Matemática Discreta para Cursos de Computação

Jackson Gomes \ jgomes@ceulp.edu.br

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Códigos-fontes

Prefácio

Este é um texto de apoio à disciplina Matemática Discreta para os cursos de computação do Centro Universitário Luterano de Palmas. