{\min}$ seja um valor extremamente baixo, por exemplo, parece ser razoável esperar que haja um número grande de elementos distintos, já que deve ser pequena a chance de que poucos elementos distintos produzam um valor muito baixo de $h_{\min}$. A partir do valor observado de $h_{\min}$, o algoritmo visa a inferir D, da seguinte forma.

Seja H_{MIN} a variável aleatória que representa o mínimo valor de dispersão sobre os elementos de um fluxo arbitrário. Como argumentado, a esperança de H_{MIN}, ao ser determinada, deve envolver uma expressão em função do número D de elementos distintos desse fluxo. Como o valor dessa variável aleatória é conhecida ao fim da execução do algoritmo do valor da variável $h_{\min}$, pode-se estimar D usando a esperança de H_{MIN} como estimador de h_{min}. O Algoritmo 5.9 e o Teorema 5.4 fornecem os detalhes dessa estratégia de estimar D, que suporta o retorno do algoritmo.

Algoritmo 5.9 Contagem de Elementos Distintos - Dispersão Mínima

função CONTA-ELEMENTOS-DISTINTOS(S):
    seja a função de dispersão $h\colon \{0, 1, \ldots, 2^{\ell} - 1\} \to [0, 1]$
    $h_{\min} \leftarrow +\infty$
    $a \leftarrow$ *próximo*(S)
    enquanto $a \neq \emptyset$ **:**
        $h_{\min} \leftarrow \min\{h_{\min}, h(a)\}$
        $a \leftarrow$ *próximo*(S)
    retornar $1/h_{\min} - 1$

Teorema 5.4. *Sejam S um fluxo e h_{min} o valor computado pelo Algoritmo 5.9 associado a S. Seja H_{MIN} a variável aleatória que representa o mínimo valor de dispersão sobre os elementos de um fluxo arbitrário. Usando $E[H_{MIN}]$ como estimador para h_{min}, temos que $D = 1/h_{min} - 1$.*

Demonstração. Seja $S_D = d_1, d_2, \ldots, d_D$ uma sequência dos D elementos distintos de determinado fluxo. Assume-se que a aplicação de h a cada elemento de S_D resulta em um valor real distribuído uniformemente entre 0 e 1. Para cada $1 \leqslant i \leqslant D$, seja X_i a variável aleatória que representa o resultado da aplicação de h ao i-ésimo elemento de S_D, isto é, $X_i = h(d_i)$. Seja H_{MIN} a variável aleatória que representa o mínimo valor de dispersão sobre todos os elementos do fluxo. Logo,

$$H_{\text{MIN}} = \min_{1 \leqslant i \leqslant D} \{X_i\}$$

Note que $P(X_i \leqslant k) = k$, para todo real $0 \leqslant k \leqslant 1$. Como $X_1, X_2, \ldots, X_D$ são independentes e igualmente distribuídas, temos que

$$\begin{aligned} P(H_{\text{MIN}} \leqslant k) &= 1 - P(H_{\text{MIN}} > k) \\ &= 1 - P(X_1, X_2, \ldots, X_D > k) \\ &= 1 - P(X_1 > k)P(X_2 > k) \cdots P(X_D > k) \\ &= 1 - P(X_1 > k)^D \\ &= 1 - (1 - P(X_1 \leqslant k))^D \\ &= 1 - (1-k)^D \end{aligned}$$

Como H_{MIN} é uma variável aleatória contínua, temos que

$$E[H_{\text{MIN}}] = \int_0^1 k f(k) dk$$

onde

$$\begin{aligned} f(k) &= \frac{d}{dk} P(H_{\text{MIN}} \leqslant k) \\ &= \frac{d}{dk}(1 - (1-k)^D) = D(1-k)^{D-1} \end{aligned}$$

Assim, temos que

$$\begin{aligned} E[H_{\text{MIN}}] &= \int_0^1 k f(k) dk \\ &= \int_0^1 k D (1-k)^{D-1} dk \end{aligned}$$

Fazendo-se a integração por partes, isto é, usando o fato que

$$\int_a^b u(k) \left(\frac{d}{dk} v(k) \right) dk = [u(k)v(k)]_a^b - \int_a^b v(k) \left(\frac{d}{dk} u(k) \right) dk$$

e definindo $u(k) = k$, $\frac{d}{dk} v(k) = D(1-k)^{D-1}$, $a = 0$ e $b = 1$ (e, por consequên-

cia, $v(k) = -(1-k)^D$ e $\frac{d}{dk}u(k) = 1$), resulta que

$$\begin{aligned} E[H_{\text{MIN}}] &= \int_0^1 kD(1-k)^{D-1}dk \\ &= \left[-k(1-k)^D\right]_0^1 - \int_0^1 -(1-k)^D dk \\ &= -\left[\frac{(1-k)^{D+1}}{D+1}\right]_0^1 = \frac{1}{D+1} \end{aligned}$$

Como $h_{\min}$ é uma estimativa para $E[H_{\text{MIN}}]$, ou seja

$$h_{\min} = E[H_{\text{MIN}}] = \frac{1}{D+1}$$

obtemos, ao isolar D, a expressão

$$D = \frac{1}{h_{\min}} - 1$$

□

Relembre que o contradomínio de h é o intervalo real fechado $[0, 1]$. Caso fosse empregado uma função de dispersão

$$h\colon \{0, 1, \ldots, 2^\ell - 1\} \to [0, L]$$

para alguma constante $L = O(n)$, é possível adaptar diretamente a prova (Exercício 4) para concluir que

$$D = \frac{L}{h_{\min}} - 1$$

Além disso, é possível mostrar que tal estimativa é uma aproximação do caso discreto, isto é, quando h é uma função de dispersão do tipo

$$h\colon \{0, 1, \ldots, 2^\ell - 1\} \to \{0, 1, \ldots, L\}$$

Nesse caso, como o algoritmo precisa manter apenas o valor $h_{\min}$ sendo computado, usa espaço $\Theta(\log L)$ bits, ou seja, possui complexidade de espaço $O(\log n)$.

O algoritmo *HyperLogLog*

O algoritmo *HyperLogLog* pode ser apresentado como uma extensão do algoritmo da contagem probabilística, que será apresentado primeiro.

O algoritmo da Contagem Probabilística

Seja $S = a_1, a_2, \ldots, a_n$ um fluxo para o qual se deseja computar o número D de elementos distintos, com $|a_i| \leqslant \ell$ para todo $i = 1, 2, \ldots, n$, e L um natural arbitrário. Seja

$$h\colon \{0, \ldots, 2^\ell - 1\} \to \{0,1\}^L$$

uma função de dispersão, ou seja, h é uma função de dispersão que mapeia cada elemento de $\{0, 1, \ldots, 2^\ell - 1\}$ a uma cadeia binária com L dígitos.

O algoritmo da *contagem probabilística*, assim como o algoritmo da dispersão mínima, também utiliza o resultado de uma função de dispersão para estimar o número de elementos distintos do fluxo, porém de uma outra maneira.

Para elaborar a estratégia, é necessário definir a notação $z(x)$, onde x é uma cadeia binária qualquer, que representa o número de dígitos 0 à esquerda da representação binária de x. Por exemplo, $z(110) = 0$, $z(0100) = 1$, $z(001) = 2$ e $z(00000) = 5$.

A estratégia da contagem probabilística é a seguinte. Para cada elemento a_i do fluxo, computa-se a cadeia binária $h(a_i)$ e verifica-se a quantidade de zeros à sua esquerda, isto é, o valor $z(h(a_i))$. Em seguida, computa-se o valor $z_{\max}$ como o máximo valor de $z(h(a_i))$ sobre todos os elementos a_i do fluxo, isto é,

$$z_{\max} = \max_{1 \leqslant i \leqslant n} \{z(h(a_i))\}$$

Note que o valor de $z_{\max}$ é influenciado apenas pelo número D de elementos distintos do fluxo, e não por n. Mais especificamente, quanto maior o valor de D, maior a chance de que $z_{\max}$ atinja valores maiores. Em particular, para que $z_{\max}$ seja um valor extremamente alto, por exemplo, parece ser razoável esperar que haja um número grande de elementos distintos, já que deve ser pequena a chance de que poucos elementos distintos produzam um valor muito alto de $z_{\max}$. A partir do valor observado de $z_{\max}$, portanto, o algoritmo visa inferir D. Tal estimativa é feita calculando-se a quantidade de elementos distintos para que ocorra, com alta probabilidade, uma cadeia binária que comece por $z_{\max}$ 0's. O Algoritmo 5.10 e o Teorema 5.5 fornecem os detalhes dessa ideia.

Algoritmo 5.10 Contagem de Elementos Distintos - Contagem Probabilística

função CONTA-ELEMENTOS-DISTINTOS(S):
seja a função de dispersão $h: \{0, 1, \ldots, 2^\ell - 1\} \to \{0, 1\}^L$
$z_{\max} \leftarrow 0$
$a \leftarrow$ *próximo*(S)
enquanto $a \neq \emptyset$ **:**
$z_{\max} \leftarrow \max\{z_{\max}, z(h(a))\}$
$a \leftarrow$ *próximo*(S)
retornar $2^{z_{\max}}$

Teorema 5.5. *Sejam $S = a_1, a_2, \ldots, a_n$ um fluxo com D elementos distintos e $z_{\max}$ o valor computado pelo algoritmo associado ao fluxo. Seja X a variável aleatória que representa o número de elementos distintos em S. Usando $E[X]$ como estimador de D, temos que $2^{z_{\max}} \leqslant D < 2^{z_{\max}+1}$.*

Demonstração. Seja Y_z a variável aleatória que representa o menor número de elementos distintos de S de modo que o resultado do algoritmo seja pelo menos igual a z. Portanto,

$$E[Y_{z_{\text{máx}}}] \leqslant E[X] < E[Y_{z_{\text{máx}+1}}]$$

Note que Y_z é uma variável aleatória geométrica, cujo valor esperado pode ser determinado por

$$E[Y_z] = \frac{1}{P(E_z)}$$

onde E_z representa o evento de que um elemento $a \in S$ seja tal que $h(a)$ comece por *no mínimo* z zeros. Como cada dígito de $h(a)$ pode ser 0 ou 1 de forma independente dos demais, temos que

$$P(E_z) = \left(\frac{1}{2}\right)^z = \frac{1}{2^z}$$

Assim, temos que

$$E[Y_z] = 2^z$$

Portanto,

$$2^{z_{max}} \leqslant E[X] < 2^{z_{max}+1}$$

□

Primeira extensão: o algoritmo *LogLog*

O algoritmo *LogLog* é uma melhoria daquele da Contagem Probabilística, apresentado na subseção anterior. Note que a variância que se pode esperar do estimador do número D de elementos distintos, usado no Teorema 5.5, é enorme. Com efeito, as estimativas que podem ser feitas pelo algoritmo anterior são sempre potências de 2, que se distanciam uma da outra cada vez mais à medida que tais potências se tornam maiores. A motivação do algoritmo *LogLog* é diminuir tal variância.

Uma primeira ideia, que será melhorada na sequência, é descrita a seguir. Suponha que, ao invés de apenas um estimador D para o número de elementos distintos, sejam usados k estimadores $D_1, D_2, \ldots, D_k$. Por exemplo, tais estimadores podem ser obtidos individualmente tal como D, porém cada um utilizando uma função de dispersão distinta. Assim, se passarmos a usar como estimador de D a média aritmética $\overline{D}_{\mathrm{A}}$ de $D_1, D_2, \ldots, D_k$, a variância já diminui por uma razão de k em relação àquela de D. Com efeito, pela Proposição 2.16, temos que

$$\begin{aligned}\mathrm{Var}(\overline{D}_{\mathrm{A}}) &= \mathrm{Var}\left(\frac{\sum_{i=1}^{k} D_i}{k}\right) = \mathrm{Var}\left(\sum_{i=1}^{k}\frac{1}{k}D_i\right)\\ &= \sum_{i=1}^{k}\frac{1}{k^2}\mathrm{Var}(D_i) = \sum_{i=1}^{k}\frac{1}{k^2}\mathrm{Var}(D) = \frac{\mathrm{Var}(D)}{k}\end{aligned}$$

No entanto, note que para atingir tal objetivo, a eficiência computacional será reduzida, uma vez que k aplicações de funções de dispersão serão executadas, ao invés de apenas uma. Em seguida, apresentaremos novas ideias que melhorarão o estimador em dois aspectos. O primeiro deles diz respeito ao peso de cada estimativa D_i na composição da média. O segundo sobre a eficiência computacional.

Comecemos pelo primeiro aspecto. Note que cada D_i é obtido a partir da expressão 2^{q_i}, para algum natural q_i. Portanto, considere o cenário em que $q_1 = q_2 = \cdots = q_{k-1} = a$, para certa constante a, e $q_k = a + 1$. Todas as estimativas, com exceção de uma, resultam em 2^a. No entanto, a estimativa única de 2^{a+1}, pelo fato de ser igual ao dobro de cada uma das outras, põe um grande peso na média aritmética $\overline{D}_{\mathrm{A}}$. Dessa forma, pequenas variações nas estimativas individuais ainda resultam, mesmo com a média aritmética, em uma estimativa média que se aproxima com maior peso aos valores das maiores estimativas individuais. Assim, basta apenas uma estimativa entre $D_1, D_2, \ldots, D_k$ falhar, produzindo uma estimativa muito maior que o valor real, para que a média $\overline{D}_{\mathrm{A}}$ seja fortemente afetada.

O algoritmo *LogLog* tenta contornar essa fraqueza utilizando uma média alternativa à aritmética. Em seu lugar, o algoritmo utiliza a média geométrica, dada por

$$\overline{D}_{\mathrm{G}} = \sqrt[k]{D_1 D_2 \cdots D_k}$$

Compare a diferença no impacto de valores atípicos em uma série nas duas médias. Seguindo o exemplo anterior, se $D_1 = D_2 = \cdots = D_{k-1} = a$ e $D_k = 2a$, temos que a média aritmética

$$\overline{D}_{\mathrm{A}} = \frac{\left(\sum_{i=1}^{k-1} a\right) + 2a}{k} = \frac{\sum_{i=1}^{k} a}{k} + \frac{a}{k} = a + \frac{a}{k}$$

e verificamos que a média é maior por um fator aditivo de a/k em relação ao que seria se todas as estimativas fossem iguais a a. Tal acréscimo é significativo se $k << a$, o que normalmente é o caso na aplicação de contagem de elementos distintos. Por outro lado, no caso da média geométrica, temos que

$$\overline{D}_{\mathrm{G}} = \sqrt[k]{\left(\prod_{i=1}^{k-1} a\right) 2a} = \sqrt[k]{2} \sqrt[k]{\prod_{i=1}^{k} a} = a \sqrt[k]{2}$$

e a média é maior por um fator multiplicativo de $\sqrt[k]{2}$ em relação ao que seria se todas as estimativas fossem iguais a a. Note que tal fator multiplicativo independe do valor de a, muito próximo de 1 para valores modestos de k. Por exemplo, para $k = 64$, $\sqrt[k]{2} \approx 1{,}0109$, o que equivale a dizer que a média se altera pouco mais de 1% no exemplo anterior, independente da ordem de grandeza de a.

Quanto ao segundo aspecto, sobre a necessidade de computar diversas funções de dispersão, para cada elemento do fluxo, e a consequente preocupação no custo computacional, o algoritmo emprega a seguinte ideia. A cada elemento a_i do fluxo é aplicada apenas uma função h de dispersão. Porém, os elementos são particionados em k partes e cada parte produzirá sua própria estimativa, como se toda a entrada fosse reduzida somente àqueles elementos da parte sendo considerada. A partição e a computação de cada estimativa $D_1, D_2, \ldots, D_k$ são conduzidas da seguinte maneira. Seja a_i um elemento do fluxo. Observa-se o sufixo composto dos b dígitos binários à direita de $h(a_i)$, para alguma constante b escolhida, o qual denotaremos por $s_b(h(a_i))$. A ideia é que elementos a_i, a_j do fluxo sejam agrupados pertencentes a uma mesma parte do fluxo se, e somente se, compartilham do mesmo sufixo associado, isto é, $s_b(h(a_i)) = s_b(h(a_j))$.

Para fazer referência a cada parte da partição dos elementos do fluxo, associaremos um natural a cada valor de $s_b(h(a_i))$, o qual identificará a parte. Note que o valor $s_b(x)$ consiste de uma cadeia binária de b dígitos. Assim sendo, podemos associar um natural a cada parte, pela conversão da cadeia binária de $s_b(x)$ para o decimal correspondente. Para que a numeração das partes comece em 1, tomamos na verdade o natural que sucede aquele produzido pela conversão. Por consequência, se b dígitos mais à direita são empregados para determinar as partes, o tamanho da partição é 2^b, o que implicará em $k = 2^b$ estimadores $D_1, D_2, \ldots, D_k$ distintos.

Dito de outra maneira, o algoritmo funciona da seguinte forma. Em primeiro lugar, escolhe-se $k = 2^b$, para certo natural $b < L$. Para cada elemento a_i lido do fluxo, considera-se o valor $h(a_i)$ no cálculo apenas da estimativa D_{j+1}, onde j é o natural correspondente a $s_b(h(a_i))$. A Figura 5.2 exemplifica a partição da entrada para a produção de $k = 4$ estimativas distintas para um fluxo com 10 elementos. Para cada elemento a_i do fluxo, $h(a_i)$ é uma cadeia com $L = 8$ dígitos binários, dos quais os $b = 2$ últimos são usados para a partição da entrada. Para o exemplo, as estimativas D_1, D_2, D_3, D_4 são calculadas da seguinte forma. No exemplo, note que $h(a_5) = 01011100$. Assim, $z(h(a_5)) = 1$, pois há exatamente um 0 na cadeia binária de $h(a_5)$ antes do 1 mais à esquerda. Além disso, como $k = 4 = 2^b$, então os $b = 2$ dígitos mais à direita 00 de $h(a_5)$ serão utilizados para determinar em qual estimativa será computado o elemento a_5. Como 00 representa o decimal 0, isto significa que a_5 será computado na estimativa $D_{0+1} = D_1$. Analogamente, os elementos a_6 e a_9 também são computados na estimativa para D_1, calculada como $2^{\max\{1,2,2\}} = 4$. As estimativas de D_2, D_3, D_4 são computadas analogamente como aquela de D_1 e são detalhadas na figura. Portanto, a média geométrica da estimativa de elementos distintos dada por cada parte é de $\sqrt[4]{4 \cdot 8 \cdot 2 \cdot 4} = 4$. Note que, nesse exemplo, tal média está subestimando o valor real de elementos distintos, que deve ser no mínimo igual a 8, já que é o número distinto dos valores de dispersão apresentados.

Essa discussão resulta no Algoritmo 5.11. No algoritmo, utilizamos a notação $(x)_{10}$ para representar o decimal correspondente à cadeia binária x.

A expressão final que o algoritmo retorna se justifica da seguinte maneira. Em primeiro lugar, note que, quando se particiona a entrada em k partes, é esperado que cerca de D/k dos D elementos distintos estejam presentes em cada parte. Portanto, cada estimador D_i, dado pelo valor de $2^{z_{\max}[i]}$, está estimando a quantidade

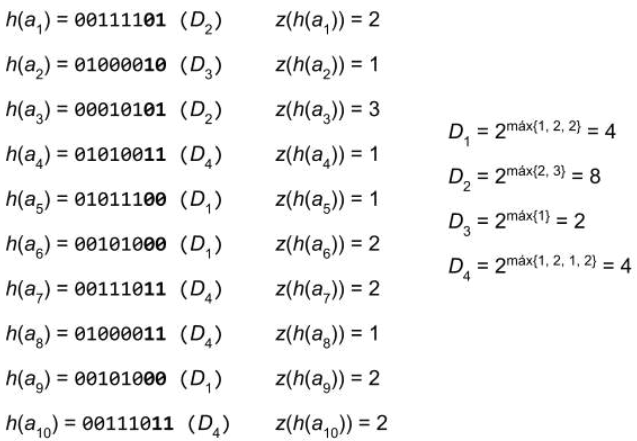


Figura 5.2: Partição da entrada para a produção de $k = 4$ estimativas.

D/k. Utilizando a média geométrica das estimativas, temos que

$$\frac{D}{k} = \sqrt[k]{\prod_{i=1}^{k} D_i} = \sqrt[k]{\prod_{i=1}^{k} 2^{z_{\max}[i]}} = 2^M, \text{ onde } M = \frac{\sum_{i=1}^{k} z_{\max}[i]}{k}$$

e, portanto,

$$D = k2^M$$

Em verdade, a análise anterior possui um viés que pode ser corrigido com a multiplicação de uma constante, que depende exclusivamente do número de partes k. A derivação dessa análise está fora do propósito deste texto por ser demasiadamente técnica. A constante $\alpha = 0{,}79402$ utilizada no algoritmo pode ser empregada para todo $k \geqslant 64$ sem notáveis diferenças do valor real.

Segunda extensão: o algoritmo *HyperLogLog*

O algoritmo *HyperLogLog* é uma melhoria do algoritmo *LogLog* apresentado na subseção anterior basicamente por um único aspecto. Análises subsequentes demostraram que a média harmônica possui resultados melhores do que a geométrica.

Algoritmo 5.11 Algoritmo *LogLog*

função LogLog(S):
  seja a função de dispersão $h\colon \{0,1,\ldots,2^\ell-1\} \to \{0,1\}^L$
  seja $b < L$
  definir vetor $z_{\max}[1..2^b] \leftarrow 0$
  $a \leftarrow$ *próximo*(S)
  enquanto $a \neq \emptyset$ **:**
    $v \leftarrow h(a)$
    $j \leftarrow (s_b(v))_{10} + 1$
    $z_{\max}[j] \leftarrow \max\{z_{\max}[j], z(v)\}$
    $a \leftarrow$ *próximo*(S)
  $M \leftarrow \dfrac{\sum_{i=1}^{k} z_{\max}[i]}{k}$
  retornar $\alpha k 2^M$, onde $\alpha = 0{,}79402$

A média harmônica $\overline{D}_{\mathrm{H}}$ de $D_1, D_2, \ldots, D_k$ é dada por

$$\overline{D}_{\mathrm{H}} = \frac{k}{\dfrac{1}{D_1} + \dfrac{1}{D_2} + \cdots + \dfrac{1}{D_k}}$$

Realizando análise análoga àquela da seção anterior, temos que cada estimativa D_i é referente a cerca de D/k elementos, o que resulta no caso da média harmônica que

$$\frac{D}{k} = \frac{k}{\frac{1}{D_1} + \frac{1}{D_2} + \cdots + \frac{1}{D_k}} = k \left(\sum_{i=1}^{k} 2^{-z_{\max}[i]} \right)^{-1}$$

resultando que

$$D = k^2 \left(\sum_{i=1}^{k} 2^{-z_{\max}[i]} \right)^{-1}$$

A expressão anterior possui também viés que pode ser calculado e, como no caso do algoritmo *LogLog*, pode ser corrigido pela multiplicação de uma constante α_k cuja análise também está fora dos propósitos do texto. A expressão apresentada no Algoritmo 5.12 para α_k serve contanto que $k \geqslant 128$.

Algoritmo 5.12 Algoritmo *HyperLogLog*

função HyperLogLog(S):
  seja a função de dispersão $h\colon \{0,1,\ldots,2^{\ell}-1\} \to \{0,1\}^{L}$
  seja $b < L$
  defina vetor $z_{\max}[1..2^b] \leftarrow 0$
  $a \leftarrow$ *próximo*(S)
  enquanto $a \neq \emptyset$ **:**
    $v \leftarrow h(a)$
    $j \leftarrow (s_b(v))_{10} + 1$
    $z_{\max}[j] \leftarrow \max\{z_{\max}[j], z(v)\}$
    $a \leftarrow$ *próximo*(S)
  retornar $\alpha_k k^2 \left(\sum_{i=1}^{k} 2^{-z_{\max}[i]}\right)^{-1}$, onde $\alpha_k = 0{,}7213/(1 + 1{,}079/k)$

Por fim, é necessário observar que as implementações de *HyperLogLog* também trocam de algoritmos de estimação quando as estimativas se aproximam para valores fora de certo intervalo previsível (isto é, que são muito pequenos ou muito grandes). Tais limites são apontados nos trabalhos originais e estão omitidos no texto.

5.6 Pertinência ao Fluxo

Seja $S = a_1, a_2, \ldots, a_n$ um fluxo, com $|a_i| \leqslant \ell$ para todo $1 \leqslant i \leqslant n$. O problema de *pertinência ao fluxo* S é aquele de determinar se um dado valor s é um elemento de S, isto é, determinar se

$$s = a_i \text{ para algum } 1 \leqslant i \leqslant n$$

Existem inúmeros exemplos nos quais uma solução eficiente para tal problema é de interesse. Note que esse problema consiste em uma das operações fundamentais de estrutura de dados, que é aquela de buscar se um dado elemento foi previamente inserido na estrutura de dados.

Como um exemplo moderno de aplicação, podemos citar o problema de executar um *web crawler*. O *crawler* consiste de uma aplicação que, a partir de uma página *web*, acessa outras páginas *web* através de ponteiros (*links*) contidos nesta página. Para cada página encontrada desta forma, o processo se repete de modo

recursivo. O objetivo do *crawler* é visitar todas as páginas (ou a maior quantidade possível delas) que são encontradas a partir da página inicial em navegação recursiva. Naturalmente, deseja-se que tal busca evite reprocessar páginas anteriormente já visitadas. Assim, para cada página carregada, é necessário descobrir se ela já foi encontrada por uma outra sequência de *links*. Uma navegação deste tipo é conduzida, por exemplo, pelos *sites* de busca, que precisam periodicamente varrer a Internet à procura de novas páginas.

Em termos do problema de pertinência a fluxos, podemos modelar a aplicação da seguinte maneira. A página inicial, cujo endereço é inicialmente dado, é carregada e considerada o elemento a_1 do fluxo S. Note que a_1 representa os bits da página propriamente dita, e não apenas a descrição da URL dessa página. Isto é útil, pois é comum que URLs diferentes carreguem efetivamente o mesmo conteúdo. Para $i \geqslant 1$, cada nova página a_i carregada insere novos elementos ao fluxo. Cada um deles consiste de uma página que é carregada através dos *links* de a_i, lidos em alguma ordem arbitrária. Naturalmente, tais novos elementos não são necessariamente distintos dos anteriores, já que uma página pode ser descoberta por diferentes sequências de navegações a partir da página inicial. Para evitar inspecionar uma página mais de uma vez, o *crawler* precisa decidir, ao carregar uma nova página $s = a_j$ através de um *link* de a_i, com $j > i$, se $s \in \{a_1, a_2, \ldots, a_{j-1}\}$. Se a resposta for afirmativa, os *links* de s são ignorados, uma vez que eles já foram anteriormente considerados. Note que S pode possuir bilhões ou trilhões de elementos, cuja manutenção em memória seria de altíssimo custo.

O filtro de Bloom

Sejam $S = a_1, a_2, \ldots, a_n$ um fluxo, com $|a_i| \leqslant \ell$ para todo $1 \leqslant i \leqslant n$ e s uma dado valor. O problema da pertinência ao fluxo S é determinar se $s \in S$ ou $s \notin S$.

Uma solução eficiente em tempo é a utilização de uma tabela de acesso direto. Nessa solução, definimos um vetor $V[0..2^\ell - 1]$ de bits, inicialmente nulo. Para cada $i = 1, 2, \ldots, n$, após a leitura de $a_i \in S$, inserimos a_i na tabela atribuindo a $V[a_i]$ o valor 1. Dessa forma, para testar se s pertence a S, basta verificar se $V[s] = 1$. Esta ideia é elaborada no Algoritmo 5.13. No algoritmo, a função LEITURA é encarregada de ler todos os elementos do fluxo e preparar a estrutura de dados que será usada pela função PERTINÊNCIA, que efetivamente fará o teste de pertinência dados a estrutura V, previamente preparada, e o elemento s a ser buscado. Naturalmente, a viabilidade dessa solução fica condicionada à disponibilidade de um espaço de 2^ℓ bits.

O filtro de Bloom é uma estrutura que resolve o problema de pertinência quando

Algoritmo 5.13 Pertinência a Conjuntos - Paradigma Clássico

```
função LEITURA(S):
    defina o vetor V[0..2^ℓ − 1] ← 0
    a ← próximo(S)
    enquanto a ≠ ∅ :
        V[a] ← 1
        a ← próximo(S)
    retornar V

função PERTINÊNCIA(V, s):
    retornar V[s] = 1
```

não se dispõe de 2^ℓ bits de espaço. Ele é empregado quando pode-se dispor de αn bits em memória, para alguma constante α não muito alta, e $n << 2^\ell$. Como exemplo, considere a aplicação de *web crawler* descrita na introdução. Naquela aplicação, o valor de n pode ser considerado da ordem dos bilhões. Para um conjunto de servidores, não é difícil armazenar algumas centenas de bilhões de bits, o que seria espaço da ordem de $100n$. No entanto, como cada a_i consiste de uma página *web*, o valor de ℓ seria o tamanho máximo que uma página pode ter em bits. Tal tamanho já é, como problema inicial, impossível de determinar. Mesmo aplicando um limite máximo de 13 KB $\approx$ 100.000 bits para o tamanho das páginas, o que provavelmente estaria abaixo do tamanho real de diversas páginas, a solução por tabela de acesso direto requer espaço de 2^{100000} bits, o que é impossível.

A ideia do filtro de Bloom é a criação de um vetor $V[1..L]$ de bits (chamado de *filtro*), para certa constante $L = O(n)$. Associadas ao filtro, estão k funções de dispersão

$$h_1, h_2, \ldots, h_k : \{0, 1, \ldots, 2^\ell - 1\} \to \{1, 2, \ldots, L\}$$

independentes duas a duas. O filtro é inicialmente nulo. Para cada $a_i \in S$, inserimos a_i ao filtro, o que é feito através da atribuição a $V[h_j(a_i)]$ do valor 1 para todo $1 \leqslant j \leqslant k$. Para verificar a pertinência de certo dado s a S, o algoritmo responde SIM se $V[h_j(s)] = 1$ para todo $1 \leqslant j \leqslant k$ ou, caso contrário, responde NÃO. O Algoritmo 5.14 corresponde a essa ideia geral.

Por construção, se $V[h_j(s)] = 0$ para algum $1 \leqslant j \leqslant k$, então claramente $s \notin S$ e a resposta NÃO é correta. No entanto, se $V[h_j(s)] = 1$ para todo $1 \leqslant j \leqslant k$, a resposta SIM pode estar equivocada. É possível que parte dos bits 1 encontrados foi devida à presença de certo $a_p \in S$, enquanto a outra parte foi devida à presença

Algoritmo 5.14 Pertinência a Conjuntos - Filtro de Bloom

```
função LEITURA(S):
    sejam funções de dispersão h_1, h_2, ..., h_k : {0, 1, ..., 2^ℓ − 1} → {1, 2, ..., L}
    seja L = αn, para alguma constante α não muito alta
    defina vetor V[1..L] ← 0
    a ← próximo(S)
    enquanto a ≠ ∅ :
        para j ← 1 até k :
            V[h_j(a)] ← 1
        a ← próximo(S)
    retornar V

função PERTINÊNCIA(V, s):
    para j ← 1 até k :
        se V[h_j(s)] = 0 então
            retornar NÃO
    retornar SIM
```

de outros $a_q \in S$, de modo que $V[h_j(s)] = 1$ para todo $1 \leqslant j \leqslant k$ mesmo que $s \notin S$. Portanto, o filtro de Bloom tem como característica a possibilidade de produzir falsos positivos a consultas de pertinência, mas nunca falsos negativos. O Teorema 5.6 determina a probabilidade da ocorrência de um falso positivo.

Teorema 5.6. *Seja $S = a_1, a_2, \ldots, a_n$ um fluxo. Considere o filtro do Bloom associado às funções de dispersão $h_1, h_2, \ldots, h_k$ independentes duas a duas com imagem no conjunto $\{1, 2, \ldots, L\}$. Seja FALSOPOSITIVO o evento em que, dado um valor $s \notin S$, o algoritmo responde SIM, isto é, $V[h_j(s)] = 1$ para todo $1 \leqslant j \leqslant k$. Então,*

(i) $P(\text{FALSOPOSITIVO}) \approx (1 - \mathrm{e}^{-nk/L})^k$

(ii) mantendo-se fixa a razão n/L, a probabilidade do item (i) é minimizada para

$$k = \frac{L\ln(2)}{n}$$

correspondendo a

$$P(\text{FALSOPOSITIVO}) \approx 0{,}6185^{L/n}$$

Demonstração. Sob a hipótese de que as funções de dispersão são independentes,

$$\begin{aligned} P(\text{FalsoPositivo}) &= P(V[h_j(s)] = 1 \text{ para todo } 1 \leqslant j \leqslant k) \\ &= P(V[h_1(s)] = 1)P(V[h_2(s)] = 1) \cdots P(V[h_k(s)] = 1) \\ &= P(V[1] = 1)^k \end{aligned}$$

sendo que a última igualdade decorre da independência e uniformidade das funções. Note que, para certo $1 \leqslant j \leqslant k$ e $1 \leqslant i \leqslant n$, temos que

$$P(h_j(a_i) \neq 1) = \frac{L-1}{L} = 1 - \frac{1}{L}$$

Como para que certa posição p seja tal que $V[p] = 0$ é necessário que $h_j(a_i) \neq p$ para todo $1 \leqslant j \leqslant k$, $1 \leqslant i \leqslant n$, temos que

$$\begin{aligned} P(V[1] = 1) &= 1 - P(V[1] = 0) \\ &= 1 - P(h_j(a_i) \neq 1 \text{ para todo } 1 \leqslant j \leqslant k, 1 \leqslant i \leqslant n) \\ &= 1 - \left(1 - \frac{1}{L}\right)^{nk} \end{aligned}$$

Usando o fato que $\mathrm{e}^r = \lim_{n\to\infty}(1 + r/n)^n$, temos que

$$\begin{aligned} P(V[1] = 1) &= 1 - \left(\left(1 + \frac{-1}{L}\right)^L\right)^{nk/L} \\ &\approx 1 - \mathrm{e}^{-nk/L} \end{aligned}$$

Portanto,

$$P(\text{FalsoPositivo}) \approx (1 - \mathrm{e}^{-nk/L})^k$$

provando o item (i). Passemos agora ao item (ii), referente ao número k de funções de dispersão que minimiza a probabilidade de falso positivo, mantendo-se fixo n/L. Para tanto, derivando a expressão $P(\text{FalsoPositivo})$ com respeito a k e igualando-a a zero, temos uma equação em k que conduz ao valor mínimo. Para simplificar um pouco as contas, contudo, utiliza-se do fato de que para funções quaisquer $f(k), g(k)$ tais que

$$f(k) = \mathrm{e}^{g(k)}$$

temos que

$$\frac{\mathrm{d}f(k)}{\mathrm{d}k} = \mathrm{e}^{g(k)}\frac{\mathrm{d}g(k)}{\mathrm{d}k}$$

e, como $\mathrm{e}^{g(k)}$ não se anula, a derivada de $f(k)$ se anula exatamente nos mesmos pontos para os quais a derivada de $g(k)$ se anula. Aplicando tal observação, com $f(k) = P(\textsc{FalsoPositivo})$ e $g(k) = \ln P(\textsc{FalsoPositivo}) = k\ln(1-\mathrm{e}^{-nk/L})$, queremos determinar as raízes de

$$\frac{\mathrm{d}g(k)}{\mathrm{d}k} = \ln(1-\mathrm{e}^{-nk/L}) + \frac{nk}{L}\cdot\frac{\mathrm{e}^{-nk/L}}{1-\mathrm{e}^{-nk/L}}$$

É fácil verificar que

$$k = \frac{L\ln(2)}{n}$$

é uma raiz da função acima. Pode-se mostrar também (Exercício 5) que este é o ponto de mínimo global. Naturalmente, o número ótimo de funções de dispersão deve ser um valor inteiro próximo daquele de k calculado acima. Portanto, para tal valor de k,

$$\begin{aligned} P(\textsc{FalsoPositivo}) &\approx \left(\frac{1}{2}\right)^{\frac{L\ln(2)}{n}} \\ &\approx (0{,}6185)^{\frac{L}{n}} \end{aligned}$$

□

Assim, para $L = 10n$, a probabilidade de se obter falsos positivos é de 0,82%.

5.7 Frequência de Elementos

Seja $S = a_1, a_2, \ldots, a_n$ um fluxo, com $|a_i| \leqslant \ell$ para todo $1 \leqslant i \leqslant n$. Nesta seção, trataremos de um problema mais geral que aquele de pertinência. Estaremos interessados em determinar a frequência de um elemento, ou seja, a quantidade de elementos do fluxo iguais a dado valor de consulta. Entre as aplicações desse tipo de problema, podemos citar aquela relacionada a um processo de votação, a partir de um fluxo com os votos, cada voto consistindo da identificação de um candidato dentre $m = 2^\ell$ possíveis, deseja-se saber a frequência com que os candidatos foram votados. Em particular, pode-se desejar determinar o candidato mais votado,

aqueles que foram votados acima de um limiar de número de votos ou se há um candidato que tenha recebido mais da metade dos votos, entre outros problemas. A próxima seção é dedicada a esse último problema.

O algoritmo do elemento majoritário

O primeiro problema dessa categoria, que chamaremos de *Problema do Elemento Majoritário*, será aquele de decidir se algum elemento do fluxo recebeu mais da metade dos votos e, no caso afirmativo, determinar tal elemento. Esse problema já foi estudado na Seção 3.3.2, no contexto de algoritmos randomizados. No contexto deste capítulo, os votos são considerados um fluxo, para o qual é impraticável (ou inconveniente) manter seus elementos todos em memória. Em primeiro lugar, mostramos que qualquer algoritmo determinístico que resolva em uma única leitura do fluxo requer espaço $\Omega(\min(n, m))$, o que torna impraticável no contexto de dados massivos.

Teorema 5.7. *Seja $S = a_1, a_2, \ldots, a_n$ um fluxo, com $0 \leqslant a_i < m = 2^\ell$ para todo $1 \leqslant i \leqslant n$. Qualquer algoritmo determinístico que resolva o problema do elemento majoritário em uma única leitura do fluxo requer espaço $\Omega(\min(n, m))$.*

Demonstração. Seja um algoritmo que resolva o problema do elemento majoritário em uma única leitura do fluxo empregando espaço $o(\min(n, m))$. Considere um fluxo do qual $n/2$ elementos foram lidos e $n/2$ elementos ainda restam ser lidos pelo algoritmo. Note que qualquer elemento já lido pode eventualmente ser o elemento majoritário, o que ocorrerá se os próximos $n/2$ elementos a serem lidos são iguais a qualquer elemento previamente lido. Assim, para que tal algoritmo possa cumprir seu objetivo, é necessário que nesse ponto da leitura do fluxo todos os elementos distintos possam ser distinguidos em memória, pois quaisquer um deles pode ser a resposta final.

A prova é delineada da seguinte forma. Mostra-se a existência de dois fluxos com $n/2$ elementos, com diferentes conjuntos de elementos distintos, tais que o espaço de memória disponível é insuficiente para distinguir os elementos distintos do primeiro fluxo em relação àqueles do segundo, pois os respectivos estados de memória serão mostrados idênticos. Como o conjunto de elementos distintos de ambos os fluxos são diferentes, sem perda de generalidade assuma que no primeiro fluxo há um elemento c inexistente no segundo. Considere agora a operação de adicionar a ambos os fluxos $n/2$ elementos, todos iguais a c. No primeiro fluxo, haverá um elemento majoritário (o elemento c), enquanto no segundo não haverá. Se os estados de memória do algoritmo são indistintos até a leitura da primeira

metade dos elementos de cada um dos fluxos, então as conclusões do algoritmo após a leitura da segunda metade também deverão ser as mesmas, uma vez que os elementos da segunda metade de ambos os fluxos são os mesmos. Naturalmente, uma das respostas estará equivocada, o que resulta em uma contradição o algoritmo resolver o problema. Portanto, não pode existir um algoritmo que resolva o problema empregando espaço $o(\min(n, m))$.

Passemos a mostrar a existência dos dois fluxos anteriormente mencionados. Sejam $k = \lceil n/2 \rceil$ e $\mathcal{C}_k$ o conjunto de todos os fluxos com k elementos entre 0 e $m - 1$. Seja $f(S')$ a função que mapeia um fluxo $S' \in \mathcal{C}_k$ ao conjunto de seus elementos (lembre que um conjunto não admite repetição de elementos). Isto é, os elementos do contradomínio da função f são os subconjuntos de $\mathcal{S} = \{0, 1, \ldots, m-1\}$. Se $k \geqslant m$, qualquer subconjunto de $\mathcal{S}$ é imagem de algum fluxo de $\mathcal{C}_k$. Logo, o conjunto $\mathrm{Im}(f)$ imagem de f é tal que $|\mathrm{Im}(f)| = 2^m$. Se $k < m$, então nem todo subconjunto de $\mathcal{S}$ pode ser mapeado por algum elemento de $\mathcal{C}_k$. Em especial, por exemplo, o próprio conjunto $\mathcal{S}$ não pode ser mapeado, pois não há número suficiente de elementos no fluxo. Nesse caso, temos que

$$|\mathrm{Im}(f)| = \sum_{i=1}^{k} \binom{m}{i} \geqslant \sum_{i=1}^{k} \binom{k}{i} = 2^k - 1 = \Theta(2^k)$$

Como $k = \lceil n/2 \rceil$, temos que

$$|\mathrm{Im}(f)| = \Omega(\min(2^{n/2}, 2^m))$$

Para representar de maneira distinta todos os elementos de $\mathrm{Im}(f)$, é necessário uma memória de $\Omega(\min(n, m))$ bits. Logo, para qualquer algoritmo que utilize espaço M de memória com $M = o(\min(n, m))$ bits, existiriam dois fluxos $S'_1, S'_2 \in \mathcal{C}_k$, com $f(S'_1) \neq f(S'_2)$, para os quais o algoritmo representaria indistintamente $f(S'_1)$ e $f(S'_2)$ em M, ou seja, ambos os espaços de memória seriam idênticos. Isto completa a prova. □

Como consequência do Teorema 5.7, os algoritmos determinísticos para se resolver o problema do elemento majoritário em única leitura de um fluxo são inviáveis se n, m são ambos muito grandes. No entanto, há um algoritmo determinístico com espaço $O(\log n + \ell)$ se for permitido relaxar o requerimento de que o algoritmo diferencie o caso de haver ou não um elemento majoritário. O único requerimento para o algoritmo é que, caso exista um elemento majoritário, que ele seja determinado corretamente. Caso não exista, o algoritmo poderia reportar

qualquer elemento. Em suma, há considerável ganho de espaço se é admissível falsos positivos na solução do problema, mas não falsos negativos.

O algoritmo é como segue. Inicializa-se a variável s com o valor 0 e define-se a variável c como nula. O papel da variável c é guardar, a todo momento, o elemento que será eleito como majoritário ao fim do algoritmo. O valor de s pode ser entendido, informalmente por ora, como o grau de confiança que o algoritmo tem sobre c ser o elemento correto. O algoritmo poderá mudar o valor de c ao longo de sua execução, mas ao final, sempre reportará o valor c como majoritário. O critério de escolha do elemento majoritário é dado a seguir. Para cada elemento a_i lido do fluxo, incrementa-se s se $c = a_i$, isto é, se o novo elemento lido é igual ao candidato corrente a majoritário. Caso $c \neq a_i$, então decrementa-se s se $s > 0$. Se s for nulo, o algoritmo "perde a confiança" de que o valor vigente de c é o candidato certo e troca para o novo lido, isto é, faz-se $c = a_i$ e $s = 1$. Ao final da leitura do fluxo, retorna-se c como elemento majoritário. O Algoritmo 5.15 apresenta esse algoritmo, e o Teorema 5.8 se preocupa com sua correção. Como c é representável por ℓ bits e $s = O(n)$, segue que a complexidade de espaço do algoritmo é $O(\log n + \ell)$.

Algoritmo 5.15 Elemento Majoritário

```
função ELEMENTO-MAJORITÁRIO(S):
    c, s ← NULO, 0
    a ← próximo(S)
    enquanto a ≠ ∅ :
        se c = a então
            s ← s + 1
        senão
            se s > 0 então
                s ← s − 1
            senão
                c, s ← a, 1
        a ← próximo(S)
    retornar c
```

Teorema 5.8. *Seja $S = a_1, a_2, \ldots, a_n$ um fluxo, com $|a_i| \leqslant \ell$ para todo $1 \leqslant i \leqslant n$, onde cada a_i pertence a $\{0, 1, \ldots, m-1\}$, com $m = 2^\ell$. O Algoritmo 5.15 retorna o elemento majoritário caso exista um.*

Demonstração. Para a prova, vejamos cada novo elemento do fluxo como um voto e precisamos verificar, ao final, se há um candidato eleito majoritariamente. A correção do algoritmo pode ser verificada da seguinte maneira. Primeiro, afirmamos que a seguinte proposição é válida após a inicialização do algoritmo para $i = 0$ e ao término da i-ésima iteração para todo $1 \leqslant i \leqslant n$:

$$\text{nvotos}(p, i) \leqslant \begin{cases} \dfrac{i+s}{2} & \text{, se } p = c \\ \dfrac{i-s}{2} & \text{, se } p \neq c \end{cases}$$

onde $\text{nvotos}(p, i)$ consiste do número de votos que o candidato p recebe nos primeiros i elementos de S. Sob a hipótese da validade dessa proposição, temos que, ao final do algoritmo, vale que

$$\text{nvotos}(p, n) \leqslant \frac{n-s}{2}$$

para todo $p \neq c$. Como $s \geqslant 0$, equivale a dizer que todo candidato distinto de c não recebeu votos da maioria. Logo, há um candidato votado pela maioria e este é precisamente c ou simplesmente não há, o que comprova a correção. Falta apenas comprovar a validade da proposição.

A prova é feita por indução em i. Para $i = 0$, temos que

$$\text{nvotos}(p, 0) = \frac{i-s}{2} = \frac{i+s}{2} = 0$$

para qualquer p, pois $i = s = 0$ após a inicialização, o que torna a proposição trivialmente válida. Se $i > 0$, suponha que a afirmação seja válida para todo $0 \leqslant i' < i$. Para o caso geral, note que a proposição faz duas subafirmações, uma para o candidato corrente c e outra para os demais. Além disso, as variáveis c, s são modificadas de três diferentes maneiras, conforme se

$$\text{(i) } c = a_i\text{, (ii) } c \neq a_i \text{ e } s > 0\text{, e (iii) } c \neq a_i \text{ e } s = 0$$

Assim, a prova consiste de mostrar que cada subafirmação é válida em cada um dos três casos, o que totaliza 6 casos a serem analisados. Serão analisados 2 de tais casos em seguida.

- para $c = a_i$: nesse caso, a variável s é incrementada. Assim, por hipótese de indução, ao final da i'-ésima iteração, com $i' < i$, era válido que

$$\text{nvotos}(p, i') \leqslant \begin{cases} \dfrac{i' + s'}{2} & \text{, se } p = c \\ \dfrac{i' - s'}{2} & \text{, se } p \neq c \end{cases}$$

onde s' representa o valor de s ao fim daquela iteração. Para o caso particular da iteração anterior, temos que

  - se $p = c$:

$$\text{nvotos}(p, i) = \text{nvotos}(p, i-1) + 1 \leqslant \frac{(i-1) + (s-1)}{2} + 1 = \frac{i+s}{2}$$

  - se $p \neq c$:

$$\text{nvotos}(p, i) = \text{nvotos}(p, i-1) \leqslant \frac{(i-1) - (s-1)}{2} = \frac{i-s}{2}$$

Os demais quatro casos podem ser analisados de forma análoga (Exercício 6). □

5.8 Semelhança entre Conjuntos

Nesta seção, discutiremos o problema de determinar, dados os conjuntos A e B, o quão semelhantes eles são entre si. A medida de semelhança utilizada é a proporção de elementos comuns entre A e B. Formalmente, a *semelhança* entre A e B, conhecida como *coeficiente de semelhança de Jaccard*, denotada por $J(A, B)$, e definida como

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Pode-se considerar diversas aplicações relacionadas ao problema. Originalmente, a determinação de semelhança foi empregada na detecção de páginas *web* que eram cópias de outras, a menos de pequenas modificações. Para isso, cada página era representada por seu conjunto de palavras e considerava-se que duas páginas eram semelhantes o suficiente para serem consideradas cópias se a semelhança entre os conjuntos de palavras correspondentes ultrapassava certo limiar.

Aplicações análogas na detecção de imagens similares e detecção de plágios em documentos são encontradas na literatura. Em aplicações dessa natureza, os conjuntos a terem sua semelhança medida podem ser de cardinalidade considerável. Nesta seção, assume-se que tais cardinalidades são arbitrariamente grandes, de modo que os elementos de cada conjunto são lidos como fluxos.

Sejam $S_A = a_1, a_2, \ldots, a_r$ e $S_B = b_1, b_2, \ldots, b_s$ fluxos para os quais se deseja computar $J(A, B)$, onde $A = \{a_1, a_2, \ldots, a_r\}$ e $B = \{b_1, b_2, \ldots, b_s\}$. Sejam $n = \max\{r, s\}$ e ℓ o maior tamanho de elemento em bits. No paradigma clássico, pode-se computar $J(A, B)$ em tempo $O(n \log n)$ e espaço $O(n\ell)$ (ver Exercício 7). Porém, em tais algoritmos, todos os elementos precisam ser guardados ao mesmo tempo em memória. Há algoritmos de tempo $O(n)$ e espaço $O(2^\ell)$ para os quais isto não é necessário, mas o espaço requerido é, em geral, proibitivo. A solução apresentada a seguir produz uma estimativa em complexidade de tempo $O(n)$, espaço constante e requer apenas uma leitura de cada elemento.

A técnica a ser apresentada se assemelha àquela empregada no algoritmo de dispersão mínima, apresentado na Seção 5.5. Seja h uma função de dispersão tal que

$$h\colon \{0, 1, \ldots, 2^\ell - 1\} \to \{1, 2, \ldots, L\}$$

para algum natural L. Adicionalmente, para essa aplicação, assumiremos que h seja injetora, isto é, $h(x) \neq h(y)$ se $x \neq y$.

O algoritmo para semelhança se baseia no valor $h^A_{\min}$ (resp. $h^B_{\min}$) definido como o valor mínimo da função h sobre todos os elementos de S_A (resp. S_B). Isto é,

$$h^A_{\min} = \min_{1 \leqslant i \leqslant r} \{h(a_i)\} \quad \text{e} \quad h^B_{\min} = \min_{1 \leqslant i \leqslant s} \{h(b_i)\} \tag{5.3}$$

Em seguida, compara-se o valor de $h^A_{\min}$ e $h^B_{\min}$. Se $A = B$, então certamente $h^A_{\min} = h^B_{\min}$. Caso contrário, ainda pode ocorrer de $h^A_{\min} = h^B_{\min}$ mesmo quando $A \neq B$. É o ponto fundamental da ideia é a determinação da probabilidade com que tal evento ocorre.

Teorema 5.9. *Sejam $h^A_{\min}$ e $h^B_{\min}$ determinados como em (5.3). Temos que*

$$P(h^A_{\min} = h^B_{\min}) = J(A, B)$$

Demonstração. Primeiramente, seja $h_{\min}$ o menor valor de dispersão sobre todos os elementos de A e B. Ou seja,

$$h_{\min} = \min_{a \in A \cup B} \{h(a)\}$$

Note que

$$h_{\min} = \min\{h^A_{\min}, h^B_{\min}\}$$

Seja $a_{\min} \in A \cup B$ tal que $h(a_{\min}) = h_{\min}$. Observe que $h^A_{\min} = h^B_{\min}$ se e somente se

$$h^A_{\min} = h^B_{\min} = h_{\min}$$

que por sua vez ocorre se e somente se

$$a_{\min} \in A \cap B$$

Portanto, a probabilidade de que $h^A_{\min} = h^B_{\min}$ é a mesma daquela que o elemento especial $a_{\min}$ de $A \cup B$ esteja em ambos A e B. Formalmente,

$$P(h^A_{\min} = h^B_{\min}) = P(a_{\min} \in A \cap B) = \frac{|A \cap B|}{|A \cup B|} = J(A, B)$$

□

Seja X_h a variável aleatória de Bernoulli tal que, dados fluxos A, B, $X_h = 1$ se $h^A_{\min} = h^B_{\min}$. Portanto, pelo Teorema 5.9, temos que $X_h = 1$ com probabilidade $p = J(A, B)$ e, assim, o valor esperado $E[X_h]$ é igual a p. Porém, a variância de X_h é muita alta, pois em cada medição, os únicos valores possíveis de X_h são 0 ou 1 e nada entre eles. O algoritmo que virá a seguir emprega a ideia de utilizar k funções $h_1, h_2, \ldots, h_k$ de dispersão e, para cada uma, medir o valor da variável aleatória X_{h_i} e usar como estimador $\overline{J}$ de $J(A, B)$ a média aritmética de $X_{h_1}, X_{h_2}, \ldots, X_{h_k}$. Em outras palavras,

$$\overline{J} = \frac{\sum_{i=1}^{k} X_{h_i}}{k}$$

Enquanto $\text{Var}(X_h)$ corresponde à variância no caso de uma única função de dispersão h, para o estimador $\overline{J}$ temos que

$$\text{Var}(\overline{J}) = \text{Var}\left(\frac{\sum_{i=1}^{k} X_{h_i}}{k}\right) = \text{Var}\left(\sum_{i=1}^{k} \frac{1}{k} X_{h_i}\right) = \sum_{i=1}^{k} \frac{1}{k^2} \text{Var}(X_{h_i})$$

$$= \sum_{i=1}^{k} \frac{1}{k^2} \text{Var}(X_h) = \frac{\text{Var}(X_h)}{k}$$

Algoritmo 5.16 Semelhança de Conjuntos – Coeficiente de Jaccard

função OBTÉM-ASSINATURA(S):
    definir vetor $H_{\min}[1..k] \leftarrow +\infty$
    $a \leftarrow$ *próximo*(S)
    enquanto $a \neq \emptyset$ **:**
        para $i \leftarrow 1$ **até** k **:**
            $H_{\min}[i] \leftarrow \min\{H_{\min}[i], h_i(a)\}$
        $a \leftarrow$ *próximo*(S)
    retornar $H_{\min}$

função SEMELHANÇA(S_A, S_B):
    $H^A_{\min} \leftarrow$ OBTÉM-ASSINATURA(S_A)
    $H^B_{\min} \leftarrow$ OBTÉM-ASSINATURA(S_B)
    $c \leftarrow 0$
    para $i \leftarrow 1$ **até** k **:**
        se $H^A_{\min}[i] = H^B_{\min}[i]$ **então**
            $c \leftarrow c + 1$
    retornar c/k

Portanto, note que o uso de mais funções de dispersão tem o efeito de diminuir a variância da estimativa quando comparada àquela que usa apenas uma função de dispersão. O Algoritmo 5.16 formaliza a discussão anterior para a semelhança de conjuntos pelo coeficiente de Jaccard. A tupla $((h_1)^A_{\min}, (h_2)^A_{\min}, \ldots, (h_k)^A_{\min})$ com os valores de $h^A_{\min}$ para cada função de dispersão $h = h_1, h_2, \ldots, h_k$ com respeito ao conjunto A é chamada de *assinatura* de A.

5.9 Exercícios

5.1 Ajuste o Exemplo 2 para que o algoritmo escolha k elementos de S com distribuição uniforme, ao invés de apenas um.

5.2 Determine as probabilidades $P(K = 11 \mid r = 2^{10} - 1)$ e $P(K = 12 \mid r = 2^{10} - 1)$ no contexto da expressão (5.2).

5.3 Para cada problema $\mathcal{P}$ sendo solucionado neste capítulo por um algoritmo de dados massivos, produza um conjunto de testes conforme a seguinte es-

tratégia.

Seja A certo número arbitrário de amostras. Seja D um conjunto de valores crescentes de naturais, representando tamanhos de fluxo. Para cada $N \in D$, produza A conjuntos de entradas arbitrárias, todas envolvendo um fluxo de tamanho N, e execute dois algoritmos distintos para cada entrada produzida:

- o algoritmo de dados massivos apresentados no capítulo;
- um algoritmo determinístico exato para o problema.

Para cada conjunto de amostras, seja r_{M} a média aritmética dos resultados do algoritmo de dados massivos nas A amostras verificadas. Analogamente, seja r_{D} a média aritmética dos resultados do algoritmo determinístico exato. Tabule os seguintes dados:

Problema $\mathcal{P}$, empregando A amostras:

N	r_{M}	r_{D}	% erro ($\lvert\frac{r_{\mathrm{M}}-r_{\mathrm{D}}}{r_{\mathrm{D}}}\rvert$)
…	…	…	…

Para o problema $\mathcal{P}$, produza uma análise baseada nos dados tabulados acima, envolvendo a média do percentual de erro e outras métricas estatísticas que julgar adequadas.

5.4 Adapte o Teorema 5.4 na hipótese que o contradomínio da função de dispersão h seja um intervalo real fechado $[0, L]$, para alguma constante $L = O(n)$. Nesse caso, mostre que

$$D = \frac{L}{h_{\min}} - 1$$

5.5 Prove que, para filtros de Bloom onde se mantém n/L fixo, o valor de $k = \frac{L\ln(2)}{n}$ é um ótimo global para a expressão da probabilidade de falsos positivos.

5.6 Termine a prova de correção do algoritmo do elemento majoritário (Seção 5.7), estendendo a análise do caso geral da indução para as outras 4 verificações que ficaram pendentes, conforme discutido na prova.

5.7 Sejam A e B conjuntos de cardinalidade n com elementos arbitrários. Considere que o tamanho de cada elemento de A ou B possua no máximo ℓ bits. Elabore algoritmos para determinar $J(A, B)$ com as seguintes complexidades:

(i) tempo $O(n \log n)$ e espaço $O(n\ell)$

(ii) tempo $O(n)$ e espaço $O(2^\ell)$

5.10 Notas Bibliográficas

O algoritmo da contagem de elementos que satisfazem certa propriedade foi apresentado por Morris (1978), cujas ideias incentivaram o emprego de técnicas similares em outros problemas relacionados. No problema de determinação da quantidade de elementos distintos, a solução da Contagem Linear foi apresentada por Whang, Zanden e Taylor (1990). O algoritmo da Contagem Probabilística foi introduzido por Flajolet e Martin (1985). A versão deste algoritmo, conforme apresentada neste capítulo, é devida a Alon, Matias e Szegedy (1999). A partir daí, Flajolet trabalhou com diferentes grupos de coautores para avançar na solução desse problema. Um deles foi o algoritmo LogLog, por Durand e Flajolet (2003). Mais tarde, o algoritmo HyperLogLog por Flajolet, Fusy et al. (2007) foi apresentado. Gibbons (2016) compara os diversos algoritmos de dados massivos para o problema da determinação da quantidade de elementos distintos. O algoritmo para a determinação do elemento majoritário é devido a Misra e Gries (1982). Os filtros de Bloom foram introduzidos em 1970, levando o nome do seu criador (Bloom 1970). O algoritmo para determinação da semelhança de conjuntos foi descoberto por Broder (1997), cujo primeiro emprego foi na detecção de páginas *web* duplicadas no mecanismo de busca do AltaVista. Uma referência recente com vários dos algoritmos apresentados neste capítulo, entre outros problemas em dados massivos, é o livro de Blum, Hopcroft e Kannan (2020). Outra referência, com uma abordagem mais prática, é o livro de Gakhov (2019).

6 Aprendizado de Máquina

6.1 Introdução

Nesse capítulo, abordaremos o tema do aprendizado de máquina. No início do livro, mencionamos que a ciência de dados é matéria relativamente recente, cuja abrangência exata ainda não é consenso comum aos pesquisadores da área. Dependendo da autoria, um texto de ciência de dados pode ou não incluir certos temas. O mesmo se aplica, talvez até em maior grau, à matéria de aprendizado de máquina. Devido, possivelmente, a sua popularidade alcançada nos últimos anos, há casos em que tópicos não inerentes, inclusive à ciência de dados, recebem a denominação geral de *aprendizado de máquina* e são incluídos em seu estudo.

Para compor o presente capítulo, selecionamos alguns tópicos, os quais julgamos relevantes para o aprendizado de máquina. Observamos ainda, que a seleção realizada certamente não esgota o assunto.

O capítulo se inicia com a descrição de uma motivação para o estudo do aprendizado de máquina. Em seguida, descrevemos o problema geral que será tratado no capítulo. Na sequência, apresentamos os principais tipos de aprendizado de máquina encontrados na literatura. A seção seguinte apresenta a aplicação inicial que será tratada no capítulo. Na sequência, descrevemos o algoritmo do Perceptron. O importante conceito da dimensão de Vapnik–Chervonenkis é apresentado

em seguida. O Teorema de Vapnik–Chervonenkis é então formulado. Finalmente, o método da regressão linear é apresentado, cujo objetivo é estabelecer uma relação linear, quando possível, entre dois conjuntos de variáveis, independente e dependente, respectivamente.

6.2 Motivação

O *aprendizado de máquina* é parte da área da ciência da computação conhecida como inteligência artificial. Embora tenha se popularizado, especialmente nos últimos anos, a inteligência artificial, incluído o aprendizado de máquina, já era considerada no passado entre as áreas essenciais para a computação. Por exemplo, nos anos 60, já havia departamentos inteiros em universidades, dedicados quase que integralmente a essas áreas. Depois de uma certa estagnação nos resultados apresentados, essas áreas perderam um pouco sua proeminência, embora permanecessem ativas ao longo dos anos.

Esse panorama sofreu enorme mudança a partir do início desse século. A inteligência artificial, em particular o aprendizado de máquina, tem colhido resultados concretos e relevantes, em diferentes áreas de aplicação. Os departamentos especializados em inteligência artificial estão ressurgindo nas universidades, motivados pelo sucesso dos resultados da área. Métodos baseados em aprendizado de máquina, apresentam importantes aplicações em atividades tão distintas, como, por exemplo, mercado financeiro: para avaliar grau de risco na concessão de empréstimos; clínica médica: para elaborar possíveis diagnósticos médicos de pacientes; análise de conteúdo de correio eletrônico: para identificar mensagens espúrias; análise de jogos: para elaborar estratégias vencedoras de jogos discretos e muitos outros. Um exemplo elucidativo é o jogo de xadrez. Um programa de computador, baseado em aprendizado de máquina, recentemente derrotou no jogo de xadrez, um outro programa, que utilizava técnica diferente, o qual havia derrotado o enxadrista campeão mundial Garry Kasparov na década de 90. Há poucos anos, seria impensável imaginar que um algoritmo pudesse derrotar o campeão mundial de xadrez.

6.3 A Técnica Geral

O problema geral a ser considerado é o da *classificação*. No caso, supomos que seja dado um conjunto X, cujos elementos desejamos classificar em dois ou mais tipos. De acordo com a classificação encontrada, para cada elemento do conjunto,

seria realizada alguma operação relativa a ele. Por exemplo, se X for um conjunto de possíveis doenças, cada qual quantificada por meio de um conjunto de sintomas atribuídos a um certo indivíduo, o sistema de classificação objetivaria decidir se o indivíduo em questão estaria acometido ou não da doença considerada.

O objetivo é aplicar um método de aprendizado para realizar a classificação. Para tal, supomos a existência de uma etapa inicial, que consiste do *treinamento*. Nessa ocasião, já é conhecida a classificação final de cada elemento do conjunto de dados X, obtida a partir dos quantificadores a ele atribuídos. No exemplo em que X corresponde a um conjunto de doenças, conheceríamos o diagnóstico final, que classificaria cada indivíduo em acometido ou não da doença em questão. Para tal, seriam utilizados os sintomas dos indivíduos e seus possíveis quantificadores. O conjunto de todas essas informações, os sintomas, quantificadores e diagnóstico, constituem os *exemplos de treinamento*.

A técnica do aprendizado de máquina utiliza os exemplos de treinamento para ensinar a máquina a classificar corretamente os dados. Na fase de treinamento, a máquina tentaria efetuar a classificação, a qual seria comparada com a classificação correta, informada por um *professor*, utilizando os exemplos de treinamento. No caso em que a classificação calculada pela máquina não coincidir com a classificação informada pelo professor, seria realizada uma etapa de *correção*. O objetivo é treinar o computador, de modo a que posteriormente possa adquirir independência para realizar a classificação sem o auxílio do professor e dos exemplos de treinamento.

6.4 Tipos de Aprendizado

O tipo mais comum de aprendizado é aquele descrito, em linhas gerais, na seção anterior. Isto é, o aprendizado acontece por meio de um treinamento que utiliza um conjunto de exemplos. Para cada elemento de um conjunto de dados X, juntamente com os seus quantificadores, o professor informa a sua classificação correta. De maneira independente, o algoritmo de aprendizado computa a sua classificação. O objetivo do treinamento é obter a coincidência entre a classificação declarada pelo professor e a computada pelo algoritmo, para cada item do conjunto X. No caso que estamos exemplificando, X é um conjunto de doenças e a classificação consiste em decidir se o indivíduo considerado possui ou não a doença em questão. O algoritmo de aprendizado computará o seu diagnóstico, o qual será confrontado com o diagnóstico informado pelo professor. O algoritmo, então, efetuará sucessivamente etapas de correção, de modo a conseguir um diagnóstico similar ao do

professor. Esse método é denominado *aprendizado supervisionado*. Uma variação desse tipo de aprendizado, denominado *aprendizado semisupervisionado* consiste do caso em que está omitida a classificação do professor em parte dos exemplos de treinamento. Assim, competiria unicamente ao algoritmo de aprendizado decidir esses casos, sem a ajuda da informação do professor.

Um outro tipo de aprendizado ocorre quando a resposta informada pelo professor não é taxativa ou absoluta, como uma resposta "sim" ou "não". Ao contrário, o professor produz a sua resposta, juntamente com alguma informação que possa avaliar a sua qualidade. O algoritmo de aprendizado utilizaria a informação de qualidade, para comparar com a sua própria computação. No caso do exemplo que estamos tratando, da previsão de diagnósticos, a resposta do professor poderia ser o seu diagnóstico, juntamente com alguma informação adicional, por exemplo, acerca da confiabilidade que permitisse avaliar esse diagnóstico. Essa informação poderia ser o percentual de indivíduos cujo diagnóstico foi confirmado, para os sintomas quantificados. Esse tipo de aprendizado se denomina *aprendizado reforçado*.

Existe também um tipo de aprendizado cujos exemplos de treinamento não possuem qualquer resposta ou intervenção por parte do professor. Esses exemplos são apenas dados de entrada sem informação da saída. Esse tipo é denominado *aprendizado não supervisionado*. Toda a tarefa de analisar os dados e buscar a melhor maneira de aprendizado é delegada ao algoritmo de aprendizado. Um caso desse tipo de aprendizado poderia ser possivelmente encontrado em um algoritmo cujo objetivo seja o de jogar xadrez. Esse algoritmo poderia utilizar, como exemplos de treinamento, um conjunto de aberturas realizadas por competidores do jogo de xadrez em competições enxadrísticas. Essas aberturas não estariam classificadas quanto ao seu sucesso ou não, registradas nos movimentos posteriores do jogo. O objetivo seria ensiná-las à máquina. Caberia ao algoritmo de aprendizado efetuar a análise e decidir quanto à conveniência ou não de adotá-las.

6.5 Aplicação Inicial

Nessa seção, descrevemos um exemplo de problema que pode ser tratado através de aprendizado de máquina.

O problema consiste em elaborar um método geral para realizar diagnósticos médicos. Para tal, recorremos ao profissional especializado, no caso, um médico. Obtemos, então, uma lista de possíveis doenças, como, por exemplo, gripe, gastrite, apendicite, bronquite, covid-19 etc. Em paralelo, compilamos uma lista dos

elementos que serão usados para elaborar o diagnóstico: sintomas, temperatura do paciente, resultados do hemograma, dificuldade de locomoção do paciente etc. O conjunto das doenças corresponde à informação principal que desejamos prever, representado por um vetor. Para cada doença, isto é, elemento do vetor são considerados todos os componentes do diagnóstico com a informação da sua relevância para a doença em questão. Em princípio, iremos considerar os componentes do diagnóstico como informação binária se relevante ou não. Por outro lado, utilizamos um conjunto de pesos para ponderar cada sintoma, em relação às doenças. O peso representará a importância relativa do sintoma em questão em relação à doença considerada. Os pesos são representados por números reais, positivos ou negativos. Por exemplo, os sintomas "dor de garganta" e "falta de ar" terão pesos maiores para a doença "gripe" do que para "gastrite". No caso do sintoma "falta de ar", um peso ainda maior para a doença "covid-19".

O conjunto dos elementos que desejamos identificar e classificar que, no exemplo acima, corresponde ao conjunto das doenças, é denominado *conjunto universo*, com n elementos. Cada um desses elementos consiste em um vetor de dimensão d quantificado pelos seus atributos. Cada atributo corresponde a um possível sintoma com a sua ponderação. Esses atributos correspondem ao conjunto de sintomas, denominado *conjunto de predicados*. O conjunto dos pesos corresponde às incógnitas a serem determinadas.

Assim sendo, os dados de entrada correspondem a um vetor, o conjunto universo d-dimensional e com n elementos. Além disso, o conjunto de predicados é conhecido e corresponde a cada dimensão do vetor. Há um conjunto de pesos, a serem determinados, com o objetivo de ponderar cada predicado de cada elemento do conjunto universo. Finalmente, há também um número real $b \neq 0$, a ser determinado, o qual é denominado *limiar*. Com o auxílio desses pesos, será calculado um valor numérico para cada elemento do conjunto universo. Se o valor numérico do elemento em questão for maior do que o limiar, a saída para o elemento correspondente será "sim", isto é, o diagnóstico é positivo para a doença; caso contrário, "não".

Nesse contexto, o problema corresponde a escolher apropriadamente os d pesos para cada elemento do conjunto universo, bem como o valor limiar, de modo a permitir a classificação, "sim" ou "não", para cada elemento.

Como obter uma solução para o problema?

A ideia geral é utilizar o método do aprendizado. Isto é, partimos, inicialmente, de um subconjunto do conjunto universo, que já se encontra classificado, para o qual supõe-se conhecido um limiar apropriado b, bem como os pesos para cada elemento do subconjunto. Com essas informações, os elementos desse subcon-

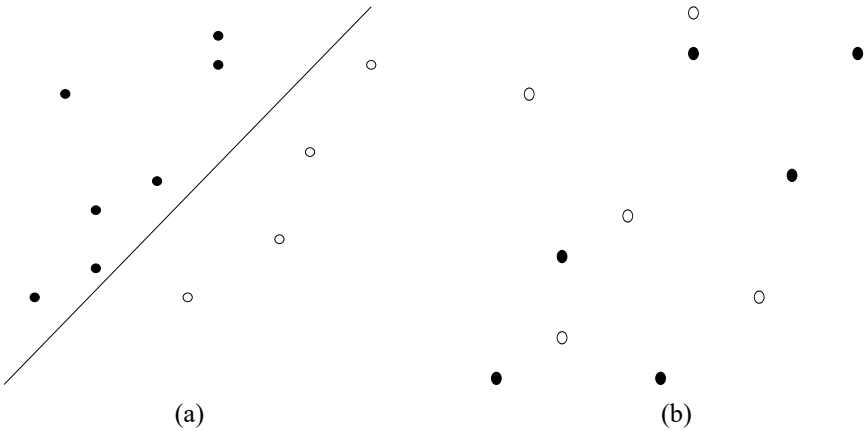


Figura 6.1: Elementos do conjunto universo, classificados segundo os pesos e o limiar. Pontos em negrito representam itens com $y'_i = 1$ e os vazados itens com $y'_i = -1$.

junto já terão o seu valor numérico associado, o qual corresponde a um vetor d-dimensional. Observando o conjunto de valores numéricos, eles deverão permitir uma separação através de um hiperplano, que divida o espaço d-dimensional em duas partes compatíveis com os valores numéricos obtidos: o semiespaço "sim" e o semiespaço "não".

No aprendizado de máquina, os valores atribuídos a esse subconjunto inicial, são obtidos seguindo as informações fornecidas pelo especialista em questão, denominado *professor*. A ideia seria aprender, de certa forma, como esses valores foram atribuídos de modo a automatizar o processo. As atribuições obtidas do professor são denominadas *exemplos de aprendizado* ou simplesmente *exemplos*.

Uma ilustração de uma possível separação em dois semiespaços é apresentada na Figura 6.1(a), no qual os valores atribuídos aos pesos e ao limiar permitiriam essa separação. No exemplo da Figura 6.1(b), no entanto, essa separação não é possível.

Ambos os exemplos são de dimensão 2, cujos hiperplanos consistem de retas no plano. Os círculos em negrito representam elementos que deveriam ser classificados como "sim", enquanto os vazados classificados como "não". Devemos ressaltar que o objetivo é definir os pesos de modo que os elementos do conjunto universo possam ser separados linearmente, isto é, segundo um hiperplano. Caso não seja possível, o processo se torna mais complexo.

De acordo com a descrição acima, o valor de cada elemento do conjunto universo é obtido por meio de uma soma dos pesos atribuídos a cada predicado do elemento. Se a soma for superior ao valor limiar, a resposta para o referido elemento será rotulado com "sim", caso contrário "não". Iremos utilizar os valores $+1$ e -1, respectivamente, para designar "sim" e "não".

O conjunto dos dados será denotado por X, com elementos $X_1, X_2, \ldots, X_n$, onde cada X_i é um vetor no espaço d-dimensional. Cada X_i, por sua vez, é ponderado por um vetor W também no espaço d-dimensional.

6.5.1 Descrição do Problema

Nessa seção, descreveremos formalmente o problema. Os dados do problema correspondem a uma tripla (X, W, b), onde $X = \{X_1, X_2, \ldots, X_n\}$ é o conjunto universo de itens, $W = \{w_1, w_2, \ldots, w_d\}$ é um conjunto de pesos a serem aplicados sobre um conjunto P de d predicados. Assim, cada item $X_i \in X$ por sua vez é um vetor $X_i = (x_{i1}, x_{i2}, \ldots, x_{id})$, onde a cada um de seus elementos x_{ij} é aplicado o peso w_j do predicado correspondente. A variável b se refere ao limiar. Cada um dos valores x_{ij}, w_j e b é um número real.

O objetivo é determinar um valor y_i igual a $+1$ ou -1, denominado *resultado* de X_i para cada conjunto X_i de entrada. O valor y_i é determinado em função de X_i, W, e b, da seguinte maneira:

$$\sum_{j=1}^{d} x_{ij} w_j + b \begin{cases} > 0 \Rightarrow y_i = +1 \\ < 0 \Rightarrow y_i = -1 \\ = 0 \Rightarrow y_i = 0 \end{cases}$$

No exemplo considerado da aplicação em diagnóstico médico, cada X_i é uma doença possível a ser considerada, cada $x_{ij} \in X_i$ é um valor atribuído ao sintoma relativo ao predicado P_j correspondente, e w_j o seu peso. O objetivo é determinar um resultado y_i, referente à doença X_i, tal que $y_i = +1$ indique que o paciente está acometido da doença X_i, enquanto $y_i = -1$ no caso em que o paciente não esteja acometido da doença. Esse objetivo é alcançado se o conjunto X for linearmente separável. Para tal, iremos determinar um hiperplano

$$\sum_{j=1}^{d} x_{ij} w_j + b = 0$$

o qual fará a separação linear do conjunto entre valores $y_i = +1$ e $y_i = -1$. Os

pontos que satisfazem essa equação estão localizados sobre o hiperplano e possuem $y_i = 0$, o que indica que o hiperplano não realizou a separação do conjunto.

Observe que se não houvesse o limiar b, a origem sempre seria um ponto do hiperplano. E, por consequência, a origem não poderia ser classificada como $+1$ ou -1 conforme a conveniência do problema em questão.

Para tornar a notação ligeiramente mais concisa, iremos eliminar do cálculo o limiar b, acrescentando em seu lugar um predicado P_0, cujos valores relativos aos itens X_i são $x_{10} = x_{20} = \ldots = x_{n0} = b$.
Adicionando também um peso inicial $w_0 = 1$, o cálculo se torna:

$$\sum_{j=0}^{d} x_{ij} w_j \begin{cases} > 0 \Rightarrow y_i = +1 \\ < 0 \Rightarrow y_i = -1 \\ = 0 \Rightarrow y_i = 0 \end{cases}$$

Finalmente, podemos escrever o resultado calculado y_i simplesmente como

$$y_i = \text{sinal}\left(\sum_{j=0}^{d} x_{ij} w_j\right), \text{ onde}$$

$$\text{sinal}(z) = \begin{cases} +1, & \text{se } z > 0 \\ -1, & \text{se } z < 0 \\ 0, & \text{se } z = 0 \end{cases}$$

Daqui em diante, o hiperplano $\sum_{j=0}^{d} x_{ij} w_j = 0$ é aquele procurado para separar linearmente os pontos da entrada.

Além do resultado y_i, para cada $X_i \in X$, há um valor y_i', denominado *resultado previsto*. O resultado previsto é fornecido juntamente com a entrada, em contrapartida ao resultado calculado y_i. O valor y_i', dado, é sempre $y_i' = +1$ ou $y_i' = -1$, não se admitindo o valor $y_i' = 0$.

Os valores de y_i podem, então, ser divididos em dois subconjuntos, segundo o valor seja igual a $+1$ ou -1. Esses valores constituem pontos no espaço de d dimensões. O método que descrevemos nesse capítulo se aplica quando esses subconjuntos forem *linearmente separáveis*. Isto é, existe um hiperplano que separa os pontos dos subconjuntos com $y_i = +1$ e $y_i = -1$. No exemplo da Figura 6.1(a), os pontos em negrito representam aqueles com $y_i' = +1$, os quais podem ser separados dos pontos vazados, com $y_i' = -1$, mediante o traçado de um hiperplano. Representam, pois, um conjunto linearmente separável. A Figura 6.1(b) ilustra um caso de um conjunto que não é linearmente separável.

Assim sendo, a ideia de utilizar esse método de aprendizado para a solução de problemas é a seguinte: são dados os valores reais dos itens X_i, isto é, x_{ij}, $i = 1, 2, \ldots, n$, e $j = 1, 2, \ldots, d$, o valor do limiar b, os valores (iniciais) dos pesos w_j, $j = 1, 2, \ldots, d$, bem como o valor de cada resultado previsto y'_i, $+1$ ou -1. O objetivo é determinar o resultado y_i, $i = 1, 2, \ldots, n$. Em relação ao resultado previsto, esse deve ser fornecido por um especialista ou professor.

A dinâmica da utilização do método está detalhada a seguir. Os valores dos resultados de $y_i, i = 1, 2, \ldots, n$, são determinados utilizando os dados da entrada. Se o resultado y_i for igual ao resultado previsto y'_i, para todo $X_i \in X$, o processo termina. Caso contrário, escolhe-se um item $X_i \in X$, tal que $y_i \neq y'_i$ isto é, sinal$(\sum_j x_{ij} w_j) \neq y'_i$. Nesse caso, os pesos w_j são atualizados, visando corrigir a divergência.

Os pesos serão atualizados segundo as seguintes expressões:

$$w_j \leftarrow w_j + y'_i x_{ij}, \text{ para } 0 \leqslant j \leqslant d \tag{6.1}$$

onde x_{ij} é um elemento do item X_i em que

$$\text{sinal}\left(\sum_{j=0}^{d} x_{ij} w_j\right) \neq y'_i$$

Os pesos, portanto, sofrerão atualizações em iterações distintas do cálculo. Seja $w_j(t)$ o peso do predicado P_j na iteração t do processo.

O lema seguinte garante que após a atualização de w_j, mediante a expressão anterior, o valor do resultado obtido sinal$(\sum_{j=0}^{d} x_{ij} w_j)$ se torna mais próximo do previsto y'_i.

Lema 6.1. *Seja X_i um item de entrada tal que o resultado previsto y'_i difere do esperado y_i, após a iteração t do processo. Suponha que, na iteração $t + 1$, os pesos serão atualizados mediante a aplicação da atribuição (6.1), relativa ao item X_i. Então*

$$\sum_j x_{ij} w_j(t+1) > \sum_j x_{ij} w_j(t)$$

se e somente se $y'_i = +1$.

Demonstração. Como $y'_i \neq y_i$ na iteração t, sabemos que

$$y'_i \neq \text{sinal}\left(\sum_{j=0}^{d} x_{ij} w_j(t)\right)$$

Na iteração $t+1$, cada peso $w_j(t)$ é atualizado para $w_j(t+1)$, cujos valores são

$$w_j(t+1) = w_j(t) + y_i' x_{ij}, \text{ para todo } 0 \leqslant j \leqslant d.$$

Vamos comparar $\sum_j x_{ij} w_j(t+1)$ com $\sum_j x_{ij} w_j(t)$. Assim,

$$\sum_j x_{ij} w_j(t+1) = \sum_j x_{ij} \left[w_j(t) + y_i' x_{ij} \right] =$$

$$= \sum_j \left(x_{ij} w_j(t) + y_i' x_{ij}^2 \right)$$

Da última igualdade, decorre que

$$\sum_j x_{ij} w_j(t+1) > \sum_j x_{ij} w_j(t),$$

se e somente se $y_i' > 0$, isto é, $y_i' = +1$. □

Decorre do lema anterior que, após a atualização de $w_j(t)$, o valor do resultado calculado y_i se tornou igual ao previsto y_i' ou, caso contrário, $\sum_j x_{ij} w_j(t+1)$ se tornou mais próximo de zero, isto é, mais próximo do ponto de mudança de sinal, onde alcançaria o valor de y_i'.

Com efeito, suponha inicialmente $y_i' = +1$. Então $y_i = -1$. Isto é, $\sum_j x_{ij} w_j < 0$. Do Lema 6.1, sabemos que

$$\sum_j x_{ij} w_j(t+1) > \sum_j x_{ij} w_j(t).$$

Nesse caso, se $\sum_j x_{ij} w_j(t+1) > 0$, então $y_i = +1$. Ou seja, a aplicação da atualização de $w_j(t)$ produziu a igualdade desejada $y_i = y_i'$. Mas, caso contrário, $\sum_j x_{ij} w_j(t+1)$ decresceu. Ou seja, se aproximou do valor que tornaria iguais y_i e y_i'.

O argumento para $y_i' = -1$ é similar.

Uma observação importante é que, apesar da atualização dos pesos w_j conduzir a uma aproximação do valor $\sum_j x_{ij} w_j(t+1)$ ao valor de y_i', não há qualquer garantia em relação aos valores de $\sum_j x_{\ell j} w_j(t+1)$, para $\ell \neq i$. De fato, a aplicação da atualização dos pesos w_j, considerando um certo item X_i, em que havia a discordância $y_i \neq y_i'$ na iteração t, pode produzir uma discordância dos valores $y_\ell \neq y_\ell'$, para um certo item X_ℓ, em que havia a igualdade $y_\ell = y_\ell'$ na iteração t, isto é, antes da atualização dos pesos.

6.6 O Algoritmo do Perceptron

O Algoritmo do Perceptron pode ser agora formulado. A entrada consiste de um conjunto universo X, composto de itens X_i, $i = 1, 2, \ldots, n$, onde cada X_i é um vetor de tamanho d, com elementos x_{ij}, $1 \leqslant j \leqslant d$. Há uma sequência de pesos $W = w_1, w_2, \ldots, w_d$, onde cada w_j é aplicado a um predicado P_j. Para cada item $X_i \in X$, é ainda fornecido um resultado previsto y_i'. Finalmente, há o valor limiar b, também fornecido na entrada. Os valores dos elementos x_{ij}, pesos w_j e limiar b são todos reais, enquanto o valor previsto y_i' é igual a ± 1.

O objetivo é determinar o resultado calculado y_i, para cada item X_i. A finalidade é conseguir a igualdade $y_i = y_i'$ para todo $1 \leqslant i \leqslant n$. Enquanto tal igualdade não for atingida, procede-se a atualização dos pesos w_j. O procedimento termina quando a igualdade $y_i = y_i'$ é alcançada para todo $1 \leqslant i \leqslant n$.

O Algoritmo 6.1 descreve o processo delineado. Algumas informações importantes em relação ao Algoritmo do Perceptron são consideradas a seguir.

O Algoritmo do Perceptron geralmente apresenta um ótimo desempenho na prática, desde que o conjunto de dados seja linearmente separável. Se não for linearmente separável, o algoritmo não termina, pois sempre haverá algum item X_i, tal que $y_i \neq y_i'$, após cada iteração.

Se o conjunto de dados for linearmente separável, é possível provar que o algoritmo converge para uma solução. Embora, na prática, ele termine rapidamente, não há qualquer garantia. Na realidade, o algoritmo pode terminar em um número exponencial de passos. A velocidade da convergência na direção de uma solução pode ser avaliada por meio do conceito de margem, descrito a seguir.

Seja X um conjunto de dados linearmente separável, com itens X_i e y_i um resultado, para cada X_i, tal que $y_i = y_i'$. A *margem do separador linear* é a menor distância do ponto correspondente ao resultado y_i de algum item X_i até o hiperplano que separa as soluções $y_i = +1$ das soluções $y_i = -1$. Ver Figura 6.2. A *margem de X* é a maior margem associada sobre todos os separadores lineares de X. O interesse é determinar um separador linear $\sum_{j=0}^{d} x_{ij} w_j = 0$ que apresente margem máxima em relação aos demais.

O conceito de margem tem algumas consequências para o desempenho e estabilidade do Algoritmo do Perceptron. Pode ser provado que o número de passos requerido pelo Algoritmo do Perceptron é, no pior caso, inversamente proporcional ao quadrado da margem da solução. Note que uma margem maior implica em duas vantagens:

1. A primeira é que uma solução é obtida em um menor número de iterações.

Algoritmo 6.1 Algoritmo do Perceptron

Dados: $X = \{X_1, \ldots, X_n\}$, $X_i = \{x_{i1}, \ldots, x_{id}\}$, $W = \{w_1, \ldots, w_d\}$, $Y' = \{y'_1, \ldots, y'_n\}$, b, onde $x_{ij}, w_j, b \in \mathbb{R}$, $b \neq 0$ e $y'_i \in \{-1, +1\}$

para $i \leftarrow 1, 2, \ldots, n$ **:**

$x_{i,0} \leftarrow b$; $X_i \leftarrow X_i \cup \{x_{i,0}\}$

$w_0 \leftarrow 1$; $W \leftarrow W \cup \{w_0\}$

$\ell \leftarrow 0$

▷ *ℓ será igual a 1 assim que uma solução for encontrada*

enquanto $\ell = 0$ **:**

▷ *computar os resultados calculados y_i*

para $i \leftarrow 1, 2, \ldots, n$ **:**

$soma \leftarrow \sum_{j=0}^{d} x_{ij} w_j$

$y_i \leftarrow \text{sinal}(soma)$, onde $\text{sinal}(z) = \begin{cases} +1, & \text{se } z > 0 \\ -1, & \text{se } z < 0 \\ 0, & \text{se } z = 0 \end{cases}$

$i \leftarrow 1$; $\ell \leftarrow 1$

▷ *determinar se a solução foi, de fato, encontrada*

repetir

se $y_i \neq y'_i$ **então**

▷ *solução não foi encontrada...*

para $j \leftarrow 0, 1, \ldots, d$ **:**

$w_j \leftarrow w_j + y'_i x_{ij}$

$\ell \leftarrow 0$

senão

$i \leftarrow i + 1$

até que $i > n$ ou $\ell = 0$

▷ *retorna o hiperplano separador encontrado: $w_1 z_1 + w_2 z_2 + \ldots + w_d z_d + w_0 b = 0$, onde $z_1, z_2, \ldots, z_d$ são os eixos coordenados de $\mathbb{R}^d$*

retornar $(w_0, w_1, \ldots, w_n)$

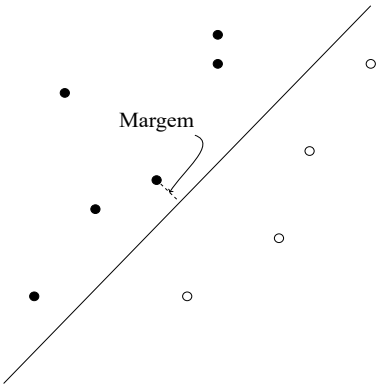


Figura 6.2: Margem de uma solução de um conjunto linearmente separável. Pontos em negrito representam pesos com $y'_i = 1$ e os vazados pesos com $y'_i = -1$.

Intuitivamente, pode ser observado que, se os elementos a serem separados já estão distantes uns dos outros, a tarefa fica facilitada.

2. A segunda é que o sistema fica menos susceptível a alterações na classificação, resultante da introdução de uma pequena variação nos predicados dos itens. Observe que pontos "próximos" aos pontos de entrada tendem a ser classificados no mesmo conjunto.

O exemplo seguinte ilustra o aprendizado da função booleana OU pelo Algoritmo do Perceptron. Os dados de entrada correspondem a $n = 4$ itens de dimensão $d = 2$. Para a entrada do algoritmo, considere $X = [(0,0),(0,1),(1,0),(1,1)]$, $b = -1$, $Y' = [-1,+1,+1,+1]$ e $W = [1,2]$. Inicialmente, o algoritmo adiciona, a cada $X_i \in X$, o elemento $x_{i0} = b = -1$ e, ao conjunto W, o elemento $w_0 = 1$.

Apresentamos, na Tabela 6.1, as iterações do laço principal, com os respectivos valores de suas variáveis antes da alteração dos pesos. Além disso, detalhamos a seguir alguns dos principais procedimentos que ocorrem nas iterações do laço principal. A Figura 6.3 ilustra os hiperplanos $w_1x + w_2y + w_0b = 0$, que são testados durante a execução do algoritmo. Considere que as dimensões 1 e 2 são representadas pelos eixos X e Y, respectivamente.

Na primeira iteração, o hiperplano $x + 2y - 1 = 0$ é testado como separador, conforme a Figura 6.3a. Verifica-se que o único elemento não classificado corretamente é X_3. Isso ocorre, pois $y_3 \neq y'_3$, já que $y_3 = 0$ e $y'_3 = +1$. Por-

tanto, l receberá o valor 0 e deverá ser feita uma adequação nos pesos, conforme as seguintes atribuições.

$$w_0 \leftarrow w_0 + y'_3 * x_{30} = 1 + 1 * (-1) = 0$$
$$w_1 \leftarrow w_1 + y'_3 * x_{31} = 1 + 1 * 1 = 2$$
$$w_2 \leftarrow w_2 + y'_3 * x_{32} = 2 + 1 * 0 = 2$$

Na segunda iteração, o hiperplano $2x + 2y = 0$ é testado como separador, conforme a Figura 6.3b. Com isso, o único elemento não classificado corretamente é X_1. Isso ocorre, pois $y_1 \neq y'_1$, já que $y_1 = 0$ e $y'_1 = -1$. Portanto, l receberá novamente o valor 0 e deverá ser feita uma nova adequação nos pesos, conforme as seguintes atribuições.

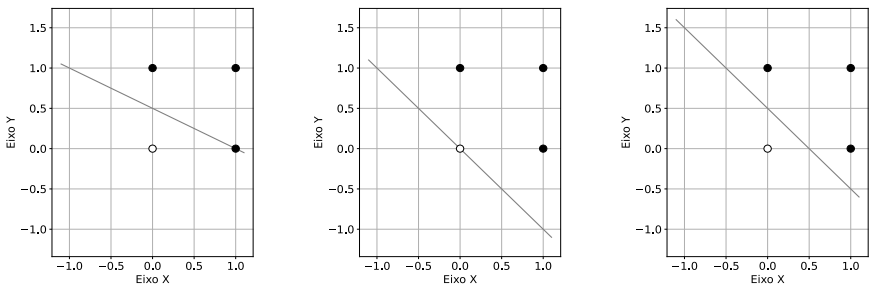
$$w_0 \leftarrow w_0 + y'_1 * x_{10} = 0 + (-1) * (-1) = 1$$
$$w_1 \leftarrow w_1 + y'_1 * x_{11} = 2 + (-1) * 0 = 2$$
$$w_2 \leftarrow w_2 + y'_1 * x_{12} = 2 + (-1) * 0 = 2$$

Na terceira iteração, o hiperplano $2x + 2y - 1 = 0$ é testado como separador, conforme a Figura 6.3c. Note que todos os elementos são classificados corretamente, ou seja, $y_i = y'_i$, $1 \leqslant i \leqslant n$. Portanto, no último laço, a variável l termina com o valor 1, o que faz com que o laço termine. Com isso, o algoritmo retorna o hiperplano $2x + 2y - 1 = 0$ de $\mathbb{R}^2$.

Iteração	y_1	y_2	y_3	y_4	w_0	w_1	w_2	Hiperplano
1	-1	$+1$	0	$+1$	1	1	2	$x + 2y - 1 = 0$
2	0	$+1$	$+1$	$+1$	0	2	2	$2x + 2y = 0$
3	-1	$+1$	$+1$	$+1$	1	2	2	$2x + 2y - 1 = 0$

Tabela 6.1: Tabela com depuração de variáveis com as iterações do laço.

O desempenho do Algoritmo do Perceptron é considerado bastante satisfatório para efeitos práticos. Geralmente, produz resultados em relativamente poucas iterações. Contudo, não há garantia de um bom desempenho. Com efeito, o número



(a) (b) (c)

Figura 6.3: Os hiperplanos definidos a partir dos pesos em W, no início de cada uma das iterações 1, 2 e 3 em (a), (b) e (c), respectivamente.

de iterações necessárias para obter a solução final não é limitado polinomialmente. Além disso, o algoritmo pode não terminar para entradas cujos resultados não admitam uma separação linear.

6.7 A Dimensão de Vapnik–Chervonenkis

O modelo de aprendizado de máquina visto, até o momento, se baseia no princípio de que há um conjunto de dados de uma certa aplicação, cujas variáveis se destinam a determinar uma resposta para cada exemplo. Supondo que a resposta seja binária, seria do tipo "sim" ou "não". Esses dados se destinam ao treinamento do sistema. Isto é, para cada exemplo, já haveria uma resposta "sim" ou "não", previamente conhecida. Mediante a aplicação da técnica de aprendizado de máquina, o objetivo é computar uma resposta calculada que pode ser comparada com a resposta correta já conhecida, para aferir a qualidade do método de aprendizado. Obviamente, a finalidade última seria não somente treinar o método de aprendizado, mas capacitá-lo a produzir respostas corretas.

Uma questão natural seria conhecer se o objetivo de capacitar o método de aprendizado para produzir as respostas corretas é realizável. Esse objetivo pode ser alcançado dentro de limites probabilísticos preestabelecidos. Para tal, utilizamos a Teoria de Vapnik–Chervonenkis, cujos princípios serão expostos nas subseções seguintes. Pela sua importância, essa teoria é considerada por muitos como a parte central do aprendizado de máquina, a qual culmina com o Teorema de Vapnik–Chervonenkis.

6.7.1 Conceituação

Um *sistema de conjuntos* $(U, \mathcal{S})$ consiste de um conjunto U, denominado *universo*, juntamente com uma família $\mathcal{S}$ de subconjuntos de U. Um subconjunto $C \subseteq U$ é dito *aniquilado* se cada subconjunto C' de C pode ser expresso como a interseção de C com algum subconjunto U' de $\mathcal{S}$:

$$C' = C \cap U'$$

Uma definição alternativa para conjuntos aniquilados é a seguinte. Em um sistema de conjuntos $(U, \mathcal{S})$, seja $C \subseteq U$. Considere a coleção formada pelas interseções de C com os subconjuntos de $\mathcal{S}$, denotada por:

$$\mathcal{S} \cap C = \{U' \cap C \mid U' \in \mathcal{S}\}$$

isto é, $\mathcal{S} \cap C$ é a coleção cujos elementos correspondem às interseções de C, com cada um dos subconjuntos $U' \in \mathcal{S}$. Nesse caso, C é um conjunto aniquilado se $\mathcal{S} \cap C$ contém precisamente todas as interseções de C com os subconjuntos de $\mathcal{S}$. Então,

$$|\mathcal{S} \cap C| = 2^{|C|}$$

A *dimensão Vapnik–Chervonenkis* ou *dimensão VC* é a cardinalidade do maior subconjunto de U que está aniquilado. Se essa cardinalidade não for limitada, a dimensão correspondente é ∞.

Em seguida, descrevemos alguns exemplos de sistemas de conjuntos e sua dimensão VC.

(i) Seja $U = \{1, 2, 3\}$ e $\mathcal{S} = \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$.

Considere os seguintes subconjuntos C de U:

$C = \{3\} \Rightarrow C \cap \mathcal{S} = \{\emptyset, \{3\}\} \Rightarrow \{3\}$ é aniquilado.

$C = \{2, 3\} \Rightarrow C \cap \mathcal{S} = \{\emptyset, \{2\}, \{3\}, \{2, 3\}\} \Rightarrow \{2, 3\}$ é aniquilado.

$C = \{1, 2\} \Rightarrow C \cap \mathcal{S} = \{\{1\}, \{1, 2\}\} \Rightarrow \{1, 2\}$ não é aniquilado.

$C = \{1, 2, 3\} \Rightarrow C \cap \mathcal{S} = \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\} \Rightarrow \{1, 2, 3\}$ não é aniquilado.

Observe que a cardinalidade do maior subconjunto de U que é aniquilado é 2. Logo, a dimensão VC do sistema $(\{1, 2, 3\}, \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\})$ é 2.

(ii) Seja U o conjunto dos pontos de uma reta real, e $\mathcal{S}$, o conjunto de intervalos da reta.

Seja $C = \{a, b \mid a, b \in \mathbb{R}, a \neq b\}$.

C está aniquilado, pois existem intervalos distintos I_1, I_2, I_3, I_4, tais que:

$$I_1 \cap C = \emptyset, \quad I_2 \cap C = \{a\}, \quad I_3 \cap C = \{b\}, \quad I_4 \cap C = \{a, b\}$$

Por outro lado, qualquer conjunto C de 3 pontos $\{a, b, c\}$, com $a < b < c$, não pode ser aniquilado, pois qualquer intervalo que contenha simultaneamente a e c, necessariamente contém b.

Assim, a dimensão VC de um conjunto de intervalos de uma reta é 2.

(iii) Seja U o conjunto dos pontos do plano cartesiano, $U = \mathbb{R}^2$, e $\mathcal{S}$ a família dos semiplanos $S \subseteq U$, isto é, delimitados por alguma reta em U.

Seja $C = \{a, b, c\} \subseteq U$, um conjunto de 3 pontos não colineares em U. Por meio do triângulo a, b, c, é fácil escolher semiplanos S', S_a, S_b, S_c, S_{ab}, S_{ac}, S_{bc}, S_{abc} tais que

$$S' \cap C = \emptyset, \quad S_a \cap C = \{a\}, \quad S_b \cap C = \{b\}, \quad S_c \cap C = \{c\},$$

$$S_{ab} \cap C = \{a, b\}, \quad S_{ac} \cap C = \{a, c\}, \quad S_{bc} \cap C = \{b, c\},$$

$$S_{abc} \cap C = \{a, b, c\}$$

Nesse caso, C é um conjunto aniquilado, pois

$$|C \cap \mathcal{S}| = 2^3 = 8$$

Examinemos, em seguida, um conjunto C com pelo menos 4 pontos. Se C contém 3 pontos colineares, então C não pode ser aniquilado, pois qualquer semiplano que contenha os pontos extremos conterá também o ponto central desses 3 pontos colineares. Suponha, agora, que não existam 3 pontos colineares. Escolher 3 pontos arbitrários de C, a, b, c e formar um triângulo com esses pontos. Seja $d \neq a, b, c$ um outro ponto arbitrário de C. Se d for interior a esse triângulo, então C não poderá ser aniquilado, pois qualquer semiplano que contenha d necessariamente conterá a, b ou c. Suponha, então, que a, b, c, d formem um quadrilátero convexo. Sejam b, c os vértices adjacentes a d nesse quadrilátero. Então qualquer semiplano que contenha b, c, deve conter necessariamente a ou d. Logo, C não pode ser aniquilado. Isto é, a dimensão VC de semiplanos no $\mathbb{R}^2$ é < 4. Logo, é igual a 3.

(iv) Seja U o conjunto dos pontos de uma circunferência em um plano. Seja $\mathcal{S}$ a família dos polígonos convexos do plano formados por pontos de U. Qualquer subconjunto dos pontos de U define um polígono convexo, basta considerá-los em ordem consecutiva na circunferência. Seja C um polígono convexo formado por n pontos de U, e C' um polígono convexo formado por um subconjunto desses n pontos. Existe um polígono convexo $U' \in \mathcal{S}$, por exemplo o próprio C', tal que os pontos de C' podem ser obtidos exatamente como interseção de U' e C. Consequentemente,

$$|\mathcal{S} \cap C| = 2^{|C|}$$

Logo, a dimensão VC de $(U, \mathcal{S})$ é infinita.

6.7.2 A Função de Aniquilamento

A *função de aniquilamento* de um sistema $(U, \mathcal{S})$ de conjuntos é a função que relaciona um inteiro n à quantidade máxima de subconjuntos de $C \subseteq U$, $|C| = n$, da forma $S \cap C$, para $S \in \mathcal{S}$. Ou seja, de acordo com a definição da dimensão VC, sabemos que se $C \subseteq U$ é um conjunto aniquilado, então $\mathcal{S} \cap C$ contém todas as intersecções de C com os subconjuntos de $\mathcal{S}$. Isto é, $|\mathcal{S} \cap C| = 2^{|C|}$. Então, a dimensão VC de $(U, \mathcal{S})$ é a máxima cardinalidade de algum subconjunto $C \subseteq U$, tal que $|\mathcal{S} \cap C| = 2^{|C|}$. Consideramos, agora, uma variável n para exprimir a cardinalidade de subconjuntos $C \subseteq U$, isto é, $|C| = n$. Seja d o valor da dimensão VC de $(U, \mathcal{S})$. Então, para $n = |C| \leqslant d$, o número máximo de subconjuntos de C que podem ser aniquilados é 2^n. Pela definição de dimensão VC, não há subconjunto C de U que possa ser aniquilado quando $|C| > d$. Nesse caso, o interesse seria determinar, para cada n, o número máximo de subconjuntos de um conjunto $C \subseteq U$ que podem ser aniquilados. Sabemos que esse valor é $< 2^n$. A função de aniquilamento visa a estudar a relação entre n e esse número máximo. Sabemos que, na faixa $0 \leqslant n \leqslant d$, o número máximo de subconjuntos que podem ser aniquilados é exponencial em n, da forma 2^n. Pode ser provado, contudo, que, a partir do valor $n > d$, esse número máximo passa a ser um polinômio de grau d, em n, isto é, a função de aniquilamento pode ser representada por um gráfico da forma da Figura 6.4.

Assim, a função de aniquilamento $\Pi_{\mathcal{S}}(n)$ de um sistema de conjuntos $(U, \mathcal{S})$ é o valor máximo de subconjuntos de algum conjunto $C \subseteq U$, $|C| = n$, que podem ser aniquilados, isto é, que correspondam a interseções de subconjuntos de

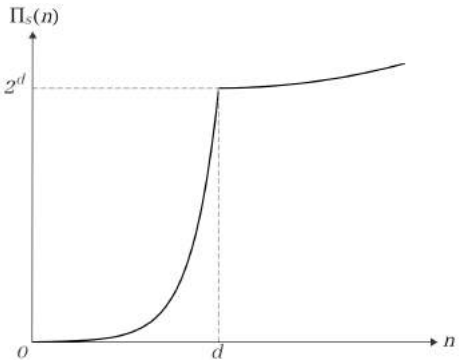


Figura 6.4: A função de aniquilamento $\Pi_{\mathcal{S}}(n)$.

$\mathcal{S}$. Ou seja,

$$\Pi_{\mathcal{S}}(n) = \max_{\substack{C \subseteq U \\ |C| = n}} \left| \{C \cap S \mid S \in \mathcal{S}\} \right|$$

Como exemplo, já foi observado que a dimensão VC de um sistema $(U, \mathcal{S})$, onde $U = \mathbb{R}^2$ e $\mathcal{S}$ é a família dos semiplanos $S \subseteq U$, é igual a 3. Nesse caso,

$$\Pi_{\mathcal{S}}(n) = 2^n, \text{ para } n \leqslant 3,$$

e, a partir de $n > 3$, o crescimento de $\Pi_{\mathcal{S}}(n)$ é mais lento.

Para sistemas de conjuntos $(U, \mathcal{S})$, cuja dimensão d seja limitada, vale o seguinte resultado.

Lema 6.2. *Para qualquer sistema de conjuntos* $(U, \mathcal{S})$*, de dimensão VC* d *limitada, para todo* n *vale*

$$\Pi_{\mathcal{S}}(n) \leqslant \sum_{i=0}^{d} \binom{n}{i} \leqslant 2n^d$$

O lema anterior implica que, para valores $n > d$, o crescimento de $\Pi_{\mathcal{S}}(n)$ é limitado por um polinômio em n de grau d. Observamos que se d for infinito, o valor de $\Pi_{\mathcal{S}}(n)$ é igual a 2^n para todo n.

Em seguida, examinaremos a função de aniquilamento de interseções de sistemas de conjuntos.

6.7.3 Interseção de Sistemas de Conjuntos

Sejam os sistemas de conjuntos $(U, \mathcal{S}_1)$, $(U, \mathcal{S}_2)$, que compartilham o mesmo conjunto universo U. O *sistema de interseção* $(U, \mathcal{S}_1 \cap \mathcal{S}_2)$ é definido como aquele em que os subconjuntos são formados pelas interseções de todos os pares de subconjuntos $S_1 \cap S_2$, $S_1 \in \mathcal{S}_1$ e $S_2 \in \mathcal{S}_2$, isto é:

$$\mathcal{S}_1 \cap \mathcal{S}_2 = \{S_1 \cap S_2 \mid S_1 \in \mathcal{S}_1, S_2 \in \mathcal{S}_2\}$$

O lema seguinte relaciona as funções de aniquilamento de dois sistemas de conjuntos com a função de aniquilamento do respectivo sistema de interseção.

Lema 6.3. *Sejam* $(U, \mathcal{S}_1)$, $(U, \mathcal{S}_2)$ *dois sistemas de conjuntos com o mesmo conjunto universo. Então*

$$\Pi_{\mathcal{S}_1 \cap \mathcal{S}_2}(n) \leqslant \Pi_{\mathcal{S}_1}(n)\Pi_{\mathcal{S}_2}(n)$$

6.8 O Teorema de Vapnik–Chervonenkis

Um dos pilares do aprendizado de máquina consiste no fato de que, por meio da escolha aleatória de amostras de elementos do conjunto universo relativamente pequenas, é possível obter informações e resultados concretos, de todo o universo, com probabilidade relativamente alta. O Teorema de Vapnik–Chervonenkis quantifica o tamanho de amostras.

No caso geral de um sistema de subconjuntos $(U, \mathcal{S})$, supondo que cada elemento do conjunto universo possa ser escolhido para compor a amostra, com uma dada probabilidade, para que a amostra seja representativa do todo, é necessário também avaliar a probabilidade com que os elementos de cada subconjunto S da família sejam membros da amostra. Nesse caso, seria desejável que a probabilidade com que os elementos de U sejam integrantes da amostra se refletisse também em cada subconjunto da família.

Supomos que, para construir a amostra, retiramos elementos do conjunto universo U, cada elemento $u \in U$ com probabilidade $p(u)$. Para um dado subconjunto $S \in \mathcal{S}$, a fração de elementos de S, em relação a $|S|$, que pertencem à amostra será então representada por uma certa probabilidade $p'(S)$. Para que a amostra possa ser de fato representativa do sistema de subconjuntos é então desejável que $p'(S)$ seja aproximadamente igual a $p(S)$, isto é, a probabilidade de que os elementos de S sejam escolhidos.

Dado um sistema de subconjuntos $(U, \mathcal{S})$, o Teorema de Vapnik–Chervonenkis (Teorema VC) visa a determinar a quantidade mínima de amostras necessárias a

serem selecionadas de U, cada elemento da amostra selecionado com uma dada probabilidade p de tal forma que, para cada $S \in \mathcal{S}$, a fração de amostras em S seja, com alta probabilidade, igual a $p(S)$.

Teorema 6.4. *Seja $(U, \mathcal{S})$ um sistema de subconjuntos de dimensão VC igual a d. Considere uma probabilidade arbitrária $p(u)$ de distribuição dos objetos de $u \in U$, e $U' \subseteq U$, $|U'| = n$, um conjunto de amostras retiradas de U, de acordo com a probabilidade $p(u)$. Para qualquer $\epsilon \in (0,1)$, se $n = \Omega(\frac{d}{\epsilon^2} \log \frac{d}{\epsilon})$, então*

$$P\left(\exists S \in \mathcal{S} \mid \left|\frac{|S \cap U'|}{n} - p(S)\right| > \epsilon\right) \leqslant 2\mathrm{e}^{-\epsilon^2 n/6}$$

Em seguida, detalhamos o significado do teorema.

Consideramos um conjunto universo U e uma família $\mathcal{S}$ de subconjuntos de U. Seja $p(u)$ uma probabilidade arbitrária dos objetos $u \in U$, isto é, ao selecionarmos objetos de U, a seleção será realizada segundo a probabilidade $p(u)$. Escolhemos um subconjunto $U' \subseteq U$ com n amostras. A quantidade dessas amostras presentes em cada $S \in \mathcal{S}$ é portanto $|S \cap U'|$. Para uma aproximação desejada $\epsilon \in (0,1)$ e um valor $n = \Omega(\frac{d}{\epsilon^2} \log \frac{d}{\epsilon})$, admitimos que o valor $\left|\frac{|S \cap U'|}{n} - p(S)\right|$ possa ser superior a ϵ, mas, nesse caso, a probabilidade de que isso ocorra seja pequena, limitada a $2\mathrm{e}^{-\epsilon^2 n/6}$. Essa propriedade deve ser satisfeita para qualquer subconjunto $S \in \mathcal{S}$, o que garante, de certa forma, a representatividade da amostra, colhida aleatoriamente.

6.9 Regressão Linear

Consideremos um conjunto de dados da forma (x_i, y_i), $1 \leqslant i \leqslant n$, que podem corresponder a um conjunto de observações ou medidas. Por exemplo, o valor x_i pode corresponder ao consumo de carnes pelo indivíduo i e y_i ao seu nível de colesterol. O nosso objetivo é determinar uma relação entre o valor x_i e o valor y_i. Por meio dessa relação, o valor y_i pode ser determinado em função do valor de x_i, o que possibilitaria sua previsão. Essa técnica se constitui, portanto, em mais um instrumento de aprendizagem de máquina.

A relação mais simples entre os valores

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

é a relação linear. Nesse caso, a tentativa é que cada par (x_i, y_i) corresponda às coordenadas de um ponto de uma reta. Então a relação procurada será expressa por uma equação do tipo

$$y = ax + b$$

A questão será determinar os parâmetros a e b, de tal forma que os valores (x_i, y_i) possam satisfazer

$$y_i = ax_i + b$$

Naturalmente, será aceitável que os pontos (x_i, y_i) não estejam exatamente sobre a reta obtida $y = ax + b$. Mas, nesse caso, deverão estar próximos à reta. Ou seja, os parâmetros a, b a serem calculados introduzem um erro, que será denotado por $E(a, b)$. O nosso objetivo é determinar os parâmetros a, b, a partir dos pares (x_i, y_i), com $1 \leqslant i \leqslant n$, de tal forma que o erro $E(a, b)$ seja minimizado de alguma forma.

A questão agora se torna escolher exatamente a função mais adequada à minimização.

Note que o erro $E_i(a, b)$ cometido ao assumir para o valor y_i aquele obtido pela reta $y = ax + b$ é igual a

$$E_i(a, b) = y_i - (ax_i + b)$$

Portanto, o erro total $E(a, b)$ é igual a

$$E(a, b) = \sum_{i=1}^{n} [y_i - (ax_i + b)]$$

Contudo, a minimização da função $E(a, b)$ anterior é inadequada para os nossos propósitos. Uma razão importante para não adotar essa expressão como a função objetivo a ser minimizada é que os erros em cada ponto são grandezas que admitem sinal positivo ou negativo. Assim, erros negativos poderiam compensar erros positivos, o que poderia tornar o resultado final falso. Uma ideia, então, seria minimizar o módulo do erro de cada ponto. Nesse caso, a função de minimização seria:

$$E(a, b) = \sum_{i=1}^{n} |y_i - (ax_i + b)|$$

A alternativa de minimizar essa última função é também inadequada. O motivo é que tornaria o cálculo da minimização mais complicado, pois o módulo de

uma função a torna não diferenciável, o que praticamente elimina a possibilidade de utilizar o cálculo diferencial para a determinação do valor mínimo de $E(a, b)$.

Finalmente, vamos considerar a alternativa de minimizar a soma dos quadrados dos erros de cada ponto. Esses valores são todos positivos, e a função a ser minimizada é diferenciável. Essa técnica é bastante difundida e recebe o nome de Método dos Mínimos Quadrados. Será examinada na próxima subseção.

6.9.1 O Método dos Mínimos Quadrados

Seja um conjunto de dados da forma

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

onde os valores x_i são designados como variáveis independentes e os valores y_i como variáveis dependentes. O objetivo é ajustar uma relação entre essas variáveis. No caso, essa relação corresponde a uma função linear da forma

$$y = ax + b$$

ou seja, uma reta no $\mathbb{R}^2$. Para tal, necessitamos determinar os parâmetros a e b. De um modo geral, os dados (x_i, y_i), $1 \leqslant i \leqslant n$, correspondem a pontos no $\mathbb{R}^2$, e esses pontos não se distribuem exatamente em uma reta. Então o objetivo é o de encontrar a "melhor" reta, ou seja, determinar os parâmetros a e b que conduzem à reta que representará o conjunto de dados (x_i, y_i). Naturalmente, a reta ajustada será uma aproximação, em princípio não passará exatamente sobre os pontos (x_i, y_i). Assim, em lugar de passar pelo ponto (x_i, y_i), a reta passará pelo ponto $(x_i, ax_i + b)$. O erro cometido, em relação ao ponto, é da forma $E_i(a, b) = y_i - (ax_i + b)$.

Conforme mencionado na subseção anterior, o objetivo é encontrar a reta que minimize a soma dos quadrados dos erros. Isto é, a e b serão determinados de modo a minimizar

$$E(a, b) = \sum_{i=1}^{n} E_i^2(a, b) = \sum_{i=1}^{n} [y_i - (ax_i + b)]^2$$

Para encontrar os valores de a e b que minimizam a função anterior, empregamos o cálculo diferencial. Para tal, devemos encontrar a derivada parcial $\frac{\partial f}{\partial a} E(a, b)$ relativa a a, bem como a derivada parcial $\frac{\partial f}{\partial b} E(a, b)$ relativa a b. Os pontos minimizantes são exatamente aqueles que anulam essas derivadas parciais.

Os valores das derivadas parciais são:

$$\frac{\partial E}{\partial a} = 2\sum_{i=1}^{n}[y_i - (ax_i + b)].(-x_i)$$

$$\frac{\partial E}{\partial b} = 2\sum_{i=1}^{n}[y_i - (ax_i + b)].(-1)$$

Igualando as derivadas a zero, obtemos

$$\frac{\partial E}{\partial a} = 0 \;\Rightarrow \sum_{i=1}^{n}[y_i - (ax_i + b)]x_i = 0$$

$$\frac{\partial E}{\partial b} = 0 \;\Rightarrow \sum_{i=1}^{n}[y_i - (ax_i + b)] = 0$$

Considerando, inicialmente, a equação $\frac{\partial E}{\partial b} = 0$

$$\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} b = 0$$

Dividindo por n

$$\frac{1}{n}\sum_{i=1}^{n} y_i - \frac{a}{n}\sum_{i=1}^{n} x_i - \frac{1}{n}\sum_{i=1}^{n} b = 0$$

Representando por $\overline{x}$ e $\overline{y}$, respectivamente, as médias aritméticas dos valores x_i e y_i, isto é,

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{e} \quad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

obtemos

$$\overline{y} - a\overline{x} - b = 0$$

Logo,

$$b = \overline{y} - a\overline{x}$$

Resta determinar o valor do parâmetro a, da reta. Substituindo o resultado de b, na equação $\frac{\partial E}{\partial a} = 0$, obtemos

$$\sum_{i=1}^{n}[y_i - (ax_i + \overline{y} - a\overline{x})]x_i = 0$$

$$\sum_{i=1}^{n}[x_i(y_i - \overline{y}) + ax_i(\overline{x} - x_i)] = 0$$

$$\sum_{i=1}^{n}[x_i(y_i - \overline{y})] + a\sum_{i=1}^{n} x_i(\overline{x} - x_i) = 0$$

Logo,

$$a = \frac{\sum_{i=1}^{n} x_i(y_i - \overline{y})}{\sum_{i=1}^{n} x_i(x_i - \overline{x})}$$

Com os valores de a e b calculados podemos determinar a reta $y = ax + b$, que minimiza a soma dos quadrados dos erros, conforme desejado.

Observamos que a expressão anterior somente é válida se $\sum_{i=1}^{n} x_i(x_i - \overline{x}) \neq 0$, equivalente à condição de que os valores x_i não sejam todos idênticos, isto é, $x_i \neq x_j$, para algum par i, j, com $1 \leqslant i, j \leqslant n$.

O método descrito conduz ao seguinte algoritmo para determinar a reta que minimiza a soma dos quadrados dos erros, em relação aos dados fornecidos.

Os dados compreendem os vetores $X = \{x_i\}$ e $Y = \{y_i\}$, $1 \leqslant i \leqslant n$, com a condição de que $x_i \neq x_j$ para algum par i, j. O algoritmo para ajustar uma reta $y = ax + b$ aos dados correspondentes, que retorna os valores de a e b calculados, é dado pelo Algoritmo 6.2.

Algoritmo 6.2 Regressão linear

Dados: vetores $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$

$\overline{x} \leftarrow \left(\sum_{i=1}^{n} x_i\right)/n$; $\overline{y} \leftarrow \left(\sum_{i=1}^{n} y_i\right)/n$

$a \leftarrow \left(\sum_{i=1}^{n} x_i(y_i - \overline{y})\right) / \left(\sum_{i=1}^{n} x_i(x_i - \overline{x})\right)$

$b \leftarrow \overline{y} - a\overline{x}$

retornar a, b

É imediato verificar que o algoritmo possui complexidade $O(n)$. A sua correção também é imediata, pois o algoritmo se resume a calcular os valores de a e b, segundo a fórmula descrita.

Como exemplo, suponha que se deseja avaliar a temperatura y de uma certa massa x. Sabe-se que a temperatura depende dessa massa e varia diretamente com ela. Para tal, são efetuadas medições, cujos valores da temperatura y_i e da massa correspondente foram os ilustrados na Tabela 6.2.

i	x_i	y_i
1	5	-2
2	8	0
3	10	3
4	12	5
5	13	7
6	15	8
7	20	10
8	25	13
9	30	15
10	34	18

Tabela 6.2: Valores da temperatura y_i em função da massa x_i

Para ajustar uma reta correspondente a esses pontos, aplicamos a técnica dos mínimos quadrados. A equação da reta é $y = ax + b$, onde

$$a = \frac{\sum_{i=1}^{n} x_i(y_i - \overline{y})}{\sum_{i=1}^{n} x_i(x_i - \overline{x})} \text{ e } b = \overline{y} - a\overline{x},$$

e as médias $\overline{x}$ e $\overline{y}$, respectivamente das medidas x_i e y_i são iguais a

$$\overline{x} = \frac{\sum_{i=1}^{10} x_i}{10} = 17{,}2 \quad \overline{y} = \frac{\sum_{i=1}^{10} y_i}{10} = 7{,}7$$

Logo,

$$a = \frac{553{,}6}{849{,}6} = 0{,}65 \quad b = 7{,}7 - 0{,}65 \cdot 17{,}2 = -3{,}51$$

A equação $y = ax + b$ da reta ajustada é $y = 0{,}65x - 3{,}51$. Veja a Figura 6.5.

A regressão linear no $\mathbb{R}^2$, conforme abordada nessa subseção é conhecida pela denominação *regressão linear simples*. Quando a quantidade de variáveis independentes é maior do que 1, a denominação passa a ser *regressão linear múltipla*. Nesse caso, ao invés de tentar ajustar uma reta, a tentativa é de ajuste de um hiperplano, de mesma dimensão que a quantidade de variáveis independentes. Pode-se aplicar técnica similar à utilizada para a regressão linear simples.

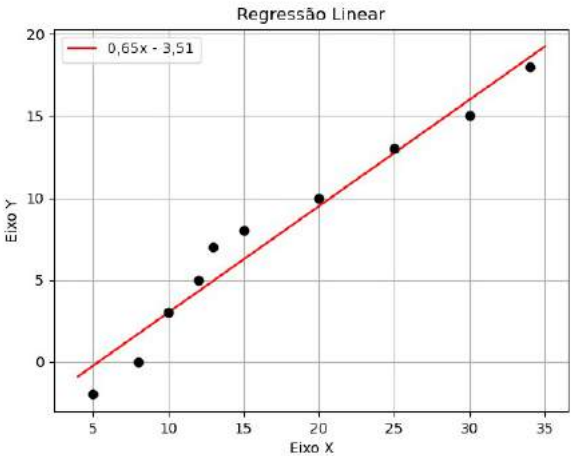


Figura 6.5: Regressão linear para o conjunto de pontos da Tabela 6.2.

6.10 Exercícios

6.1 Considere o problema de, em um conjunto de pacientes, separar aqueles que têm uma certa doença daqueles que não têm. Modelar esse problema de maneira similar àquela apresentada na introdução do capítulo.

6.2 Para cada uma das funções abaixo, apresentar a entrada do Algoritmo do Perceptron que reconheça tais funções. Quando não for possível, justificar.

(a) E lógico

(b) OU lógico

(c) OU-Exclusivo lógico

6.3 Generalizar o resultado de que a dimensão VC de semiplanos no $\mathbb{R}^2$ é igual a 3. Provar, então, que a dimensão VC de semi-hiperplanos no $\mathbb{R}^d$ é $d + 1$.

6.4 Seja U o conjunto dos pontos em um plano, e $\mathcal{S}$ o conjunto de retângulos desse plano de lados paralelos aos eixos. Mostrar que a dimensão VC de $(U, \mathcal{S})$ é igual a 4.

6.5 Seja U o conjunto de pontos de um plano, e $\mathcal{S}$ o conjunto de todos os subconjuntos finitos de U. Mostrar que a dimensão VC de $(U, \mathcal{S})$ é igual a ∞.

6.6 Seja U a circunferência de um círculo, e $\mathcal{S}$ o conjunto de k arcos desse círculo em U. Demonstrar que a dimensão VC de $(U, \mathcal{S})$ é igual a $2(k+1)$.

6.7 Sejam $(U, \mathcal{S}_1)$ e $(U, \mathcal{S}_2)$ dois sistemas de subconjuntos, partilhando o mesmo conjunto universo U. Provar ou dar contraexemplo para a afirmação:

"(dimensão VC de $\mathcal{S}_1 \leqslant$ dimensão VC de $\mathcal{S}_2$) $\Leftrightarrow \mathcal{S}_1 \subseteq \mathcal{S}_2$".

6.8 Obter a reta correspondente à regressão linear considerando apenas as 5 primeiras medições da Tabela 6.2.

6.9 Mostrar que a expressão

$$a = \frac{\sum_{i=1}^{n} x_i (y_i - \overline{y})}{\sum_{i=1}^{n} x_i (x_i - \overline{x})}$$

utilizada na determinação da reta de ajuste na regressão linear é válida se e somente se os valores $x_1, \ldots, x_n$ não são todos idênticos.

6.10 O método de regressão linear foi apresentado para o caso no $\mathbb{R}^2$. Generalizar para k dimensões, em que o vetor X se encontra no $\mathbb{R}^k$.

6.11 Dados os vetores $X = \{x_i\}$ e $Y = \{y_i\}, 1 \leqslant i \leqslant n$, ajustar entre esses dados uma equação do segundo grau do tipo

$$y = ax^2 + bx + c,$$

utilizando uma técnica similar à de minimização da soma dos quadrados dos erros cometidos.

6.12 Estabelecer uma analogia, se possível, entre o Exemplo (iii) da Dimensão VC e o funcionamento do Perceptron. Justificar.

6.11 Notas Bibliográficas

O artigo fundamental que descreveu os princípios básicos da teoria do aprendizado de máquina é o de Valiant (1984). Há contudo, outros textos anteriores que

tratam de temas que foram incorporados ao aprendizado de máquina, como o de Rosenblatt (1958), sobre o Perceptron. Entre outros trabalhos sobre o Perceptron, mencionamos aqueles de Gallant (1990) e Minsky e Papert (1988). Uma referência sobre aprendizado reforçado é (Kaelbling, Littman e A. W. Moore 1996) e, para aprendizado não supervisionado, mencionamos o trabalho de Ghahramani (2004). Uma referência para o problema de classificação, tratado nesse capítulo, é (Langford 2005). O artigo principal que deu origem a teoria de Vapnik–Chervonenkis é (Vapnik e Chervonenkis 1971). Entre os vários trabalhos dedicados ao tema, referenciamos (Abu-Mostafa 1989), (Vapnik, Levin e Cun 1994). Em particular Blumer et al. (1989) estabeleceram a relação entre os exemplos de treinamento e os exemplos considerados nos algoritmos de classificação. Em relação a livros sobre o aprendizado de máquina, inicialmente, mencionamos aqueles de Mitchell (1997) e Abu-Mostafa, Magdon-Ismail e Lin (2012) e ressaltamos os livros de Hopcroft e Kannan (2012) e Blum, Hopcroft e Kannan (2020). Mencionamos ainda o texto de Zaki e Meira (2020) com ênfase em mineração de dados. Em língua portuguesa, citamos o texto de Faceli et al. (2011) com ênfase em inteligência artificial.

7 Cadeias de Markov

7.1 Introdução

As cadeias de Markov formam uma ferramenta poderosa da matemática e da ciência da computação. Suas aplicações incluem a resolução de problemas combinatórios dos mais diversos tipos, tais como a previsão do clima, o movimento Browniano, a dispersão de gases e o espalhamento de doenças. Sua popularidade é devida principalmente ao enorme número de problemas que podem ser modelados através de sua estrutura, adicionado ao seu relacionamento direto com a álgebra linear e com a teoria dos grafos. Essa conexão permite a resolução com eficiência de problemas combinatórios relevantes.

No contexto da ciência de dados, as cadeias de Markov são consideradas um método para uma área chamada de *modelagem preditiva*, que consiste em uma área da ciência de dados interessada na determinação eficiente de probabilidades de eventos.

Neste capítulo, estudamos as cadeias de Markov finitas de tempo discreto. Mostramos como a cadeia de Markov está relacionada à potência da matriz de transição. Mostramos exemplos combinatórios de aplicação das cadeias de Markov, entre eles, a previsão climática. Estudamos suas propriedades topológicas. Definimos as distribuições limitantes e estacionárias de uma cadeia de Markov e

mostramos um algoritmo para determinar a distribuição estacionária de uma cadeia de Markov ergódica.

Inicialmente, apresentamos a definição formal de alguns conceitos importantes e o Teorema 7.1 que associa a potência P^m da matriz de probabilidades com a probabilidade de m passos de uma cadeia de Markov. Em seguida, apresentamos o problema da previsão climática, que será visto sob dois aspectos diferentes, tanto do ponto de vista das cadeias de Markov, quanto da transformação de um problema estocástico não Markoviano em uma cadeia de Markov. Veremos algumas propriedades topológicas da cadeias de Markov. Em particular, o Teorema 7.3 estabelece que as cadeias de Markov com somente estados transientes e absorventes admitem um algoritmo para, dado um estado transiente, determinar o tempo esperado para absorção. A última seção contém as definições das distribuições limitantes e estacionárias de uma cadeia de Markov, como também o Teorema 7.8, o qual estabelece condições suficientes para uma cadeia de Markov admitir uma distribuição limitante que também será estacionária. Ao fim do capítulo, veremos o método da potência que produz a distribuição estacionária de uma cadeia de Markov ergódica.

7.2 Cadeias de Markov

Dados X um conjunto finito e não vazio, com $|X| = n > 0$, $T = \{0, 1, \ldots, t\} \subset \mathbb{N}$, e $X(i) \in X$ uma *variável* ou *estado* que depende de $i \in T$. Um *processo* $C = (X, T)$, também notado $C = \Big(X(0), X(1), \ldots, X(t)\Big)$, muda seu estado de $X(i)$ para o estado $X(i+1)$ em dependência da variável $i \in T$. Dado um processo $\Big(X(i)\Big)_{i \in T}$, o conjunto ou *espaço de estados* de X, é o conjunto $X = \Big\{X(0), X(1), \ldots, X(t)\Big\}$.

Um processo é dito *determinístico* se o valor da variável $X(i)$ pode ser computado de maneira determinística.

Seja $p\colon X \times T \to [0, 1]$ uma probabilidade que associa a mudança do estado $X(i)$ para o estado $X(i+1)$. Um processo $C = (X, T, p)$ é dito ser *estocástico* se $X(i) \in X$ é uma variável aleatória, isto é, se o processo $\Big(X(i)\Big)_{i \in T}$ pode evoluir no tempo de acordo com p, tal que

$$P\Big(X(i+1) = x_{i+1} \mid X(i) = x_i, X(i-1) = x_{i-1}, \ldots, X(0) = x_0\Big)$$

é a *probabilidade de transição* de $X(i)$ para $X(i+1) = x_{i+1}$, dado que $X(i) = x_i, X(i-1) = x_{i-1}, \ldots, X(0) = x_0$.

Um processo estocástico é de *Markov* se

$$P\Big(X(i+1) = x_{i+1} \mid X(i) = x_i, X(i-1) = x_{i-1}, \ldots, X(0) = x_0\Big) =$$

$$P\Big(X(i+1) = x_{i+1} \mid X(i) = x_i\Big) \tag{7.1}$$

isto é, se a probabilidade de um estado $X(i+1)$ ser igual a $x_{i+1} \in X$ depende somente do valor x_i do estado prévio $X(i)$, como definido na Equação 7.1. Em contraste com um processo estocástico, quando p_{jk} pode depender dos demais estados anteriores. Dessa forma, a cadeia de Markov $C = (X, T, p)$, com $|X| = n$ é definida univocamente pela matriz $P_{n\times n}$, onde o elemento

$$p_{jk} = P\Big(X(i+1) = k \mid X(i) = j\Big), j, k \in X$$

Chamamos de *matriz de transição* de C à matriz $P_{n\times n}$.

$$P = \begin{pmatrix} p_{1,1} & p_{1,2}\cdots & p_{1,n} \\ p_{2,1} & p_{2,2}\cdots & p_{2,n} \\ \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2}\cdots & p_{n,n} \end{pmatrix}$$

Dada uma cadeia de Markov $C = (X, T, p)$ com $|X| = n$, definimos o *digrafo completo ponderado* $D(C)$ *associado* à cadeia de Markov C, onde

$$D(C) = (V, E, p) = \Big(X, E, \Big\{P\Big(X(i+1) = y \mid X(i) = x\Big)\Big\}\Big)$$

Caminho Aleatório

Vamos dizer que o processo $(X(0), X(1), \ldots, X(t))$ é um *caminho aleatório* em D. Os caminhos aleatórios serão estudados com mais profundidade no Capítulo 8.

Dada uma cadeia de Markov $C = (X, T, p)$ e $t \geqslant 1$, denotamos por $P^{(t)}$ a *matriz de transição em* $t \geqslant 0$ *passos*, onde cada elemento $p_{kj}^{(t)}$ consiste da probabilidade de *transição do estado* k *para o estado* j *em* t *passos*, isto é

$$p_{kj}^{(t)} = P\Big(X(i+t) = j \mid X(i) = k\Big)$$

Teorema 7.1. *Dada uma cadeia de Markov $C = (X, T, p)$, e $t \geqslant 1$, $P^{(t)} = P^t$, onde $P^t = P^{t-1}P$ é a matriz potência de P.*

Demonstração. Por indução em $t \geqslant 1$. Veja que é válido para $t = 1$. Pois, pela definição de matriz de transição de 1 passo, temos que

$$p_{kj}^{(1)} = P\Big(X(i+1) = j \mid X(i) = k\Big) = p_{kj}$$

Assim, $P^{(1)} = P^1 = P$.

Suponha que seja válido para algum $t \geqslant 1$, isto é, para esse t teremos $P^{(t)} = P^t$. Então

$$P^{t+1} = P^t P = P^{(t)} P = \begin{pmatrix} p_{11}^{(t)} & \cdots & p_{1j}^{(t)} & \cdots & p_{1n}^{(t)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{k1}^{(t)} & \cdots & p_{kj}^{(t)} & \cdots & p_{kn}^{(t)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{n1}^{(t)} & \cdots & p_{nj}^{(t)} & \vdots & p_{nn}^{(t)} \end{pmatrix} \begin{pmatrix} p_{11} & \cdots & p_{1j} & \cdots & p_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{i1} & \cdots & p_{ij} & \cdots & p_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{n1} & \cdots & p_{nj} & \cdots & p_{nn} \end{pmatrix}$$

Assim, cada elemento q_{kj} da matriz produto anterior será

$$q_{kj} = \sum_{\ell=1}^{n} p_{k\ell}^{(t)} p_{\ell j} = \sum_{\ell=1}^{n} p_{\ell j} p_{k\ell}^{(t)}$$

$$\sum_{\ell=1}^{n} P\Big(X(i+t+1) = j \mid X(i+t) = \ell\Big) P\Big(X(i+t) = \ell \mid X(i) = k\Big)$$

cujo último termo pela lei da probabilidade total é igual a $p_{kj}^{(t+1)}$. E assim, $P^{t+1} = P^{(t+1)}$ □

7.3 Problemas iniciais

Nesta seção, descrevemos alguns problemas climáticos que podem ser tratados através de cadeias de Markov. Na primeira parte, vamos ver uma aplicação direta das cadeias de Markov. Na segunda parte, veremos um problema não Markoviano e um algoritmo que permite converter esse problema, em tempo polinomial, para um problema Markoviano. E então, estabelecer sua resolução.

7.3.1 Previsão de Clima

Suponha que existem dois possíveis estados para o clima em uma determinada cidade: *sol* e *chuva*. Você pretende prever, com alguma chance de acerto, qual o clima vai fazer amanhã. Então você determina, pela análise dos boletins meteorológicos, as probabilidades relacionadas à previsão de estados futuros tendo em vista alguns estados anteriores.

Exemplo 1. Em um primeiro exemplo, considere que existe uma chance de 20% de termos um dia de sol em seguida à um dia de chuva, e de 30% de termos um dia de chuva em seguido à um dia de sol.

$$P = \begin{array}{c} \\ chuva \\ sol \end{array} \begin{array}{c} \begin{array}{cc} chuva & sol \end{array} \\ \begin{bmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{bmatrix} \end{array}$$

A cadeia de Markov para essa modelagem é uma matriz $P_{2\times 2}$ com as probabilidades de transições entre os estados possíveis sol e chuva. Cada elemento p_{ij} representa a probabilidade da mudança do estado i para o estado j.

Na primeira linha e coluna, consideramos o estado **chuva**. A probabilidade de após um dia chuva haver um dia também chuva é de 0,8. Na segunda linha e coluna, consideramos o estado **sol**. Assim, a probabilidade após um dia sol haver um dia também sol é de 0,7. O que nos permite considerar a cadeia de Markov $C = (\{1, 2\}, T, p)$, onde chuva=1 e sol=2.

Oferecemos ao leitor na Figura 7.1 um desenho para o digrafo associado à cadeia de Markov.

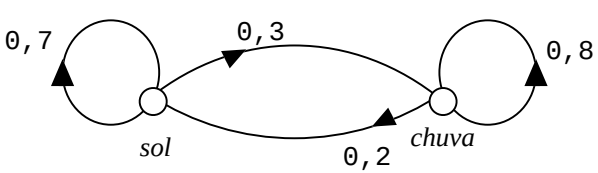


Figura 7.1: Digrafo associado à cadeia C para o Exemplo 1.

Dado que hoje está um dia de sol, qual a probabilidade de ser um dia de chuva daqui a dois dias?

Vamos examinar o valor de:

$$P^{(2)} = \begin{pmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{pmatrix} \begin{pmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{pmatrix} = \begin{pmatrix} 0{,}7 & 0{,}3 \\ 0{,}45 & 0{,}55 \end{pmatrix}.$$

Podemos responder essa pergunta de 2 maneiras:

1. Usando a cadeia de Markov C e o Teorema 7.1, temos que o elemento $p_{2,1}^{(2)} = 0{,}45$ da matriz $P^{(2)}$, ou
2. Usando a lei da probabilidade total

$$P\Big(X(2) = chuva \mid X(0) = sol\Big) =$$

$$\sum_{i \in \{chuva, sol\}} P\Big(X(2) = chuva \mid X(1) = i, X(0) = sol\Big)$$

$$P\Big(X(1) = i \mid X(0) = sol\Big)$$

Como as cadeias de Markov não possuem memória, então a expressão anterior resulta em

$$\sum_{i \in \{chuva, sol\}} P\Big(X(2) = chuva \mid X(1) = i\Big) P\Big(X(1) = i \mid X(0) = sol\Big) =$$

$$P\Big(X(2) = chuva \mid X(1) = chuva\Big) P\Big(X(1) = chuva \mid X(0) = sol\Big) +$$

$$P\Big(X(2) = chuva \mid X(1) = sol\Big) P\Big(X(1) = sol \mid X(0) = sol\Big) =$$

$$0{,}8 \cdot 0{,}3 + 0{,}3 \cdot 0{,}7 = 0{,}45$$

Uma propriedade importante dessa cadeia de Markov (ergódica), que vamos ver ao fim do capítulo, é que, para um número polinomial m de iterações, a matriz potência $P^{(m)}$ converge para uma distribuição limitante independente do estado inicial. Podemos mostrar que, em nosso exemplo

$$P^{(m)} \cong P^{(m+1)} \cong \begin{pmatrix} 0{,}6 & 0{,}4 \\ 0{,}6 & 0{,}4 \end{pmatrix}$$

para $m \geqslant 7$. Isso quer dizer que, em um passeio aleatório no digrafo D, o vértice da *chuva* será visitado muitas vezes mais que o vértice do *sol*, independente do vértice inicial.

7.3.2 Um processo estocástico não Markoviano

Um importante desdobramento do nosso exemplo climático anterior é considerar que podemos analisar um processo estocástico $N = (X, T, p_N)$ com $|X| = n$, mais refinado, com a profundidade d da probabilidade de transição. Isto é, a probabilidade

$$p_N\Big(X(t) = x_{d+1} \mid X(t-1) = x_d, X(t-2) = x_{d-1}, \ldots, X(t-d) = x_1\Big)$$

de uma transição no instante t do estado x_d para um estado x_{d+1} depende dos d estados anteriores nos tempos $t-1, t-2, \ldots, t-d$. Vamos dizer nesse caso que N guarda a memória com esta profundidade d. Isto contraria a definição de cadeia de Markov, porque mais restritamente a cadeia de Markov tem profundidade $d = 1$. Definiremos um algoritmo polinomial $O(n^d)$ no número de estados $n = |X|$ que define uma cadeia de Markov $M = M(N)$ com n^d estados a partir do processo com memória $N = (X, T, p_N)$, que permite usar a cadeia de Markov $M(N)$ para resolver problemas de um processo estocástico de entrada não Markoviano N.

A transformação será feita para um processo $N = (X, T, p_N)$ com somente $n = |X| = 2$ estados, mas pode ser facilmente generalizada para um processo com $n \geqslant 2$ estados.

Dado $N = (X, T, p_N)$ um processo de profundidade $d \geqslant 1$ com um conjunto $X = \{0, 1\}$ com 2 estados.

Vamos usar N e descrever o digrafo $D(M)$ correspondente à cadeia de Markov $M(N)$.

O digrafo $D(M) = \big(V, E\big)$ terá 2^d vértices, onde

$$V = \Big\{x_1 x_2 \ldots x_d : x_i \in \{0, 1\}\Big\}$$

e

$$E = \Big\{(x_1 x_2 \ldots x_d, x_2 \ldots x_d x_{d+1}) :$$

$$p_M\Big(X(t) = x_2 \ldots x_d x_{d+1} \mid X(t-1) = x_1 x_2 \ldots x_d\Big) =$$

$$p_N\Big(X(t) = x_{d+1} \mid X(t-1) = x_d, X(t-2) = x_{d-1}, \ldots, X(t-d) = x_1\Big)\Big\}$$

Como aplicação, revisitamos o Exemplo 1.

Exemplo 2. Consideramos que a probabilidade de chover ou fazer sol em um determinado dia dependa do clima dos dois dias anteriores. Para facilitar a notação, fazemos *sol*=0 e *chuva*=1. Estendendo o exemplo, poderíamos definir que:

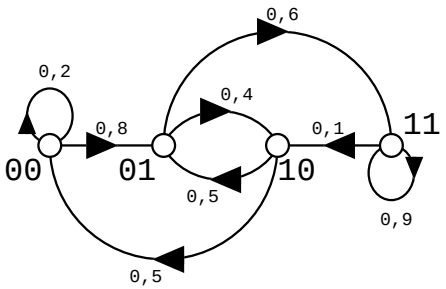


Figura 7.2: Digrafo com pesos associado à cadeia de Markov $M(N)$ para o Exemplo 2.

1. A probabilidade de fazer sol no dia t dado que fez sol nos dias $t-1$ e $t-2$ é 0,2 $\left(p_N(X(t)=0 \mid X(t-1)=0, X(t-2)=0)=0{,}2\right)$.

2. A probabilidade de fazer sol no dia t dado que choveu no dia $t-1$ e fez sol no dia $t-2$ é 0,4 $\left(p_N(X(t)=0 \mid X(t-1)=1, X(t-2)=0)=0{,}4\right)$.

3. A probabilidade de fazer sol no dia t dado que fez sol no dia $t-1$ e choveu no dia $t-2$ é 0,5 $\left(p_N(X(t)=0 \mid X(t-1)=0, X(t-2)=1)=0{,}5\right)$.

4. A probabilidade de fazer sol no dia t dado que choveu nos dias $t-1$ e $t-2$ é 0,1 $\left(p_N(X(t)=0 \mid X(t-1)=1, X(t-2)=1)=0{,}1\right)$.

$$P = \begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{bmatrix} 0{,}2 & 0{,}8 & 0 & 0 \\ 0 & 0 & 0{,}4 & 0{,}6 \\ 0{,}5 & 0{,}5 & 0 & 0 \\ 0 & 0 & 0{,}1 & 0{,}9 \end{bmatrix} \end{array} \qquad P^{(6)} = \begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{bmatrix} 0{,}08 & 0{,}13 & 0{,}14 & 0{,}64 \\ 0{,}08 & 0{,}13 & 0{,}11 & 0{,}66 \\ 0{,}08 & 0{,}16 & 0{,}13 & 0{,}62 \\ 0{,}06 & 0{,}11 & 0{,}11 & 0{,}71 \end{bmatrix} \end{array}$$

(a) (b)

Se fez sol no sábado e domingo, perguntamos qual a probabilidade de chover no próximo sábado?

A matriz P de probabilidade de M é dada pelo item (a), e o digrafo correspondente é mostrado na Figura 7.2.

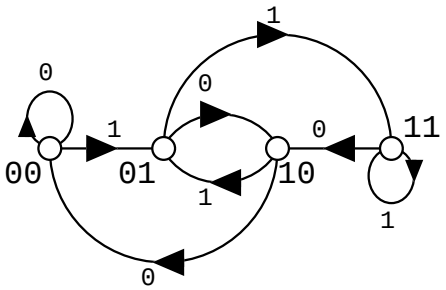


Figura 7.3: Autômato correspondente a uma cadeia de Markov equivalente a um processo estocástico com conjunto de Estados $\{0, 1\}$, cujas probabilidades de transição de estados dependem dos $d = 2$ estados anteriores.

Primeiro de tudo observe que, por hipótese, faz sol nos dias consecutivos: sábado e domingo. Assim, em nossa cadeia de Markov M, podemos considerar que estamos no estado 00. Uma vez que na segunda-feira tenhamos um dia também com sol, continuamos no estado 00 correspondente aos dias correntes: domingo e segunda-feira. Se no entanto tivermos chuva na segunda-feira, então estamos no estado corrente 01, quando temos sol no domingo e chuva na segunda-feira.

O problema é, então, determinar qual a probabilidade de após 6 dias estarmos nos estados 01 ou 11, que consistem nos estados onde teríamos chuva no sábado.

Para isso, vamos olhar agora para a matriz $P^{(6)}$ no item (b).

Dado que fez sol no sábado e no domingo, a probabilidade de chover no sábado corresponde à soma dos elementos $m_{00,01}$ com $m_{00,11}$ da matriz P^6, que é dada por $m_{00,01} + m_{00,11} = 0{,}13 + 0{,}64 = 0{,}77$.

Para um número de estados igual a $n \geqslant 2$, a cadeia de Markov $M(N)$ terá um digrafo correspondente com graus de entrada e saída iguais a n. O digrafo possui uma estrutura semelhante a um autômato finito[1] determinístico associado A, cujas construções são ilustradas nas Figuras 7.3 e 7.4 para os casos onde $n = 2$ e, respectivamente, profundidades $d = 2, 3$.

[1]Na teoria da computação, autômatos finitos são máquinas de estados finitas usadas para aceitar as cadeias de uma Linguagem Formal.

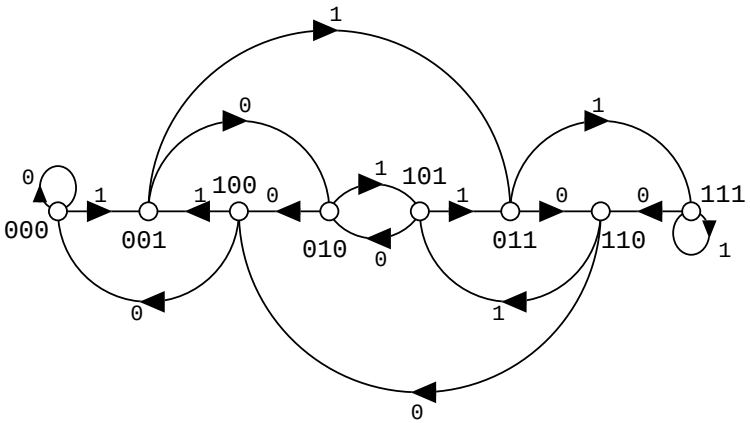


Figura 7.4: Autômato correspondente a uma cadeia de Markov com $n = 2$ estados $\{0, 1\}$, cujas probabilidades de transição de estados dependem dos $d = 3$ estados anteriores.

7.4 Cadeias de Markov - Propriedades topológicas

Nesta seção estamos interessados em definir e estudar as propriedades topológicas que vão definir as propriedades dinâmicas de uma cadeia de Markov.

Dada uma cadeia de Markov $C = (X, T, p)$, dizemos que um estado x_j é *acessível* a partir de um estado x_i se, para algum inteiro t, $p_{ij}^{(t)} > 0$. Os estados x_i e x_j são *comunicantes* se x_i é acessível a partir de x_j e x_j é acessível a partir de x_i. Uma *classe comunicante* é um subconjunto maximal $S \subseteq X$ de estados comunicantes. Dizemos que a cadeia de Markov $C = (X, T, p)$ é *conexa* ou *irredutível* se para todo par $i, j \in X$ i e j são comunicantes. Um estado x_i é *absorvente* se $p_{ii} = 1$. Um estado x_i é *persistente* se

$$P\Big(X(t) = x_i, \text{para algum } t > 0 \mid X(0) = i\Big) = 1$$

e x_i é *transiente* se x_i não é persistente.

Para a conveniência do leitor, oferecemos na Figura 7.5 um exemplo de uma cadeia de Markov $C = (X, T, p)$ na qual estes conceitos são mostrados.

Na cadeia identificamos a classe comunicante $\{a, b\}$ porque $p_{ab} = 0{,}3 > 0$ e $p_{ba} = 0{,}4 > 0$. O vértice c não pertence à classe comunicante $\{a, b\}$ porque

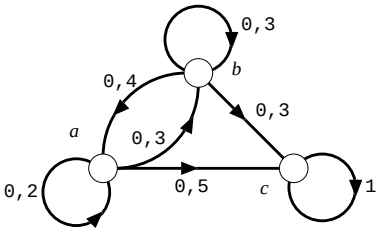


Figura 7.5: Digrafo $D = (X, E, p)$ correspondente a uma cadeia de Markov $C = (X, T, p)$ com duas classes comunicantes.

$p_{cx} = 0$ para todo $x \neq c$. Assim, nem a e nem b são acessíveis a partir de c. A outra classe comunicante de C é $\{c\}$, pois $p_{cc} = 1 > 0$. O vértice c é um estado absorvente. A cadeia de Markov C não é irredutível. O vértice c é persistente porque $P(X(t) = c$, para todo $t > 0 \mid X(0) = c) = 1$. Os vértices a, b são transientes, porque $P(X(t) = a$, para algum $t > 0 \mid X(0) = a) \leqslant 0{,}5$ e $P(X(t) = b$, para algum $t > 0 \mid X(0) = b) \leqslant 0{,}7$.

O próximo teorema caracteriza quando uma cadeia de Markov é irredutível.

Teorema 7.2. *Uma cadeia de Markov $C = (X, T, p)$ é irredutível se e somente se para todo $i, j \in X$, existe $t \in T$ tal que $p_{ij}^{(t)} > 0$.*

7.4.1 Cadeias de Markov com Apenas Estados Absorventes e Transientes

Nesta seção consideramos cadeias de Markov que possuem somente estados transientes e absorventes. Queremos calcular o tempo esperado para, a partir de um estado transiente dado, atingir algum estado absorvente.

Teorema 7.3. *Dada $C = (X, T, p)$ uma cadeia de Markov que só tem estados absorventes e transientes. Sejam A e Q os conjuntos de estados absorventes e transientes de C, respectivamente. Se $x \in Q$ e t_x é o tempo esperado para, a partir do estado x, a cadeia atingir algum dos estados de A, então*

$$t_x = 1 + \sum_{q_i \in Q} p_{xq_i} t_{q_i}$$

Demonstração. Sejam $A = \{a_1, a_2, \ldots, a_k\}$, $Q = \{q_1, q_2, \ldots, q_m\}$ e $|X| = n$.

Veja que pela definição

$$t_x = E\Big(t, X(t) \in A \mid X(0) = x\Big)$$

Usando a linearidade da esperança e a lei da probabilidade total

$$t_x = \sum_{y \in X} E\Big(t, X(t) \in A \mid X(1) = y, X(0) = x\Big) P\Big(X(1) = y \mid X(0) = x\Big)$$

Como C é uma cadeia de Markov, a expressão anterior resulta em

$$t_x = \sum_{y \in X} E\Big(t, X(t) \in A \mid X(1) = y\Big) p_{xy}$$

Agrupando os termos, obtemos o resultado

$$t_x = \sum_{a_i \in A} E\Big(t, X(t) \in A \mid X(1) = a_i\Big) p_{xa_i} +$$

$$\sum_{q_i \in Q} E\Big(t, X(t) \in A \mid X(1) = q_i\Big) p_{xq_i}$$

Usando o fato que $a_i \in A$, a definição de t_{q_i}, e o método do primeiro passo

$$\begin{aligned} t_x &= \sum_{a_i \in A} 1.p_{xa_i} + \sum_{q_i \in Q} (1 + t_{q_i}) p_{xq_i} \\ &= \sum_{y \in X} p_{xy} + \sum_{q_i \in Q} p_{xq_i} t_{q_i} = 1 + \sum_{q_i \in Q} p_{xq_i} t_{q_i} \end{aligned}$$

□

Apesar da formulação do Teorema 7.3 não ser explícita com os valores dos tempos esperados de absorção a partir dos estados transientes, é sempre possível considerar um sistema linear com o qual esses valores podem ser determinados. Consideramos agora um exemplo de aplicação do Teorema 7.3.

Exemplo 3. Dada a cadeia de Markov $C = (X, T, p)$ a seguir, com somente estados absorventes e transientes, determine os tempos de absorção a partir dos estados transientes.

$$P = \begin{pmatrix} 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1/4 & 0 & 0 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

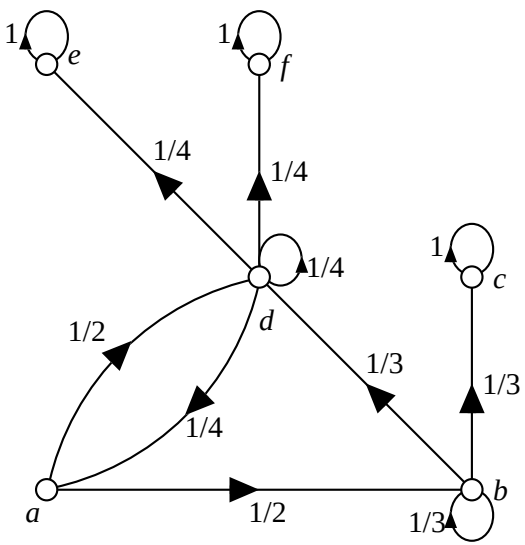


Figura 7.6: Digrafo com pesos $D = (X, E, p)$ para a cadeia de Markov com estados absorventes e transientes para o Exemplo 3.

Pelo Teorema 7.3, temos que os tempos esperados para absorção a partir dos estados transientes $\{a, b, d\}$ de C são dados por

$$t_a = 1 + \sum_{x \in \{a,b,d\}} p_{ax} t_x$$

$$t_b = 1 + \sum_{x \in \{a,b,d\}} p_{bx} t_x$$

$$t_d = 1 + \sum_{x \in \{a,b,d\}} p_{dx} t_x$$

Desenvolvendo os termos temos, que

$$t_a = 1 + p_{ab}t_b + p_{ad}t_d = 1 + \frac{1}{2}t_b + \frac{1}{2}t_d$$

$$t_b = 1 + p_{bb}t_b + p_{bd}t_d = 1 + \frac{1}{3}t_b + \frac{1}{3}t_d$$

$$t_d = 1 + p_{da}t_a + p_{dd}t_d = 1 + \frac{1}{4}t_a + \frac{1}{4}t_d$$

E assim, temos

$$t_a = \frac{33}{9}\,, t_b = \frac{25}{9}\,, t_d = \frac{23}{9}$$

Uma possível interpretação da solução do Exemplo 3 é a seguinte. Suponha que, uma partícula se movimentasse de acordo com as probabilidades dessa cadeia de Markov a partir do vértice a. Então, na maior parte das vezes, após 3 movimentos, a partícula estaria em um dos estados absorventes.

7.5 Distribuição limitante e distribuição estacionária

Nesta seção, vamos definir a distribuição limitante e a distribuição estacionária de uma cadeia de Markov $C = (X, T, p)$.

A *distribuição limitante* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ para $C = (X, T, p)$ é tal que, para todo $x, y \in X$,

$$\lim_{t \to +\infty} P\Big(X(t) = y \mid X(0) = x\Big) = \lim_{t \to +\infty} p_{xy}^{(t)} = \lambda_y$$

Observe que, pela definição, a distribuição limitante independe do estado inicial $X(0) = x$, quando existente.

A *distribuição estacionária* $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ de $C = (X, T, p)$ é dada por

$$\pi_x = \sum_{y \in X} \pi_y p_{yx}$$

Uma distribuição limitante, quando existe, é sempre uma distribuição estacionária, mas o inverso não é verdadeiro. Pode haver uma distribuição

estacionária, que não seja distribuição limitante. No Exemplo 4, mostramos um caso onde isso ocorre.

Exemplo 4. Na Figura 7.7, é apresentada uma cadeia de Markov que tem uma distribuição estacionária e não tem uma distribuição limitante.

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

A cadeia (C, T, p) tem distribuição estacionária $(\pi_1, \pi_2) = (0{,}5, 0{,}5)$, pois

$$\pi_1 = \pi_1 p_{11} + \pi_2 p_{21} = \pi_2$$

Como π é uma probabilidade sob X, temos que $\pi_1 = \pi_2 = 0{,}5$.
Observe que, quando t tende para infinito, não existe um estado $X(t) = y$ que não dependa do valor $X(0)$.

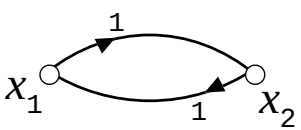


Figura 7.7: Cadeia de Markov sem distribuição limitante, mas com distribuição estacionária

O Exemplo 4 nos motiva a buscar condições necessárias para a existência da distribuição limitante.

Dados uma cadeia de Markov $C = (X, T, p)$ e $x \in X$, o *período* $per(x)$ é

$$per(x) = mdc\left(n \in \mathbb{N} \mid p_{xx}^{(n)} > 0\right)$$

Uma cadeia de Markov $C = (X, T, p)$ é *aperiódica* se qualquer $x \in X$ satisfaz $per(x) = 1$, e *periódica* caso contrário. Dizemos que C é *ergódica* se C é irredutível e aperiódica.

Exemplo 5. Na Figura 7.8, oferecemos um exemplo $C = (X, T, p)$ de cadeia de Markov irredutível e periódica, pois $per(x_0) = per(x_1) = per(x_2) = per(x_3) = 2$, todos os passeios fechados possuem comprimento par e cada estado é acessível a partir de qualquer outro.

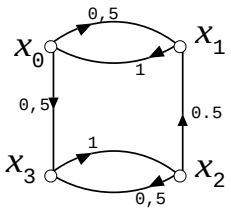


Figura 7.8: Cadeia de Markov irredutível e periódica.

Exemplo 6. Dois cachorros: Rex e Lola possuem k pulgas. Para cada tempo $t \in T$ uma pulga salta de um cachorro para outro. Modelamos o experimento por uma cadeia de Markov $C = (X, T, p)$ onde cada estado de $X = \{0, 1, \ldots, k\}$ representa o número de pulgas correntemente em Lola. A matriz de transição para $k = 4$ é

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 0{,}25 & 0 & 0{,}75 & 0 & 0 \\ 0 & 0{,}5 & 0 & 0{,}5 & 0 \\ 0 & 0 & 0{,}75 & 0 & 0{,}25 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right] \end{array}$$

Para um número arbitrário de pulgas, podemos definir a probabilidade de transição na cadeia como

$$p_{ij} = \begin{cases} (k-i)/k, & \text{se } j = i+1 \\ i/k, & \text{se } j = i-1 \\ 0, & \text{caso contrário} \end{cases}$$

A Figura 7.9 mostra o digrafo $D = (X, E, p)$ correspondente à cadeia de Markov.

Observamos que esta cadeia não é ergódica, porque apesar de ser irredutível, não é aperiódica. Pois, cada estado $x \in X$, possui $per(x) = 2$.

Teorema 7.4. *Dada uma cadeia de Markov $C = (X, T, p)$, se existe $i \in X$, tal que $p_{ii} > 0$, então $per(i) = 1$.*

Demonstração. Decorre da definição de período. □

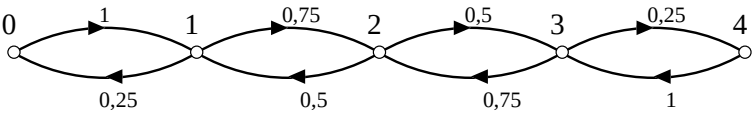


Figura 7.9: Cadeia de Markov irredutível e periódica sem distribuição limitante.

Teorema 7.5. *Dada uma cadeia de Markov $C = (X, T, p)$, se x, y são comunicantes, então $per(x) = per(y)$.*

Demonstração. Considere $D = (X, E, p)$ o digrafo associado a C. Sejam T_{xy} e T_{yx} respectivamente dois passeios em D que ligam x até y e y até x. Quando consideramos a composição de T_{xy} com T_{yx}, temos que $p_{yy}^{(|T_{xy}|+|T_{yx}|)} > 0$, e pela definição $per(y)$ divide $|T_{xy}| + |T_{yx}|$. Dado $r \in \mathbb{N}$, se $p_{xx}^{(r)} > 0$, então temos um passeio fechado R que começa e termina em x de comprimento r. A composição de R com T_{xy} e T_{yx} consiste em um passeio fechado que começa e termina em y. Pela definição de período, $per(y)$ divide $\big(r + |T_{xy}| + |T_{yx}|\big)$.

E temos que $s.per(y) = r + k.per(y)$, que define $r = q.per(y)$, com $k, q, s \in \mathbb{Z}$. Assim, $per(y)$ divide r. E dessa forma, pela definição de período e sabendo que $p_{xx}^{(r)} > 0$, temos $per(y)$ divide $per(x)$. De maneira análoga, $per(x)$ divide $per(y)$. E assim, $per(x) = per(y)$. □

Corolário 7.6. *Dada uma cadeia de Markov $C = (X, T, p)$, se $S \subseteq X$ é uma classe comunicante, então, para todo par $x, y \in S$, $per(x) = per(y)$.*

Teorema 7.7. *Dada uma cadeia de Markov $C = (X, T, p)$, se existe uma distribuição limitante λ de C, então, para todo $x \in X$,*

$$\lambda_x = \sum_{y \in X} \lambda_y p_{yx}$$

Demonstração. Usando a lei da probabilidade total temos que a probabilidade de em um tempo $t + 1$ estarmos no estado x será igual ao somatório dos produtos das probabilidades de estarmos em um estado qualquer y, multiplicado pela probabilidade de fazermos a transição para o estado x a partir do estado y no tempo $t + 1$, isto é

$$P\Big(X(t+1) = x\Big) = \sum_{y \in X} P\Big(X(t+1) = x \mid X(t) = y\Big) P\Big(X(t) = y\Big)$$

Tomando o limite em ambos os lados da igualdade quando $t \to +\infty$, obtemos

$$\lim_{t\to+\infty} p_{zx}^{(t)} = \sum_{y\in X} p_{yx} \lim_{t\to+\infty} p_{zy}^{(t)}$$

para qualquer $z \in X$. E assim, como ambos os limites existem pela hipótese, temos

$$\lambda_x = \sum_{y\in X} \lambda_y p_{yx}$$

□

O teorema a seguir produz uma condição suficiente para a existência de uma distribuição estacionária em uma cadeia de Markov.

Teorema 7.8. *Se $C = (X, T, p)$ é uma cadeia de Markov ergódica, então existe uma única distribuição estacionária π de C, onde*

1. *para todo $x \in X$, $\pi_x = \sum_{y\in X} \pi_y p_{yx} = \lambda_x$*
2. $\sum_{x\in X} \pi_x = 1$

Exemplo 7. Seja $0 < \alpha < 1$, $0 < \beta < 1$, e $C = (X, T, p)$ uma cadeia de Markov, com matriz de transição $P = \begin{bmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{bmatrix}$. Vamos determinar a distribuição estacionária $\pi = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix}$ de C.

Observe que, pelo Teorema 7.8, temos

$$\pi_1 = \pi_1(1-\alpha) + \pi_2\beta$$

$$\pi_2 = \pi_1(\alpha) + \pi_2(1-\beta)$$

$$\pi_1 + \pi_2 = 1$$

Assim, escrevemos

$$\pi_1\alpha = \pi_2\beta$$

$$\pi_1 = 1 - \pi_2$$

E obtemos que

$$\pi_1 = \frac{\beta}{\alpha+\beta} \text{ e } \pi_2 = \frac{\alpha}{\alpha+\beta}$$

Em seguida, revisitamos o Exemplo 1.

Exemplo 8. Relembre que a matriz P de transição do Exemplo 1 é dada por

$$P = \begin{matrix} \\ chuva \\ sol \end{matrix} \begin{matrix} \begin{matrix} chuva & sol \end{matrix} \\ \begin{bmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{bmatrix} \end{matrix}$$

Para este caso, temos $\alpha = 0,2$ e $\beta = 0,3$ e, portanto, a distribuição estacionária π é dada por

$$\pi = \begin{matrix} \begin{bmatrix} chuva & sol \\ \frac{0,3}{0,3+0,2} & \frac{0,2}{0,3+0,2} \end{bmatrix} \end{matrix} = \begin{matrix} \begin{bmatrix} chuva & sol \\ 0,6 & 0,4 \end{bmatrix} \end{matrix}$$

Quando consideramos nesse exemplo a potência $P^{(10)}$, vemos que

$$P^{(10)} = \begin{bmatrix} 0{,}6 & 0{,}4 \\ 0{,}6 & 0{,}4 \end{bmatrix}$$

Isto significa que a partir das condições climática de um dia i qualquer, a probabilidade que, em um tempo t grande o suficiente, seja um dia chuvoso será de 0,6 enquanto que a probabilidade que seja um dia de sol será de 0,4, independente do dia inicial i e de sua condição climática.

7.5.1 O método da potência

Vamos assumir que $C = (X, T, p)$ é uma cadeia de Markov ergódica. Pelo Teorema 7.8, existe uma única distribuição estacionária $\pi = (\pi_1, \ldots, \pi_j, \ldots, \pi_n)$ de C satisfazendo, para todo $j \in \{1, 2, \ldots, n\}$,

$$\pi_j = \sum_{i=1}^{n} \pi_i \, p_{ij} = \lambda_j$$

Podemos reescrever a expressão anterior como

$$1.\pi = \Big(\pi_1, \ldots, \pi_j, \ldots, \pi_n\Big) =$$

$$\left(\pi_1, \ldots, \pi_j, \ldots, \pi_n\right) \begin{pmatrix} p_{11} & \cdots & p_{1j} & \cdots & p_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{n1} & \cdots & p_{nj} & \cdots & p_{nn} \end{pmatrix} = \pi P$$

O *método da potência* consiste em um procedimento convergente que, a partir de um vetor inicial não nulo, representando uma proposta inicial para a distribuição estacionária, refina sucessivamente tal proposta pela multiplicação desse vetor com a matriz P. Sabe-se que, se tal proposta estiver próxima à distribuição estacionária, tal multiplicação não deve alterar o vetor significativamente.

Algoritmo 7.1 Determina uma aproximação $r(t+1)$ para a distribuição estacionária π com erro máximo $||r(t+1) - r(t)|| < \varepsilon$ dados uma cadeia ergódica de Markov $C = (X, T, p)$ com $|X| = n$ e $\varepsilon > 0$.

---

função DIST-ESTAC($C = (X, T, p), \varepsilon$):
  $r(0) \leftarrow (r_1, \ldots, r_n) = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$
  $t \leftarrow 0$
  $r(1) \leftarrow r(0)P$
  enquanto $||r(t+1) - r(t)|| \geqslant \epsilon$ **:**
    $t \leftarrow t + 1$
    $r(t+1) \leftarrow r(t)P$
  retornar $(r(t+1))$

Mais formalmente, consideramos inicialmente um vetor não nulo $r(0)$ e, iterativamente para cada tempo $t \geqslant 0$, obtemos um vetor $r(t+1)$ a partir de $r(t)$, que aproxima a distribuição estacionária π com o valor

$$r(t+1) = r(t)P$$

até que, para algum $t \geqslant 0$, considerando certo erro $\epsilon > 0$, tenhamos

$$||r(t+1) - r(t)|| < \varepsilon$$

onde $||x|| = \sqrt{\sum_{i=1}^{n} x_i^2}$ denota a norma do vetor $x = (x_1, \ldots, x_n)$. Nesse momento, $r(t+1)$ será a distribuição aproximada da estacionária π da cadeia de Markov ergódica C. Isto é

$$r(t+1) = \left(r_1(t+1), r_2(t+1), \ldots, r_j(t+1), \ldots, r_n(t+1)\right) =$$

$$\Big(r_1(t), r_2(t), \ldots, r_j(t), \ldots, r_n(t)\Big) \begin{pmatrix} p_{11} & \cdots & p_{1j} & \cdots & p_{1n} \\ p_{21} & \cdots & p_{2j} & \cdots & p_{2n} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ p_{j1} & \cdots & p_{jj} & \cdots & p_{jn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nj} & \cdots & p_{nn} \end{pmatrix} =$$

$$\Big(r_1(t), r_2(t), \ldots, r_j(t), \ldots, r_n(t)\Big) P$$

O método da potência é formalizado pelo Algoritmo 7.1.

7.6 Exercícios

1. Dados uma matriz quadrada P de ordem n e um natural k, o algoritmo de exponenciação rápida P^k pode ser descrito pela seguinte recorrência:

$$P^k = \begin{cases} I_n, & \text{se } k = 0 \\ (P^{k/2})^2, & \text{se } k > 0 \text{ é par} \\ (P^{k-1})P, & \text{se } k \text{ é ímpar} \end{cases}$$

   onde I_n denota a matriz identidade de ordem n.

   (a) Descreva formalmente tal algoritmo.

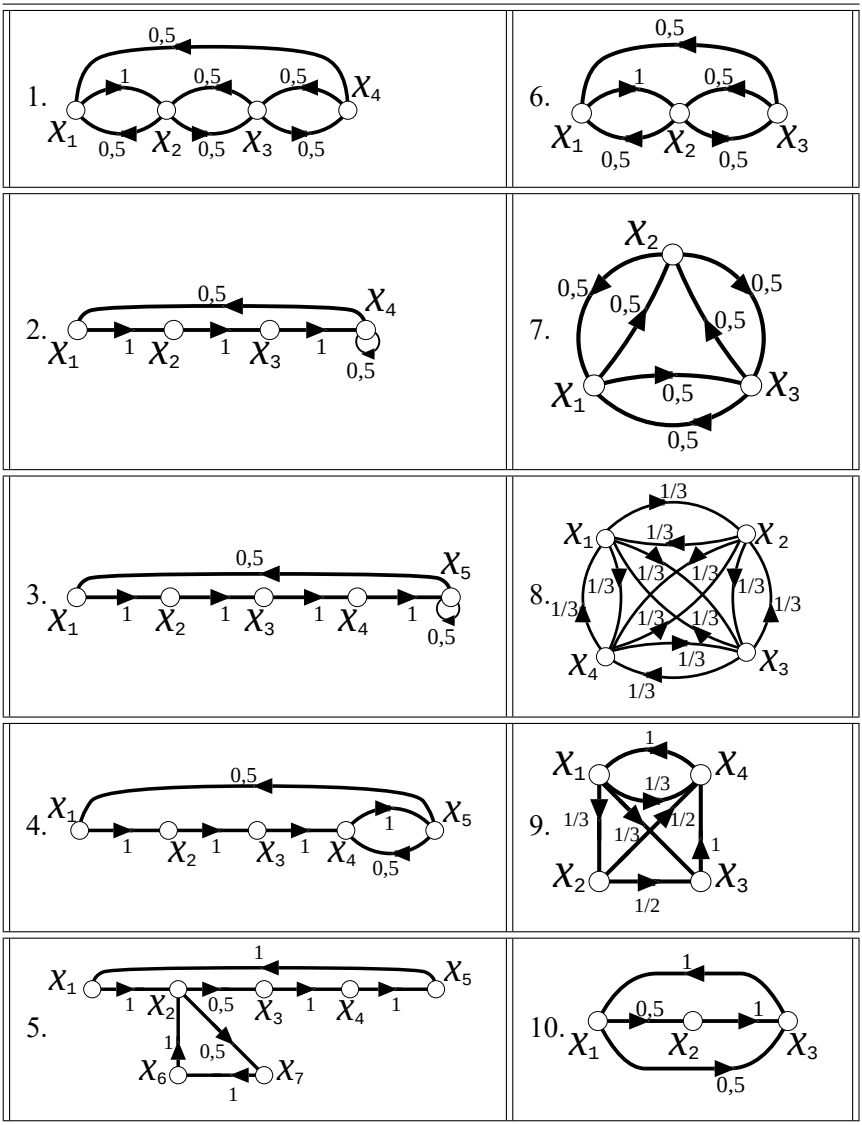
   (b) Implemente tal algoritmo.

   (c) Mostre que a complexidade de tempo é de $O(f(n) \log k)$, onde $f(n)$ denota a complexidade de tempo da multiplicação de duas matrizes quadradas de ordem n.

2. Para cada digrafo a seguir.

   (a) Determine a cadeia de Markov correspondente.

   (b) Verifique justificando se cada cadeia é ou não é ergódica. E neste caso, determine experimentalmente o número de iterações e uma aproximação para a distribuição limitante, usando um erro de $\varepsilon = 10^{-7}$.



3. São dadas duas bolas em uma urna, uma vermelha e uma azul. Uma extração é feita e uma bola vermelha ou azul é retirada e retornada à urna. A cada extração, a probabilidade de uma bola retornada ter a mesma cor da última extraída é de 80%. Qual a probabilidade da quinta extração ser de uma bola vermelha?

4. Uma mensagem binária, 0 ou 1, será enviada de um canal de sinal composto por vários estágios, onde a transmissão em cada estágio está sujeita a uma probabilidade fixa de erro p. Seja p_{ij} a probabilidade de ocorrer a transmissão do bit i e o bit j ser recebido no estágio seguinte. Suponha que $C = (X = \{0, 1\}, T, p)$ seja uma cadeia de Markov com probabilidades de erro $p_{01} = \alpha > 0$ e $p_{10} = \beta > 0$.

   (a) Determine a probabilidade de, em 5 estágios, termos uma transmissão correta.

   (b) Determine a distribuição estacionária dessa cadeia de Markov.

5. O país de Oz é maravilhoso por muitas coisas, mas não pelo seu clima. Nunca há dois dias bons (B) consecutivos. Em um dia bom, há uma probabilidade igual de nevar (N) ou chover (C) no dia seguinte. Se em um dia está nevando ou chovendo, a probabilidade de manter o mesmo tipo de clima no dia seguinte é $1/4$, e a probabilidade de ter um dia bom no dia seguinte é $1/2$. Determine:

   (a) A distribuição estacionária do clima em Oz.

   (b) Dado que temos um dia bom na segunda, a probabilidade de termos novamente um dia bom no próximo sábado.

6. Suponha que o clima em Oz seja um pouco diferente. Sempre que ocorre um dia bom, segue-se de um dia de chuva ou neve com mesma probabilidade. Mas,

   - A probabilidade de um dia bom seguir a dois dias não bons alternados é de 0,4.
   - A probabilidade de um dia bom seguir a dois dias não bons idênticos é de 0,3.
   - A probabilidade de um dia não bom seguir a um dia não bom de mesmo tipo é de 0,2.
   - Nunca ocorrem 2 dias bons em 3 dias consecutivos.

   Pergunta-se qual a probabilidade de termos um dia bom no próximo sábado, dado que

   (a) No sábado anterior fez tempo bom.

(b) Choveu no sábado e domingo anteriores.

7. Um conjunto com cinco dados é lançado. Todas os dados que saem 6 são removidos do conjunto. Essa operação é repetida iterativamente até que todos os dados são removidos. Pergunta-se

   (a) Qual a cadeia de Markov $C = (X, T, p)$ correspondente?

   (b) Qual a probabilidade de, após 6 lançamentos, todos os dados terem sidos removidos?

   (c) Qual a distribuição estacionária de C?

   (d) Qual o número esperado de lançamentos para que todos os dados sejam removidos?

8. Duas equipes, A e B, devem jogar uma melhor de 3 séries de jogos. Suponha que os resultados de jogos sucessivos sejam independentes e cada um seja ganho por A com probabilidade $\frac{2}{3}$.

   (a) Determine a cadeia de Markov $C = (X, T, p)$ correspondente.

   (b) Determine a probabilidade de A vencer.

   (c) Dado que B marcou primeiro, qual a probabilidade de A virar o jogo?

   (d) Dado que A marcou primeiro, qual a probabilidade de B virar o jogo?

7.7 Notas Bibliográficas

Andrei Andreyevich Markov nasceu na Rússia em 1856. Nos artigos seminais (Markov 1906a,b), o autor descreveu suas cadeias e mostrou a distribuição estacionária das cadeias de Markov ergódicas.

As cadeias de Markov foram rapidamente absorvidas pela comunidade matemática. P. Ehrenfest e T. Ehrenfest (1906) usaram as cadeias de Markov para descrever o comportamento dos gases. O problema foi descrito originalmente com urnas. Dobrow (2016) adaptou esse problema usando uma abordagem com cachorros e pulgas, a qual é apresentada no Exemplo 6.

Poincaré (1912) devotou seis seções de um livro para estudar o problema do embaralhamento de cartas. Ele mostrou que, sob alguns critérios, qualquer método de embaralhamento conduzia as cartas a estarem embaralhadas sem qualquer ordem previsível. Esse trabalho seguia em linha com a teoria mais geral, desconhecida por Poincaré, das cadeias de Markov, seis anos antes publicada por Markov.

Em (Markov 1913, 2006) o autor analisou as frequências de letras em um livro russo determinando a cadeia associada a frequências de vogais e consoantes desse livro. Este estudo é o modelo base para os exemplos climáticos apresentados no capítulo.

Em Shannon e Weaver (1949), consideram-se cadeias de Markov para a simulação automática da prosa inglesa. Os autores primeiro consideraram a matriz de transição das letras e depois a das palavras. Essa foi a base para os algoritmos de autocorreção e autocompletação que usamos hoje nos nossos celulares. Mais tarde, Kolmogorov (1956) também considerou os conceitos de Shannon e Weaver (1949) para algoritmos de autocompletação.

Doeblin (1938) provou que se uma cadeia de Markov é ergódica, então existe uma única distribuição estacionária que é também a distribuição limitante da cadeia.

De acordo com Ventsel (1992) durante a Segunda Guerra Mundial, Kolmogorov contribuiu para o esforço de guerra russo, aplicando a teoria estatística à artilharia, desenvolvendo um esquema de distribuição estocástica de balões de barragem com o objetivo de ajudar a proteger Moscou dos bombardeiros alemães.

Kolmogorov introduziu e estudou um conjunto particular de processos de Markov conhecidos como processos de difusão, em que derivou um conjunto de equações diferenciais que descrevem o processo. Independente do trabalho de Kolmogorov, Sydney Chapman derivou em um artigo de 1928 uma equação, agora chamada de equação de Chapman–Kolmogorov, enquanto estudava o movimento Browniano. Segundo Ventsel (1992), em um artigo de 1931, Kolmogorov (1931) desenvolveu grande parte da teoria inicial de processos de Markov de tempo contínuo. Kolmogorov foi parcialmente inspirado pelos trabalhos Markov (1906a), Markov (1906b) e Bachelier (1900) sobre flutuações no mercado de ações, assim como pelo trabalho de Norbert Wiener sobre o modelo do movimento Browniano proposto por Einstein (1911).

O estudo que apresentamos, na Seção 7.3.2, sobre modelos estocásticos transformados em cadeias de Markov, foi baseado em um estudo similar no livro de Ross (2006) (página 193). Alguns exercícios foram baseados em exercícios do livro de Fernandez (1975), publicado no 10^o Colóquio Brasileiro de Matemática do IMPA, outros em alguns dos livros de Kemeny, Snell e Thompson (1974), Karlin e Taylor (2012) e Ross (2014). Alguns livros recentes sobre modelagem preditiva são de Kuhn e K. Johnson (2013) e de Sarma (2017).

8 Passeios Aleatórios

8.1 Introdução

Passeios aleatórios são um dos objetos básicos estudados na ciência de dados. A motivação para seu estudo vem de observações de vários movimentos aleatórios no mundo real, dentro da química, física e biologia. Originalmente, os passeios aleatórios foram estudados em associação ao movimento errático dos grãos de pólen imersos em um fluido - chamado de movimento Browniano. O nome *passeio aleatório* foi dado em 1905 em um artigo sobre a propagação de mosquitos na revista Nature.

As aplicações dos passeios aleatórios incluem as supercordas, o movimento dos gases, o mercado de ações e muitas outras. Mais recentemente, os passeios aleatórios celebrizaram-se por sua aplicação na concepção e nos fundamentos do Algoritmo Pagerank usado como principal ferramenta na máquina buscadora da Google.

Dessa forma, a relação mais pronunciada entre os passeios aleatórios com a ciência de dados se encontra na organização dos dados das bilhões de páginas da rede web. Mas, os passeios aleatórios possuem muitas outras aplicações. Tendo em vista que os passeios aleatórios são parte da teoria das cadeias de Markov, a estrutura intrínseca preditiva dessas cadeias é herdada pelos passeios aleatórios em

importantes modelos combinatórios e no projeto de algoritmos para problemas de otimização.

Iniciamos o capítulo com uma formulação do Algoritmo Pagerank, apresentada em detalhe com um exemplo de aplicação. Estudamos a classe formada pelos passeios aleatórios de dimensão 1. Nesse contexto, apresentamos o problema da ruína do jogador e um algoritmo aleatório para o problema 2-SAT.

8.2 O Algoritmo Pagerank

Quando usamos a internet em busca de informações, frequentemente pesquisamos um determinado tema através de uma *chave de busca.* As *máquinas buscadoras* são algoritmos que produzem uma classificação das páginas da internet de acordo com a sua ordem de importância, associadas à chave de busca.

Em um modelo simplificado, as máquinas buscadoras fazem a classificação das páginas considerando dois passos principais. O primeiro passo consiste em determinar o conjunto das páginas da rede que possuem a chave de busca. O segundo passo consiste em classificar as páginas da rede de acordo com sua importância. A classificação possui complexidade de tempo muito grande. Sua atualização é realizada periodicamente pelos servidores das máquinas buscadoras, utilizando as chaves de busca mais comumente procuradas, armazenando a classificação para posterior utilização.

Fisicamente, a rede web corresponde a um grande digrafo U, onde os vértices são as páginas da internet e as arestas (i, j) correspondem a um *link* da página i para a página j, o qual permite acessar a página j a partir da página i. Assim, as páginas associadas a uma chave de busca formam um subdigrafo D de U, induzido pelas páginas que contêm a chave de busca. Esse digrafo pode ser qualquer, onde propriedades como conexidade forte, aciclicidade e existência de laços, podem ou não ocorrer.

O tratamento teórico-matemático desse digrafo é feito nos parágrafos seguintes.

Definiremos um *passeio aleatório* $P(D)$ em $D = (V, E)$ como uma sequência $P(D) = (v(1), v(2), \ldots, v(k), \ldots)$, gerada a partir do vértice $v_1 = v(1) \in V$, formada por vértices $v_i \in V$, com $d^+(v_i) > 0$, selecionando v_{i+1} dentre os vértices divergentes de v_i com a probabilidade dada por

$$q\Big(v(t+1) = v_{i+1}, v_{i+1} \in N^+(v_i) \mid v(t) = v_i\Big) = \frac{1}{d^+(v_i)}, i \geqslant 1$$

atravessando a aresta $e_i = (v_i, v_{i+1})$ do vértice v_i para o vértice corrente v_{i+1}, iterando o processo.

O passeio aleatório correspondente com a probabilidade de transição $q_{ij} = q\Big(v(t+1) = v_j, v_j \in N^+(v_i) \mid v(t) = v_i\Big)$ de mudança de estado v_i para v_j é equiprovável no conjunto $N^+(v_i)$, isto é,

$$q_{ij} = q\Big(v(t+1) = v_j \mid v(t) = v_i\Big) = \begin{cases} \frac{1}{d^+(v_i)}, & \text{se } (i,j) \in E \\ 0, & \text{se } (i,j) \notin E \end{cases}$$

Esse passeio não necessariamente corresponde a uma cadeia de Markov, porque os vértices sumidouros de D possuem linhas nulas na matriz de probabilidade.

Observe que, muitas vezes o digrafo D pode não ter uma distribuição limitante. Assim, o Algoritmo Pagerank vai, muitas vezes, encontrar a distribuição limitante em um digrafo obtido a partir de D, como será mostrado a seguir. Esse digrafo corresponderá a uma cadeia de Markov ergódica $C = (V, T, p)$ que, pelo Teorema 7.8, possui a distribuição limitante igual à distribuição estacionária e será a saída do Pagerank.

O argumento para modificar o digrafo D obtendo a cadeia de Markov ergódica $C = (V, T, p)$ se baseia em um usuário que navega aleatoriamente em todas as páginas da internet. Assim, esse usuário se move pelas páginas do digrafo D de duas maneiras: segue algum *link* da página atual para uma nova página ou, com alguma probabilidade, ele se desloca para uma página qualquer de D.

Para produzir essa cadeia de Markov a partir do digrafo $D = (V, E)$, vamos considerar os 3 passos:

1. Para cada aresta $(u, v) \in E$, definiremos a probabilidade $q_{uv} = 1/d^+(u)$, produzindo o grafo ponderado com pesos $D = (V, E, q)$. A probabilidade q determina como o caminho aleatório se desloca pelos *links* do digrafo D.

2. Para cada vértice sumidouro $u \in V$, definiremos para cada $v \in V$ uma aresta adicional ligando u até v e a probabilidade: $r_{uv} = 1/n$, nas demais arestas $(u, v) \in E$, teremos $r_{uv} = q_{uv}$. Isto é,

$$r_{uv} = \begin{cases} 1/n, & \text{se } u \text{ é um sumidouro de } D \text{ e } v \in V \\ q_{uv}, & \text{se } (u,v) \in E \end{cases}$$

   Essa nova probabilidade garante que $C^1 = (V, T, r)$ seja uma cadeia de Markov.

3. Definiremos a probabilidade de teletransporte $0 < \beta < 1$ com a qual o passeio percorre C^1, e com probabilidade $(1 - \beta)$ o passeio se move para um outro vértice qualquer de V. Isto é,

$$p_{uv} = r_{uv}\beta + 1/n(1 - \beta)$$

A cadeia de Markov $C^2 = (V, T, p)$ é ergódica. E a distribuição limitante correspondente pode ser calculada pelo método da potência.

Antes de definir o Algoritmo Pagerank, definiremos um pequeno exemplo com um digrafo D, o qual corresponde ao digrafo induzido pelo conjunto de páginas que possuem uma dada chave de busca. O digrafo D será gradualmente modificado, de acordo com os passos do algoritmo, atendendo aos parâmetros necessários do Pagerank em estabelecer uma classificação satisfatória. Por satisfatória, entendemos a apresentação da classificação em ordem decrescente de relevância e um baixo tempo de execução do algoritmo. Ao final da seção, determinamos a classificação dos vértices desse digrafo.

Exemplo 1. No exemplo da Figura 8.1, mostramos um digrafo $D = (V, E)$, e sua matriz $M_{n\times n}$ de adjacências.

$$M_{7\times 7} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array}$$

$$Q = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \left[\begin{array}{cccccc} \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{array}\right] \end{array}$$

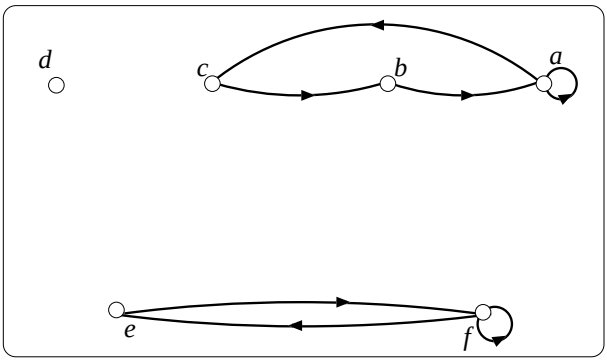


Figura 8.1: Digrafo $D = (V, E)$

Para obter a cadeia de Markov $C^1 = (V, T, r)$, a partir de $D = (V, E, q)$, alteramos as linhas de M de modo que vértices sumidouro, que por definição possuem as linhas nulas, tenham sempre a probabilidade associada igual a $\frac{1}{n}$, definindo assim a matriz de probabilidade R. No exemplo, o vértice d é um sumidouro. Ficaríamos, então, com a cadeia de Markov $C_1 = (V, T, r)$ cujo digrafo $D^1 = (V, E^1, r)$ correspondente é mostrado na Figura 8.3 e a matriz R a seguir.

$$R = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \left[\begin{array}{cccccc} 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \end{array}\right] \end{array}$$

Para obter a cadeia de Markov ergódica $C^2 = (V, T, p)$ a partir de $C^1 = (V, T, r)$, o Algoritmo Pagerank considera ainda uma probabilidade adicional β chamada de *teletransporte* que define se o passeio permanece em $C^1 = (V, T, r)$ com probabilidade β, ou se com probabilidade $(1 - \beta)$ o caminho se desloca do vértice corrente para um vértice qualquer $v \in V$. O digrafo associado a C^2 é chamado de digrafo *Google*. Normalmente, o valor de $\beta = 0{,}85$ é considerado pelo Google. Esse valor foi escolhido experimentalmente porque, para um número da ordem de dezenas de milhões de páginas, o método da potência converge em

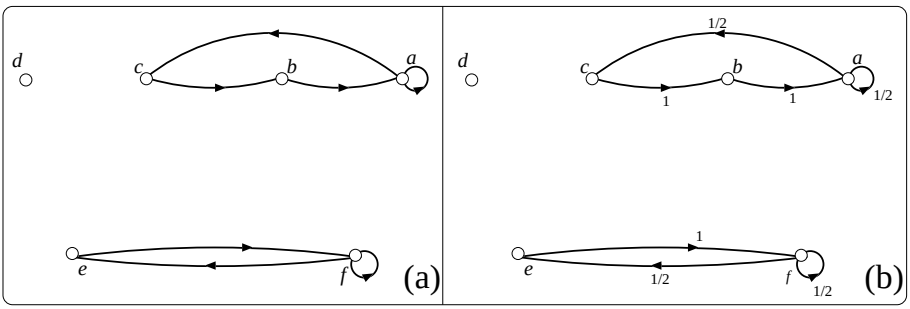


Figura 8.2: Digrafo $D = (V, E, q)$ com pesos q em (b), obtido a partir do digrafo na Figura 8.1 em (a).

torno de algumas dezenas de iterações.

Dessa forma, definimos a probabilidade p como

$$p_{ij} = \beta r_{ij} + (1 - \beta)1/n$$

que, em forma matricial, a matriz de probabilidade é expressa como

$$P = \beta R + (1 - \beta)\left[1/n\right]_{n \times n}$$

Em seguida, apresentamos a matriz P e o digrafo *Google* na Figura 8.4.

Observe que o digrafo Google é completo com laços e pesos correspondentes aos valores de probabilidades, definidos pelo Algoritmo do Pagerank.

No exemplo, teremos:

$$P = \beta \begin{bmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \end{bmatrix} + (1 - \beta) \begin{bmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{bmatrix}$$

$$P = \begin{bmatrix} 0{,}425 + 0{,}025 & 0{,}025 & 0{,}45 & 0{,}025 & 0{,}025 & 0{,}025 \\ 0{,}85 + 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 \\ 0{,}025 & 0{,}875 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}875 \\ 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}45 & 0{,}45 \end{bmatrix} =$$

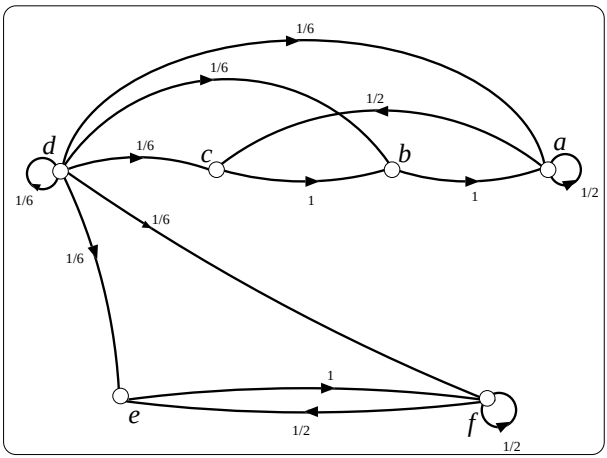


Figura 8.3: Cadeia de Markov $C^1 = (V, T, r)$, obtida a partir do digrafo $D = (V, E, q)$ da Figura 8.2.

$$\begin{bmatrix} 0{,}45 & 0{,}025 & 0{,}45 & 0{,}025 & 0{,}025 & 0{,}025 \\ 0{,}875 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 \\ 0{,}025 & 0{,}875 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}875 \\ 0{,}025 & 0{,}025 & 0{,}025 & 0{,}025 & 0{,}45 & 0{,}45 \end{bmatrix}$$

Após 30 iterações, com o método da Potência, determinamos

$$\pi = \Big(0{,}28, \quad 0{,}16, \quad 0{,}15, \quad 0{,}03, \quad 0{,}13, \quad 0{,}25\Big)$$

tal que $||\pi P - \pi|| < \varepsilon$, para $\varepsilon = 10^{-7}$ a distribuição estacionária do grafo Google correspondente.

A classificação das páginas pelo Pagerank se dá pela ordem decrescente da distribuição estacionária. No exemplo, a classificação por ordem de relevância é a, f, b, c, e, d. O grafo Google é ilustrado na Figura 8.5 com um diagrama para o digrafo D da Figura 8.1, onde o tamanho da área de cada vértice reflete sua classificação.

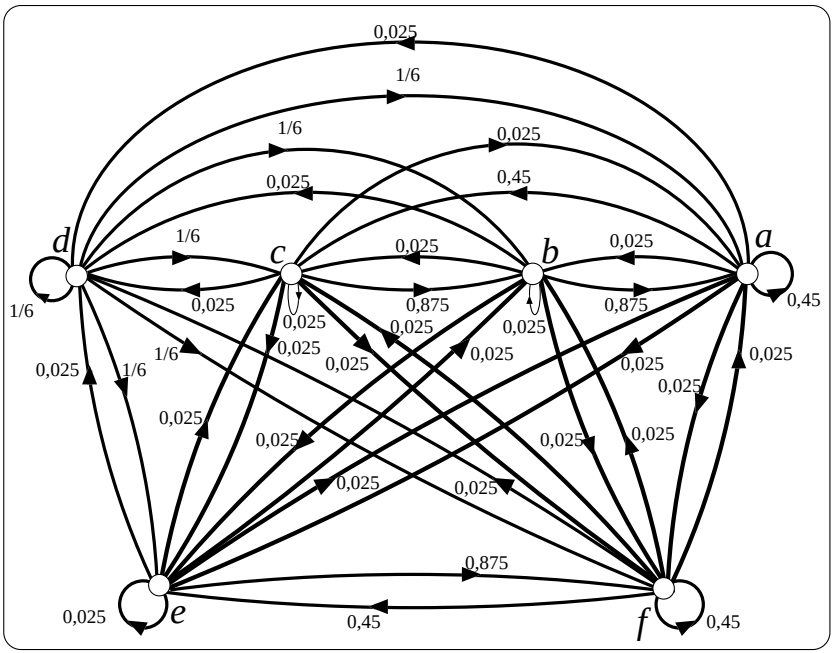


Figura 8.4: Digrafo Google associado à cadeia de Markov ergódica $C^2 = (V, T, p)$, obtida a partir da cadeia de Markov $C^1 = (V, T, r)$ da Figura 8.3, com constante de teletransporte $\beta = 0{,}85$. As arestas adicionais correspondem ao teletransporte.

O Algoritmo 8.1 descreve formalmente o método do Pagerank apresentado. Observamos que o método usa o Algoritmo 7.1 para cálculo da distribuição estacionária de uma cadeia de Markov ergódica, definido no Capítulo 7.

8.3 Passeios aleatórios de dimensão 1

Dada uma cadeia de Markov $C = (V, T, p)$ e um digrafo associado $D = (V, E, p)$, um passeio aleatório $P(D)$ em D é de *dimensão 1* se existe uma ordenação $(v_1, v_2, \ldots, v_n)$ para os vértices de V tal que para todo par $1 \leqslant i, j \leqslant n$ com $|i - j| \geqslant 2$, $p_{ij} = 0$. Nesta seção, veremos duas aplicações de passeios aleatórios de dimensão 1. A primeira consiste do problema da ruína do jogador e a

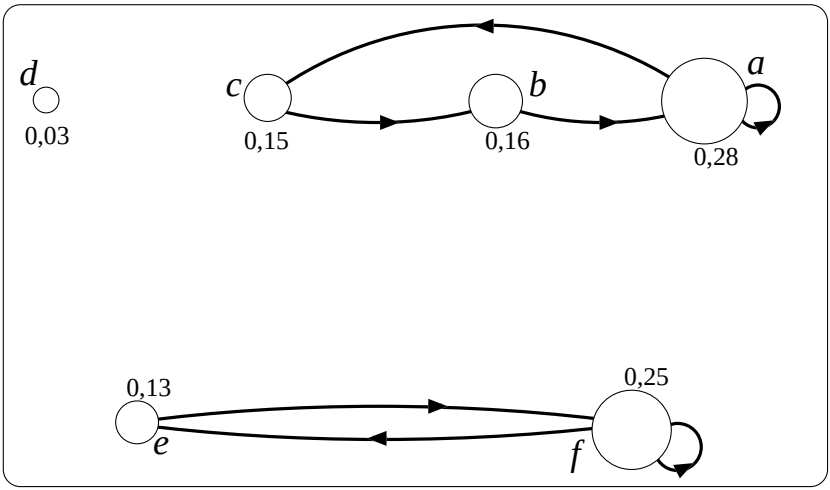


Figura 8.5: Pagerank dos vértices do digrafo D da Figura 8.1.

segunda um algoritmo randomizado para o problema 2-SAT.

8.3.1 A ruína do jogador

O problema que vamos considerar aqui chama-se a *ruína do jogador*. Dizemos que um jogador é *compulsivo* se ele realiza turnos de apostas até quebrar o cassino ou a banca ou, caso contrário, até perder todas as suas fichas.

A versão que usamos aqui assume que um jogador compulsivo tem uma quantidade inteira positiva de $i > 0$ fichas, e que o cassino quebra quando o jogador ganha $n - i$ fichas, isto é quando o jogador possui um total de n fichas. A cada rodada o jogador aposta 1 ficha. Como resultado, ganha 1 ficha adicional com probabilidade $0 < p < 1$, ou perde 1 ficha com probabilidade $1 - p$. O problema consiste em determinar a probabilidade do jogador perder suas i fichas, isto é, a probabilidade do jogador se tornar arruinado.

Teorema 8.1. *Dado que um jogador compulsivo possui $0 \leqslant i \leqslant n$ fichas, que o cassino quebra se o jogador ganha $n - i$ fichas adicionais, que a cada aposta o jogador ganha 1 ficha com probabilidade $0 < p < 1$ e que o jogador perde 1 ficha com probabilidade $1 - p$, então a probabilidade de ruína $p_{ruína}$ do jogador perder todas as suas i fichas é igual a*

Algoritmo 8.1 Algoritmo para determinar a distribuição estacionária π de um digrafo $D = (V, E)$ com $|V| = n$, taxa de teletransporte β, e erro ε

função PAGERANK($D = (V, E)$,β, ε):
  $z = (z_1, \ldots, z_n) \leftarrow (0, \ldots, 0)$
  para $i \leftarrow 1$ até n **:**
    para $j \leftarrow 1$ até n **:**
      se $(v_i, v_j) \in E$ **então**
        $q_{ij} \leftarrow \frac{1}{d^+(v_i)}$
        $z_i \leftarrow 1$
      senão
        $q_{ij} \leftarrow 0$
  para $i \leftarrow 1$ até n **:**
    se $z_i = 0$ **então**
      para $j \leftarrow 1$ até n **:**
        $q_{ij} \leftarrow \frac{1}{n}$
  para $i \leftarrow 1$ até n **:**
    para $j \leftarrow 1$ até n **:**
      $p_{ij} \leftarrow \beta q_{ij} + (1 - \beta)1/n$
  retornar DIST-ESTAC$\big(C = (V, T, p), \varepsilon\big)$

$$p_{ruína} = \begin{cases} 1 - i/n, & se\ p = 1/2 \\ 1 - \frac{1-((1-p)/p)^i}{1-((1-p)/p)^n}, & se\ p \neq 1/2 \end{cases}$$

Demonstração. Vamos considerar uma cadeia de Markov $C = (X, T, p)$ com conjunto de estados $X = \{x_0, x_1, \ldots, x_n\}$, onde cada estado $x_i \in X$ corresponde à situação na qual o jogador tem i fichas. Vamos considerar a matriz de transição correspondente

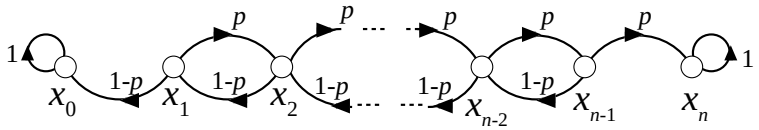


Figura 8.6: Digrafo $D = (X, E, p)$ associado à cadeia de Markov do problema da ruína do jogador.

$$P = \begin{array}{c} \\ x_0 \\ x_1 \\ \vdots \\ x_{i-1} \\ x_i \\ x_{i+1} \\ \vdots \\ x_{n-1} \\ x_n \end{array} \begin{array}{c} \begin{array}{cccccccccc} x_0 & x_1 & \dots & x_{i-1} & x_i & x_{i+1} & \dots & x_{n-1} & x_n \end{array} \\ \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ 1-p & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & p & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 1-p & 0 & p & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 1-p & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & p \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \end{array}$$

Existem dois únicos estados x_0 e x_n absorventes correspondentes, respectivamente, aos casos em que o jogador está arruinado, ou quebra a banca. Os demais estados são todos transientes. A Figura 8.6 apresenta o digrafo $D = (X, E, p)$ associado à cadeia de Markov.

Como estamos lidando com um jogador compulsivo, vamos calcular a probabilidade $p_{\text{ruína}}$ da ruína, como o complemento da probabilidade de quebrar a banca. Supondo que no tempo t o estado corrente seja absorvente, a probabilidade que a absorção tenha ocorrido no estado x_0 é de

$$P\Big(X(t) = x_0 \mid X(0) = x_i\Big) = 1 - P\Big(X(t) = x_n \mid X(0) = x_i\Big)$$

Usaremos o princípio da probabilidade total para determinar $p\big(X(t) = x_n \mid X(0) = x_i\big)$

$$P\Big(X(t) = x_n \mid X(0) = x_i\Big) =$$

$$\sum_{j=0}^{n} P\big(X(t) = x_n \mid X(1) = x_j\big).P\big(X(1) = x_j \mid X(0) = x_i\big)$$

Como estamos com um passeio aleatório de dimensão 1, a partir do estado x_i só podemos ir para o estado x_{i+1} com probabilidade p ou para o estado x_{i-1} com probabilidade $1-p$. Dessa forma, o último somatório se escreve

$$P\Big(X(t) = x_n \mid X(1) = x_{i+1}\Big).p + P\Big(X(t) = x_n \mid X(1) = x_{i-1}\Big).(1-p)$$

Chamando

$$p_i = P\Big(X(t) = x_n \mid X(0) = x_i\Big)$$

Temos que

$$p_i = p_{i+1}p + p_{i-1}(1-p)$$

Assim:

$$(p_{i+1} - p_i)p = (p_i - p_{i-1})(1-p)$$

Ou ainda,

$$(p_{i+1} - p_i) = \frac{1-p}{p}(p_i - p_{i-1})$$

Olhe agora que $p_0 = 0$ e que $p_n = 1$, pois x_0 e x_n são estados absorventes. Assim, temos

$$p_2 - p_1 = \frac{1-p}{p}p_1$$

Usando esta última igualdade temos

$$p_3 - p_2 = \frac{1-p}{p}(p_2 - p_1) = \left(\frac{1-p}{p}\right)^2 p_1$$

Por indução, e tomando a soma telescópica

$$p_i - p_1 = (p_i - p_{i-1}) + \ldots + (p_3 - p_2) + (p_2 - p_1) = \sum_{j=1}^{i-1}\left(\frac{1-p}{p}\right)^j p_1$$

E assim,

$$p_i = p_1 \sum_{j=0}^{i-1}\left(\frac{1-p}{p}\right)^j \tag{8.1}$$

Neste momento, temos que selecionar 2 casos:

1. Caso em que $p = \frac{1}{2}$ - Neste caso, a partir da Igualdade 8.1, temos que

$$p_n = p_1 \sum_{j=0}^{n-1} 1 = p_1 n$$

E assim,

$$p_1 = p_n/n = 1/n$$

E usando de novo a Igualdade 8.1, temos que $p_i = i/n$. E assim,

$$p_{\text{ruína}} = 1 - p_i = 1 - i/n$$

2. Caso em que $p \neq \frac{1}{2}$ - Neste caso, a partir da Igualdade 8.1 temos que

$$p_1 = \frac{p_n}{\frac{\left(1-\left(\frac{1-p}{p}\right)^n\right)}{\left(1-\left(\frac{1-p}{p}\right)\right)}} = \frac{1}{\frac{\left(1-\left(\frac{1-p}{p}\right)^n\right)}{\left(1-\left(\frac{1-p}{p}\right)\right)}} = \frac{\left(1-\left(\frac{1-p}{p}\right)\right)}{\left(1-\left(\frac{1-p}{p}\right)^n\right)}$$

Novamente, pela Igualdade 8.1, temos que

$$p_i = \frac{\left(1-\left(\frac{1-p}{p}\right)^i\right)}{\left(1-\left(\frac{1-p}{p}\right)^n\right)}$$

E assim,

$$p_{\text{ruína}} = 1 - p_i = 1 - \frac{\left(1-\left(\frac{1-p}{p}\right)^i\right)}{\left(1-\left(\frac{1-p}{p}\right)^n\right)}$$

□

8.3.2 Algoritmo randomizado para 2-SAT

Nesta seção, vamos apresentar um algoritmo randomizado para o problema de lógica 2-SAT. Como visto no Capítulo 1, o problema de decisão SATISFATIBILIDADE é definido como

SATISFATIBILIDADE (SAT)
DADOS: Uma expressão booleana E na FNC
QUESTÃO: E é satisfatível?

Uma instância de SAT será representada por $I = (U, C)$, onde $U = \{u_1, u_2 \ldots, u_n\}$ é um conjunto de variáveis lógicas e E é uma expressão lógica na FNC sob U.

Um exemplo de instância satisfatível é

$$I = (U, E) = \Big(\{u_1, u_2, u_3\}, (u_1 \vee u_2 \vee u_3) \wedge (u_1 \vee \overline{u_2} \vee \overline{u_3}) \wedge (\overline{u_1} \vee \overline{u_2} \vee \overline{u_3})\Big)$$

pois existe

$$\eta : U \to \{V, F\} \text{ com } \eta(u_1) = \eta(\overline{u_2}) = \eta(\overline{u_3}) = V$$

que faz com que os literais verdadeiros $u_1, \overline{u_2}, \overline{u_3}$ presentes, respectivamente, na primeira, na segunda e na terceira cláusulas.

Um exemplo de instância não satisfatível é

$$I = (U, E) = \Big(\{u_1, u_2\}, (u_1 \vee u_2) \wedge (u_1 \vee \overline{u_2}) \wedge (\overline{u_1} \vee u_2) \wedge (\overline{u_1} \vee \overline{u_2})\}\Big)$$

Mostramos em seguida uma aplicação importante das cadeias de Markov com uma construção de um algoritmo randomizado que resolve instâncias de 2-SAT. A partir de uma atribuição de verdade inicial arbitrária, o algoritmo verifica se a atribuição é satisfatível. Caso contrário, o algoritmo escolhe uma cláusula não satisfeita e modifica o valor de um dos seus literais. Essa operação é repetida iterativamente no máximo $2mn^2$ vezes. O algoritmo retorna uma atribuição de verdade satisfatível ou retorna que I não é satisfatível caso nenhuma atribuição satisfatível seja encontrada. O Algoritmo 8.2 formaliza tal método.

Teorema 8.2. *Se $I = (U, E)$ é uma instância satisfatível de 2-SAT com $|U| = n$ e E possuindo m cláusulas, então*

1. *o número esperado de passos para o Algoritmo 8.2 encontrar uma atribuição de verdade satisfatível é de n^2 passos.*

2. *com probabilidade no máximo $\frac{1}{2^k}$, o Algoritmo 8.2 não retorna uma atribuição de verdade satisfatível.*

Algoritmo 8.2 Algoritmo randomizado para 2-SAT

Dados: Uma instância 2-SAT $I = (U, E)$, com $|U| = n$, E possuindo m cláusulas e $k \in \mathbb{N}$ um inteiro positivo

função SATISFAZ($I = (U, E)$):
  para $i \leftarrow 1$ **até** n **:**
    $\eta(u_i) \leftarrow V$
  $i \leftarrow 1$
  enquanto $\Big((i \leqslant 2kn^2)$ e (η não satisfatível)$\Big)$ **:**
    escolher aleatoriamente uma cláusula não satisfeita $c = (x_j \vee x_k) \in C$
    escolher $\ell \in \{j, k\}$
    $\eta(u_\ell) \leftarrow$ **não** $\eta(u_\ell)$
    $i \leftarrow i + 1$
  se η é satisfatível **então**
    retornar ("sim", η)
  senão
    retornar ("não")

Demonstração. Suponha que $I = (U, E)$ é uma instância satisfatível de 2-SAT com $|U| = n$ e E possuindo m cláusulas. Seja $\xi : U \to \{V, F\}$ uma atribuição de verdade satisfatível para I. Lembre que η é a atribuição de verdade gerada pelo algoritmo. Queremos analisar a diferença entre η e ξ e como o algoritmo modifica η para se aproximar de ξ. Mostraremos que o tempo esperado para η se tornar igual a ξ é n^2.

Vamos dizer que η *concorda* com ξ na variável $u \in U$ se $\eta(u) = \xi(u)$. E dizemos que η e ξ *concordam* se para todo $u \in U$, $\eta(u) = \xi(u)$.

Seja $c = (x_a \vee x_b)$ uma cláusula falsa na atribuição corrente η, isto é, $\eta(x_a) = \eta(x_b) = F$. Sabemos que pode ocorrer que ξ concorde com no máximo 1 dos valores de $\eta(x_a), \eta(x_b)$, pois c é satisfeita por ξ. Vamos analisar os casos possíveis:

1. $\xi(x_a) = \xi(x_b) = V$, neste caso com probabilidade 1, na próxima iteração do algoritmo η concorda com mais uma variável de ξ.

2. $\xi(x_a) = V$ e $\xi(x_b) = F$, neste caso com probabilidade $\frac{1}{2}$, na próxima iteração do algoritmo η concorda com mais uma variável de ξ, quando a variável x_a é escolhida para trocar de valor.

3. $\xi(x_a) = F$ e $\xi(x_b) = V$, neste caso com probabilidade $\frac{1}{2}$, na próxima

iteração do algoritmo η concorda com mais uma variável de ξ, quando a variável x_b é escolhida para trocar de valor.

Observamos que a probabilidade de que, na iteração seguinte do algoritmo, as atribuições de verdade concordarem em uma variável adicional é maior ou igual a $1/2$. No entanto, demonstraremos que a cadeia de Markov $C = (X, T, p)$ equivalente, porém mais pessimista onde a probabilidade seja exatamente de $1/2$, possui tempo esperado para que η e ξ concordem em n^2 passos.

Consideraremos a cadeia de Markov $C = (X, T, p)$, onde $X = \{x_0, x_1, \ldots, x_n\}$, tal que o estado $x_i \in X$ representa o número i de variáveis que concordam em η e ξ e P é dado por

$$P = \begin{array}{c} \\ x_0 \\ x_1 \\ \vdots \\ x_{i-1} \\ x_i \\ x_{i+1} \\ \vdots \\ x_{n-1} \\ x_n \end{array} \begin{array}{c} \begin{array}{cccccccccc} x_0 & x_1 & \ldots & x_{i-1} & x_i & x_{i+1} & \ldots & x_{n-1} & x_n \end{array} \\ \left[\begin{array}{cccccccccc} 0 & 1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 \\ 1/2 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & 1/2 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 1/2 & 0 & 1/2 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 0 & 1/2 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 1/2 \\ 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 1 \end{array}\right] \end{array}$$

Veja que todos os estados de C são transientes exceto o estado x_n que é absorvente. Assim, se

$$t_i = E[t \in T, X(t) = x_n \mid X(0) = x_i]$$

pelo Teorema 7.3, se Q é o conjunto de estados transientes de X e $x \in Q$, então

$$t_x = 1 + \sum_{y \in Q} p_{xy} t_y$$

Assim,

$$t_0 = 1 + p_{01} t_1 = 1 + t_1$$

e ainda

$$t_1 = 1 + p_{10} t_0 + p_{12} t_2 = 1 + \frac{1}{2} t_0 + \frac{1}{2} t_2 = 1 + \frac{1}{2} + \frac{t_1}{2} + \frac{t_2}{2}$$

Então,

$$t_1 = 3 + t_2$$

Provaremos por indução em i que, para $0 \leqslant i \leqslant n-1$, vale que

$$t_i = 2i + 1 + t_{i+1}$$

A base de indução é para $i = 0$, que resulta em mostrar que

$$t_0 = 1 + t_1$$

cuja validade foi mostrada anteriormente. A hipótese de indução é que o resultado se verifica para algum i com $0 \leqslant i \leqslant n-2$. Pelo Teorema 7.3 temos que

$$t_{i+1} = 1 + p_{(i+1)i}t_i + p_{(i+1)(i+2)}t_{i+2} = 1 + \frac{1}{2}t_i + \frac{1}{2}t_{i+2}$$

Usando a hipótese de indução, temos

$$t_{i+1} = 1 + \frac{2i+1}{2} + \frac{t_{i+1}}{2} + \frac{1}{2}t_{i+2}$$

$$t_{i+1} = \frac{2(i+1)+1}{2} + \frac{t_{i+1}}{2} + \frac{1}{2}t_{i+2}$$

E finalmente,

$$t_{i+1} = 2(i+1) + 1 + t_{i+2}$$

Calcularemos explicitamente o valor de

$$t_0 = 1 + t_1 = 1 + 3 + t_2 = \sum_{i=0}^{n-1}(2i+1) = n^2$$

Para mostrar a segunda parte, veja que η é atualizado no máximo $2mn^2$ vezes. Se A é uma variável aleatória com $A > 0$, e $a > 0$, a desigualdade de Markov nos diz que

$$P(A \geqslant a) \leqslant \frac{E[A]}{a}$$

Assim, nos primeiros $2n^2$ passos do algoritmo a probabilidade de $\eta \neq \xi$ é

$$P(X(t) \neq x_n \mid t \geqslant 2n^2) \leqslant \frac{n^2}{2n^2} = 1/2$$

Como existem k intervalos de $2n^2$ passos no algoritmo, temos, pelo princípio multiplicativo, que a probabilidade de uma atribuição de verdade satisfatível de $I = (U, E)$ não ser obtida é

$$P(X(t) \neq x_n \mid t \geqslant 2kn^2) \leqslant \frac{1}{2^k}$$

□

8.4 Exercícios

1. Dada uma moeda equilibrada que vai ser lançada um número arbitrário de vezes. Pergunta-se

   (a) Qual o número de lançamentos esperado para que tenhamos 2 caras consecutivas.

   (b) Qual o número esperado de lançamentos para que tenhamos 2 caras consecutivas, dado que o primeiro lançamento foi feito e deu cara.

2. Revisitando Lola e Rex Dois cachorros: Lola e Rex possuem n pulgas. Em cada tempo $t \in T$ uma pulga salta para um cachorro, não necessariamente para o mesmo cachorro.

   (a) Determine a matriz de probabilidade correspondente considerando o número de pulgas em um dos cachorros.

   (b) Mostre que existe uma distribuição limitante para a cadeia.

   (c) Calcule experimentalmente a distribuição estacionária da cadeia.

3. Dado um digrafo $D(V, E)$ e G o grafo subjacente de D. Dada a cadeia de Markov $C^1 = (V, T, r)$ tal que

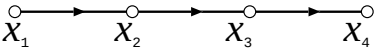
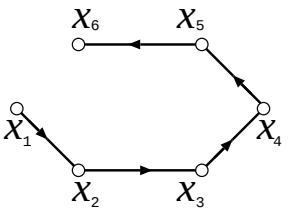
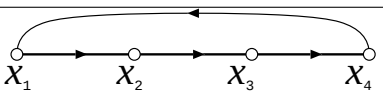
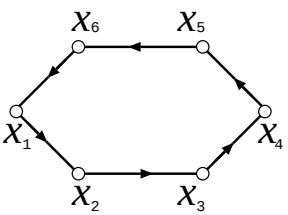
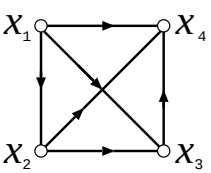
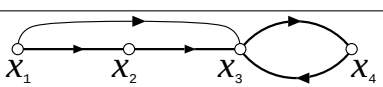
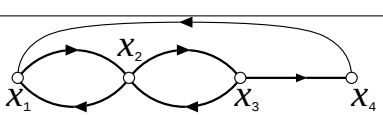
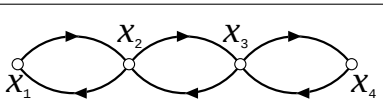
$$r_{uv} = \begin{cases} 1/n, & \text{se } u \text{ é um sumidouro de } D \text{ e } v \in V \\ q_{uv}, & \text{se } (u, v) \in E \end{cases}$$

   Prove ou refute: Se S é uma componente conexa de G, então S é uma classe comunicante de C^1.

4. Para cada digrafo a seguir.

   (a) Defina a cadeia de Markov correspondente ao digrafo Google.

   (b) Estabeleça experimentalmente cada passo do cálculo de seu pagerank.

	Digrafo	Pagerank $pr(x_i)$
1.		$pr(x_1) = 0{,}1162$ $pr(x_2) = 0{,}2149$ $pr(x_3) = 0{,}2988$ $pr(x_4) = 0{,}3701$
2.		$pr(x_1) = 0{,}0606$ $pr(x_2) = 0{,}1123$ $pr(x_3) = 0{,}1563$ $pr(x_4) = 0{,}1936$ $pr(x_5) = 0{,}2252$ $pr(x_6) = 0{,}2518$
3.		$pr(x_1) = 0{,}25$ $pr(x_2) = 0{,}25$ $pr(x_3) = 0{,}25$ $pr(x_4) = 0{,}25$
4.		$pr(x_1) = 0{,}1667$ $pr(x_2) = 0{,}1667$ $pr(x_3) = 0{,}1667$ $pr(x_4) = 0{,}1667$ $pr(x_5) = 0{,}1667$ $pr(x_6) = 0{,}1667$
5.		$pr(x_1) = 0{,}1334$ $pr(x_2) = 0{,}1712$ $pr(x_3) = 0{,}2439$ $pr(x_4) = 0{,}4512$
6.		$pr(x_1) = 0{,}0375$ $pr(x_2) = 0{,}0534$ $pr(x_3) = 0{,}4711$ $pr(x_4) = 0{,}4370$
7.		$pr(x_1) = 0{,}25$ $pr(x_2) = 0{,}3562$ $pr(x_3) = 0{,}25$ $pr(x_4) = 0{,}1437$
8.		$pr(x_1) = 0{,}1754$ $pr(x_2) = 0{,}3246$ $pr(x_3) = 0{,}3246$ $pr(x_4) = 0{,}1754$

5. Considere que em um Cassino um jogador tenha em cada jogada uma pro-

babilidade menor ou igual a $1/2$ de vencer. Mostre que se o Cassino tiver dinheiro suficiente, o apostador compulsivo sempre perde.

6. Determine a probabilidade de sucesso de um jogador que, no modelo do problema da Ruína do Jogador com uma probabilidade de vencer p por jogada, começa o jogo com R$ 10.000,00 em fichas cada uma valendo R$ 1,00:

   (a) com $p = 1/2$, decide triplicar o seu dinheiro.

   (b) com $p = 1/3$, decide triplicar o seu dinheiro.

7. Duas urnas possuem juntas 4 bolas. A cada tempo $t \geqslant 0$, uma bola é selecionada e movida para a outra urna. Pergunta-se:

   (a) Dado que uma das urnas está vazia, qual a probabilidade de que após 4 movimentos as urnas tenham a mesma quantidade de bolas?

   (b) Dado que as duas urnas tenham a mesma quantidade de bolas, qual a probabilidade de que após 4 lançamentos uma das urnas esteja vazia?

   (c) Mostre que não existe uma distribuição limitante da cadeia de Markov correspondente.

   (d) Determine uma distribuição estacionária da cadeia de Markov.

8.5 Notas Bibliográficas

O termo "passeio aleatório" foi originalmente cunhado por Pearson (1905) e Udny (1939) em 1905. Em uma carta à revista Nature, ele deu um modelo simples para descrever uma infestação de mosquitos em uma floresta. Em cada etapa de tempo, um único mosquito se move por um comprimento fixo a em um ângulo escolhido aleatoriamente. Pearson queria saber a distribuição dos mosquitos depois de várias etapas. A carta foi respondida por Lord Rayleigh, que já havia resolvido uma forma mais geral desse problema em 1880, no contexto de ondas sonoras em materiais heterogêneos.

O algoritmo de classificação de páginas Pagerank foi desenvolvido em 1998 por Brin, Motwani et al. (1998) e Brin e Page (1998), e foi usado como base para a máquina buscadora da Google (Langville e Meyer 2006).

O algoritmo do Pagerank que apresentamos é baseado na referência (Brin e Page 1998) e (Blum, Hopcroft e Kannan 2020). Uma referência sobre a ruína

do jogador é o livro de Karlin e Taylor (2012). Sobre o algoritmo para 2-SAT, referenciamos o livro de Mitzenmacher e Upfal (2005).

Bibliografia

Y. S. Abu-Mostafa (1989). "The Vapnik-Chervonenkis dimension: information versus complexity in learning". *Neural Computation* 1, pp. 312–317 (ver p. 192).

Y. S. Abu-Mostafa, M. Magdon-Ismail e H. Lin (2012). *Learning from data: a short course*. AML (ver p. 192).

G. Allan (2005). *Probability: a Graduate Course*. New York: Springer-Verlag. Zbl: 1076.60001 (ver p. 58).

N. Alon, Y. Matias e M. Szegedy (1999). "The space complexity of approximating the frequency moments". *Journal of Computer and System Sciences* 58.1, pp. 137–147. MR: 1688610. Zbl: 0938.68153 (ver p. 163).

L. Alonso, P. Chassaing, F. Gillet, S. Janson, E. M. Reingold e R. Schott (2004). "Quicksort with unreliable comparisons: a probabilistic analysis". *Combinatorics, Probability and Computing* 13.4-5, pp. 419–449. MR: 2095970. Zbl: 1097.68545 (ver p. 121).

L. Babai (1979). "Monte Carlo algorithms in graph isomorphism testing". Université de Montréal Technical Report, DMS (ver p. 97).

L. Bachelier (1900). "Théorie de la spéculation". Em: *Annales scientifiques de l'École normale supérieure*. Vol. 17, pp. 21–86. MR: 1508978. Zbl: 31.0241.02 (ver p. 217).

R. Beier e B. Vöcking (2004). "Probabilistic analysis of Knapsack core algorithms". Em: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-04)*. New Orleans, USA: ACM, pp. 461–470. MR: 2291086 (ver p. 122).

E. R. Berlekamp (1970). "Factoring polynomials over large finite fields". *Mathematics of Computation* 24.111, pp. 713–735. MR: 0276200. Zbl: 0247.12014 (ver p. 97).

J. Bernoulli (1713). *Ars Conjectandi*. Impensis Thurnisiorum, Fratrum (ver p. 58).

B. H. Bloom (1970). "Space/time trade-offs in hash coding with allowable errors". *Communications of the ACM* 13.7, pp. 422–426. Zbl: 0195.47003 (ver p. 163).

A. Blum, J. Hopcroft e R. Kannan (2020). *Foundations of Data Science*. Cambridge University Press. Zbl: 07166062 (ver pp. 163, 192, 237).

A. Blumer, A. Ehrenfeucht, D. Haussler e M. K. Warmuth (1989). "Learnability and the Vapnik-Chervonenkis dimension". *Journal of the Association for Computing Machinery* 24, pp. 929–965. MR: 1072253. Zbl: 0697.68079 (ver p. 192).

J. A. Bondy e U. S. R. Murty (2008). *Graph Theory*. Springer. MR: 2368647. Zbl: 1134.05001 (ver p. 25).

S. Brin, R. Motwani, L. Page e T. Winograd (1998). "What can you do with a web in your pocket?" *IEEE Data Engineering Bulletin* 21.2, pp. 37–47 (ver p. 237).

S. Brin e L. Page (1998). "The anatomy of a large-scale hypertextual web search engine". *Computer Networks* 30.1-7, pp. 107–117 (ver p. 237).

A. Z. Broder (1997). "On the resemblance and containment of documents". Em: *Proceedings. Compression and Complexity of Sequences*, pp. 21–29 (ver p. 163).

P. Campbell (1977). "Gauss and the eight queens problem: a study in miniature of the propagation of historical error". *Historia Mathematica* 4, pp. 397–404. MR: 0504901. Zbl: 0384.01008 (ver p. 97).

E. G. Coffman, D. S. Johnson, G. S. Lueker e P. W. Shor (1993). "Probabilistic analysis of packing and related partitioning problems". *Statistical Science* 8.1, pp. 40–47. MR: 1105768. Zbl: 0770.90031 (ver p. 122).

S. A. Cook (1971). "The Complexity of Theorem-Proving Procedures". Em: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. Ed. por M. A. Harrison, R. B. Banerji e J. D. Ullman. ACM, pp. 151–158 (ver p. 25).

T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein (2009). *Introduction to Algorithms*. 3ª ed. Massachusetts: The MIT Press, p. 1320. MR: 2572804. Zbl: 1187.68679 (ver pp. 121, 122).

S. B. Cunha, G. C. Gomes, J. I. D. Pinheiro e S. S. R. Carvajal (2012). *Probabilidade e Estatística: Quantificando a Incerteza*. Rio de Janeiro: Elsevier (ver p. 58).

J. H. Curtiss (1953). "Monte Carlo methods for the iteration of linear operators". *Journal of Mathematics and Physics* 32, pp. 209–232. MR: 0059622 (ver p. 97).

R. P. Dobrow (2016). "Markov Chains for the Long Term". Em: *Introduction to Stochastic Processes With R*. John Wiley & Sons, Ltd. Cap. 3, pp. 76–157 (ver p. 216).

V. A. Dobrushkin (2009). *Methods in Algorithmic Analysis*. Boca Raton: Taylor & Francis Goup, LLC, p. 824. MR: 2588726 (ver p. 121).

W. Doeblin (1938). "Exposé de la théorie des chaınes simples constantes de Markova un nombre fini d'états". *Mathématique de l'Union Interbalkanique* 2.77-105, pp. 78–80 (ver p. 217).

M. Durand e P. Flajolet (2003). "Loglog counting of large cardinalities". *Lecture Notes in Computer Science*, pp. 605–617. Zbl: 1266.68236 (ver p. 163).

P. Ehrenfest e T. Ehrenfest (1906). "Über eine aufgabe aus der wahrscheinlichkeitsrechnung, die mit der kinetischen Deutung der entropievermehrung zusammenhängt". *Math.-Naturwiss. Blätter* 11, p. 12 (ver p. 216).

A. Einstein (1911). "Investigations on the theory of the brownian movement". *Annalen der Physik* 34, pp. 591–592 (ver p. 217).

K. Faceli, A. C. Lorena, J. Gama e A. C. P. L. F. de Carvalho (2011). *Inteligência Artificial: uma Abordagem de Aprendizado de Máquina*. Rio de Janeiro, RJ: Livros Técnicos e Científicos (ver p. 192).

B. J. Falkowski e L. Schmitz (1986). "A Note on the queens' problem". *Information Processing Letters* 23.1, pp. 39–46. MR: 0853624 (ver p. 98).

P. J. Fernandez (1975). *Introdução aos Processos Estocásticos*. Poços de Caldas: IMPA (ver p. 217).

— (2005). *Introdução à Teoria das Probabilidades*. Rio de Janeiro: IMPA (ver p. 58).

C. Figueiredo, G. Fonseca, M. Lemos e V. Sa (2007). *Introdução aos Algoritmos Randomizados*. Publicações matemáticas. Rio de Janeiro: IMPA (ver pp. 58, 97).

P. Flajolet, É. Fusy, O. Gandouet e et al. (2007). "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm". Em: *Proceedings of the International Conference on Analyis of algorithms*. Zbl: 1192.68959 (ver p. 163).

P. Flajolet, D. Gardy e L. Thimonier (1992). "Birthday paradox, coupon collectors, caching algorithms and self-organizing search". *Discrete Applied Mathematics* 39.3, pp. 207–229. MR: 1189469. Zbl: 0762.60006 (ver p. 122).

P. Flajolet e G. N. Martin (1985). "Probabilistic counting algorithms for data base applications". *Journal of Computer and System Sciences* 31.2, pp. 182–209. MR: 0828521. Zbl: 0583.68059 (ver p. 163).

P. Flajolet e R. Sedgewick (2009). *Analytic Combinatorics*. University Press, Cambridge. MR: 2483235. Zbl: 1165.05001 (ver p. 121).

A. M. Frieze (1990). "Probabilistic Analysis of Graph Algorithms". Em: *Computational Graph Theory*. Ed. por G. Tinhofer, E. Mayr, H. Noltemeier e M. M. Syslo. Vienna: Springer. Zbl: 0729.68029 (ver p. 121).

A. M. Frieze e J. E. Yukich (2007). "Probabilistic Analysis of the TSP". Em: *The Traveling Salesman Problem and Its Variations*. Ed. por G. Gutin e A. P. Punnen. Boston: Springer, pp. 257–307. MR: 1944491. Zbl: 1113.90366 (ver p. 122).

A. Gakhov (2019). *Probabilistic Data Structures and Algorithms for Big Data Applications*. Books on Demand. MR: 3526721 (ver p. 163).

S. L. Gallant (1990). "Perceptron-based learning algorithms". *IEEE Transactions on Neural Networks* 1, pp. 179–191 (ver p. 192).

M. R. Garey e D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Bell Telephone Laboratories, Inc. MR: 0519066. Zbl: 0411.68039 (ver p. 25).

P. G. Gazmuri (1986). "Probabilistic Analysis of Algorithms for Some Combinatorial Optimization Problems". Em: *Recent Advances in System Modelling and Optimization. Lecture Notes in Control and Information Sciences*. Ed. por B. L. Contesse, F. R. Corrrea e P. A. Weintraub. Vienna: Springer, pp. 113–126. MR: 0901792. Zbl: 0615.90065 (ver p. 122).

I. P. Gent, C. Jefferson e P. Nightingale (2018). "Complexity of n-queens completion". Em: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, pp. 5608–5611 (ver p. 98).

Z. Ghahramani (2004). "Unsupervised Learning". Em: *Advanced Lectures in Machine Learning*, pp. 72–112. Zbl: 1120.68434 (ver p. 192).

P. B. Gibbons (2016). "Distinct-Values Estimation over Data Streams". Em: *Data Stream Management: Processing High-Speed Data Streams*. Ed. por M. Garofalakis, J. Gehrke e R. Rastogi. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 121–147 (ver p. 163).

J. Gill (1977). "Computational complexity of probabilistic Turing Machines". *SIAM Journal on Computing* 6.4, pp. 675–695. MR: 0464691 (ver p. 97).

G. H. Gonet e R. Baeza-Yates (1991). *Handbook of Algorithms and Data Structures in Pascal and C*. 2ª ed. Chatham: Addison-Wesley (ver p. 121).

R. L. Graham, D. E. Knuth e O. Patashnick (1994). *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Massachusetts. MR: 1397498 (ver p. 121).

C. A. R. Hoare (1962). "Quicksort". *The Computer Journal* 5.1, pp. 10–16. MR: 0142216. Zbl: 0108.13601 (ver p. 97).

E. J. Hoffman, J. C. Loessi e R. C. Moore (1969). "Constructions for the solution of the m queens problem". *National Mathematics Magazine*, pp. 66–72. MR: 0237364. Zbl: 0331.05017 (ver p. 98).

M. Hofri (1987). *Probabilistic Analysis of Algorithms*. New York: Springer-Verlag (ver p. 122).

L. Holst (1986). "On birthday, collectors', occupancy and other classical urn problems". *International Statistical Review / Revue Internationale de Statistique* 54.1, pp. 15–27. MR: 0959649. Zbl: 0594.60014 (ver p. 122).

D. Hong e J. Y.-T. Leung (1995). "Probabilistic analysis of k-dimensional packing algorithms". *Information Processing Letters* 55.1, pp. 17–24. MR: 1344778 (ver p. 122).

J. Hopcroft e R. Kannan (2012). "Computer Science Theory for the Information Age" (ver p. 192).

L. P. Kaelbling, M. L. Littman e A. W. Moore (1996). "Reinforcement learning". *Journal of Artificial Intelligence Research* 4, pp. 237–285 (ver p. 192).

L. V. Kalé (1990). "An almost perfect heuristic for the n nonattacking queens problem". *Information Processing Letters* 34.4, pp. 173–178. Zbl: 0696.68097 (ver p. 98).

S. D. Kamvar, T. H. Haveliwala, C. D. Manning e G. H. Golub (2003). "Extrapolation methods for accelerating PageRank computations". Em: *Proceedings of the Twelfth International World Wide Web Conference*. Ed. por G. Hencsey, B. White, Y. R. Chen, L. Kovács e S. Lawrence. ACM, pp. 261–270.

D. R. Karger (1993). "Global Min-cuts in RNC and other ramifications of a simple min-cut algorithm". Em: *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pp. 21–30. MR: 1213216. Zbl: 0801.68124 (ver p. 97).

D. R. Karger e C. Stein (1996). "A new approach for the minimum cut problem". *Journal of the ACM* 43, pp. 601–640. Zbl: 0882.68103 (ver p. 97).

S. Karlin e H. M. Taylor (2012). *A First Course in Stochastic Processes*. 2ª ed. Boston: Academic Press. MR: 0356197 (ver pp. 217, 238).

R. M. Karp (1972). "Reducibility among combinatorial problems". Em: *Proceedings of a symposium on the Complexity of Computer Computations*. Ed. por

R. E. Miller e J. W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, pp. 85–103. MR: 0378476.

R. M. Karp (1991). "An introduction to randomized algorithms". *Discrete Applied Mathematics* 34.1-3, pp. 165–201. MR: 1137994. Zbl: 0757 . 68085 (ver p. 97).

R. M. Karp, R. Motwani e N. Nisan (1993). "Probabilistic analysis of network flow algorithms". *Mathematics of Operations Research* 18.1, pp. 71–97. MR: 1250107. Zbl: 0780 . 90039 (ver p. 121).

R. M. Karp e J. M. Steele (1985). "Probabilistic Analysis of Heuristics". Em: *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Ed. por E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan e D. B. Shmoys. Chichester: John Willey & Sons, Inc., p. 476. Zbl: 0582 . 90100 (ver p. 122).

J. G. Kemeny, J. L. Snell e G. L. Thompson (1974). *Introduction to Finite Mathematics*. 3ª ed. Englewood Cliffs, N. J.: Prentice-Hall. MR: 0360048. Zbl: 0124 . 00101 (ver p. 217).

D. G. Kendall et al. (1990). *Andrei Nikolaevich Kolmogorov (1903–1987)*. MR: 1026769.

D. E. Knuth (1972). "Mathematical analysis of algorithms". Em: *Information Processing, Proceedings of IFIP Congress 1971 - Foundations and Systems, Ljubljana, Yugoslavia, August 23-28, 1971*. Ed. por C. V. Freiman, J. E. Griffith e J. L. Rosenfeld. Vol. 1. Republicado em Knuth 2004. North-Holland, pp. 18–24. MR: 0403310 (ver p. 121).

— (1997a). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 3ª ed. Boston: Addison-Wesley. MR: 3077154. Zbl: 0895 . 68055 (ver p. 121).

— (1997b). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3ª ed. Boston: Addison-Wesley. MR: 3077153 (ver pp. 98, 121).

— (1997c). *The Art of Computer Programming, Volume 3: Sorting and Searching*. 3ª ed. Boston: Addison-Wesley. MR: 3077154. Zbl: 0883 . 68015 (ver p. 121).

— (2004). "Selected Papers on Computer Science". Em: vol. 59. Republição de Knuth 2004. CLSI, p. 276. MR: 1415389 (ver p. 244).

A. N. Kolmogorov (1931). "On analytical methods in probability theory". *Mathematische Annalen* 104, pp. 415–458. MR: 1512678 (ver p. 217).

— (1933). *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Springer, Berlin. MR: 0362415. Zbl: 59 . 1152 . 03 (ver p. 58).

— (1956). "On the Shannon theory of information transmission in the case of continuous signals". *IRE Transactions on Information Theory* 2.4, pp. 102–108. MR: 0097987 (ver p. 217).

— (1998). "On tables of random numbers (Reprinted from "Sankhya: The Indian Journal of Statistics", Series A, Vol. 25 Part 4, 1963)". *Theoretical Computer Science* 207.2, pp. 387–395. MR: 1643414.

M. Kuhn e K. Johnson (2013). *Applied predictive modeling*. Vol. 26. Springer. MR: 3099297 (ver p. 217).

J. Langford (2005). "Tutorial on practical prediction theory for classification". *Journal of Machine Leaning Research* 6, pp. 273–306. MR: 2249822. Zbl: 1222.68243 (ver p. 192).

A. N. Langville e C. D. Meyer (2006). "Updating Markov Chains with an eye on Google's PageRank". *SIAM Journal on Matrix Analysis and Applications* 27.4, pp. 968–987. MR: 2205607 (ver p. 237).

M. N. Magalhães (2006). *Probabilidade e Variáveis Aleatórias*. 3ª ed. Edusp, São Paulo, p. 424 (ver p. 58).

A. A. Markov (1906a). "Extension of the law of large numbers to dependent events". *Bulletin of the Society Physics Mathematics, Kazan* 2.15, pp. 155–156 (ver pp. 216, 217).

— (1906b). "Extension of the law of large numbers to dependent events, Isvestia Soc". *Bulletin of the Society Physics Mathematics, Kazan* 15.4, pp. 135–156 (ver pp. 216, 217).

— (1913). "Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain". *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*. 6ª sér. 7, pp. 153–162 (ver p. 217).

— (2006). "An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains". *Science in Context* 19.4, pp. 591–600 (ver p. 217).

C. A. J. Martinhon (2002). *Algoritmos Randômicos em Otimização Combinatória*. Sociedade Brasileira de Pesquisa Operacional, Rio de Janeiro (ver p. 97).

N. Metropolis (1987). "The beginning of the Monte Carlo method". *j-LOS-ALAMOS-SCIENCE* 15, pp. 125–130. MR: 0935771 (ver p. 97).

N. Metropolis e S. Ulam (1949). "The Monte Carlo method". *Journal of the American Statistical Association* 44.247, pp. 335–341. MR: 0031341. Zbl: 0033.28807 (ver p. 97).

— (1976). "The Monte Carlo method". *Journal of the American Statistical Association* 44, pp. 335–341.

G. L. Miller (1976). "Riemann's hypothesis and tests for primality". *Journal of Computer and System Sciences* 13, pp. 300–317. MR: 0480295. Zbl: 0349.68025 (ver p. 97).

M. L. Minsky e S. Papert (1988). *Perceptrons: An Introduction to Computational Geometry*. MIT Press. Zbl: 0197.43702 (ver p. 192).

J. Misra e D. Gries (1982). "Finding repeated elements". *Science of Computer Programming* 2.2, pp. 143–152. MR: 0695463. Zbl: 0497.68041 (ver p. 163).

T. M. Mitchell (1997). *Machine Learning*. New York, NY: McGraw-Hill, p. 414 (ver p. 192).

M. Mitzenmacher e E. Upfal (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press. MR: 2144605. Zbl: 1092.60001 (ver pp. 58, 97, 238).

T. Mizoi e S. Osaki (1996). "Probabilistic analysis of the time complexity of Quicksort". *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* 79.3, pp. 34–42 (ver p. 121).

A. de Moivre (1711). *De mensura sortis, seu, de probabilitate eventuum in ludis a casu fortuito pendentibus*. Philosophical transactions of the Royal Society of London. H. Clements at the Half-Moon, e W. Innys at the Prince's Arms, in St. Paul's Churchyard (ver p. 122).

C. G. Moreira e Y. Kohayakawa (2010). *Tópicos em Combinatória Contemporânea*. Rio de Janeiro: IMPA (ver p. 58).

R. Morris (1978). "Counting large numbers of events in small registers". 21.10, pp. 840–842. Zbl: 0386.68035 (ver p. 163).

C. H. Papadimitriou e K. S. Steiglitz (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall International, Inc. MR: 0663728. Zbl: 0503.90060 (ver p. 25).

K. Pearson (1905). "The problem of the random walk". *Nature* 72.1865, pp. 294–294. Zbl: 36.0303.02 (ver p. 237).

H. Poincaré (1912). *Calcul des Probabilités*. Gauthier-Villars, Paris. MR: 1190693 (ver p. 216).

S. D. Poisson (1837). *Recherches sur la Probabilité des Jugements en Matière Criminelle et en Matière Civile: Précédées des Règles Générales du Calcul des Probabilités*. Bachelier (ver p. 58).

R. Prudêncio e T. B. Ludermir (2004). "A meta-learning approach to selecting time series models". *Neurocomputing* 61, pp. 121–137.

M. O. Rabin (1976). "Probabilistic Algorithms". Em: *Algorithms and Complexity: New Directions and Recent Results*. Ed. por J. F. Traub. New York: Academic Press, pp. 21–39. MR: 0464678 (ver p. 97).

— (1980). "Probabilistic algorithm for testing primality". *Journal of Number Theory* 12, pp. 128–138. MR: 0566880. Zbl: 0426.10006 (ver p. 97).

M. Rajeev e R. Prabhakar (1995). *Randomized Algorithms*. Cambridge University Press. Zbl: 0849.68039 (ver p. 97).

M. Reichling (1987). "A simplified solution of the n queens problem". *Information Processing Letters* 25.4, pp. 253–255. MR: 0896145. Zbl: 0637.05003 (ver p. 98).

F. Rosenblatt (1958). "The Perceptron: a probabilistic model for information storage and organization in the brain". *Psychological Review* 65, pp. 386–408 (ver p. 192).

S. M. Ross (2006). *Introduction to Probability Models*. 9ª ed. USA: Academic Press, Inc. MR: 0750654. Zbl: 1118.60002 (ver p. 217).

— (2010). *A First Course in Probability*. Pearson Prentice Hall, Upper Saddle River. Zbl: 1307.60002 (ver pp. 56–58).

— (2014). "Introduction to Probability Theory". Em: *Introduction to Probability Models*. Ed. por S. M. Ross. 11ª ed. Boston: Academic Press, pp. 1–19. MR: 0750654 (ver p. 217).

K. S. Sarma (2017). *Predictive modeling with SAS enterprise miner: Practical solutions for business applications*. SAS Institute (ver p. 217).

R. Sedgewick (1977). "The analysis of Quicksort programs". *Acta Informatica* 7.4, pp. 327–355. MR: 0451845 (ver p. 121).

C. E. Shannon e W. Weaver (1949). "The mathematical theory of communication". *Urbana: University of Illinois Press*. MR: 0032134. Zbl: 0041.25804 (ver p. 217).

L. Slominski (1982). "Probabilistic analysis of combinatorial algorithms: a bibliography with selected annotations". *Computing* 28.3, pp. 257–267. MR: 0658188. Zbl: 0479.68040 (ver p. 122).

R. Solovay e V. Strassen (1977). "A fast Monte Carlo test for primality". *SIAM Journal on Computing* 6.1, pp. 84–85. MR: 0429721. Zbl: 0345.10002 (ver p. 97).

R. Sosic e J. Gu (1990). "A polynomial time algorithm for the n-queens problem". *SIGART Bulletin* 1.3, pp. 7–11 (ver p. 98).

— (1991). "Fast search algorithms for the n-queens problem". *IEEE Transactions on Systems, Man, and Cybernetics* 21.6, pp. 1572–1576. Zbl: 0792.68168 (ver p. 98).

— (1994). "Efficient local search with conflict minimization: a case study of the n-queens problem". *IEEE Transactions on Knowledge and Data Engeneering* 6.5, pp. 661–668 (ver p. 97).

D. A. Spielman e S. H. Teng (2001). "Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time". Em: *Proceedings of the*

33rd Annual ACM Symposium on Theory of Computing. ACM, pp. 296–305. MR: 2120328. Zbl: 1323.68636 (ver p. 122).

J. L. Szwarcfiter (2018). *Teoria Computacional de Grafos: Os Algoritmos*. Elsevier Editoria Ltda. (ver p. 25).

J. L. Szwarcfiter e L. Markenzon (2010). *Estruturas de Dados e seus Algoritmos*. 3ª ed. Rio de Janeiro: LTC (ver p. 121).

G. Udny Y (1939). "Karl Pearson". *Nature* 143.3615, pp. 220–222 (ver p. 237).

L. G. Valiant (1984). "A theory of learnable". *Communications of the ACM* 27, pp. 422–426. Zbl: 0587.68077 (ver p. 191).

V. N. Vapnik e A. Y. Chervonenkis (1971). "On the uniform convergence of relative frequencies of events to theirs probabilities". *Theory of Probability and its Applications* 16, pp. 264–280. MR: 0288823. Zbl: 0247.60005 (ver p. 192).

V. N. Vapnik, E. Levin e Y. L. Cun (1994). "Measuring the VC dimension of a learning machine". *Neural Computation* 6, pp. 851–876 (ver p. 192).

A. D. Ventsel (1992). *Selected Works of A. N. Kolmogorov: Volume I: Mathematics and Mechanics, No. 09*. Vol. 26. Springer Science & Business Media, pp. 522–527 (ver p. 217).

K. Whang, B. T. V. Zanden e H. M. Taylor (1990). "A linear-time probabilistic counting algorithm for database applications". *ACM Transactions Database Systems* 15.2, pp. 208–229 (ver p. 163).

M. J. Zaki e W. Meira (2020). *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. 2ª ed. Cambridge University Press. Zbl: 1432.68001 (ver p. 192).

N. Ziviani (1999). *Projeto de Algoritmos com Implementação em C e em Pascal*. 4ª ed. Pioneira Informática, São Paulo (ver p. 25).

Índice de Notações

Índice de Autores

Índice Remissivo

Índice de Algoritmos

Títulos Publicados — 33º Colóquio Brasileiro de Matemática

Geometria Lipschitz das singularidades – *Lev Birbrair e Edvalter Sena*

Combinatória – *Fábio Botler, Maurício Collares, Taísa Martins, Walner Mendonça, Rob Morris e Guilherme Mota*

Códigos geométricos, uma introdução via corpos de funções algébricas – *Gilberto Brito de Almeida Filho e Saeed Tafazolian*

Topologia e geometria de 3-variedades, uma agradável introdução – *André Salles de Carvalho e Rafał Marian Siejakowski*

Ciência de dados: algoritmos e aplicações – *Luerbio Faria, Fabiano de Souza Oliveira, Paulo Eustáquio Duarte Pinto e Jayme Luiz Szwarcfiter*

Discovering Poncelet invariants in the plane – *Ronaldo A. Garcia e Dan S. Reznik*

Introdução à geometria e topologia dos sistemas dinâmicos em superfícies e além – *Víctor León e Bruno Scárdua*

Equações diferenciais e modelos epidemiológicos – *Marlon M. López-Flores, Dan Marchesin, Vítor Matos e Stephen Schecter*

Differential Equation Models in Epidemiology – *Marlon M. López-Flores, Dan Marchesin, Vítor Matos e Stephen Schecter*

A friendly invitation to Fourier analysis on polytopes – *Sinai Robins*

PI-álgebras: uma introdução à PI-teoria – *Rafael Bezerra dos Santos e Ana Cristina Vieira*

First steps into Model Order Reduction – *Alessandro Alla*

The Einstein Constraint Equations – *Rodrigo Avalos e Jorge H. Lira*

Dynamics of Circle Mappings – *Edson de Faria e Pablo Guarino*

Statistical model selection for stochastic systems with applications to Bioinformatics, Linguistics and Neurobiology – *Antonio Galves, Florencia Leonardi e Guilherme Ost*

Transfer operators in Hyperbolic Dynamics - an introduction – *Mark F. Demers, Niloofar Kiamari e Carlangelo Liverani*

A course in Hodge Theory: Periods of Algebraic Cycles – *Hossein Movasati e Roberto Villaflor Loyola*

A dynamical system approach for Lane-Emden type problems – *Liliane Maia, Gabrielle Nornberg e Filomena Pacella*

Visualizing Thurston's geometries – *Tiago Novello, Vinícius da Silva e Luiz Velho*

Scaling problems, algorithms and applications to Computer Science and Statistics – *Rafael Oliveira e Akshay Ramachandran*

An introduction to Characteristic Classes – *Jean-Paul Brasselet*

impa

Instituto de
Matemática
Pura e Aplicada

ISBN 978-65-89124-47-4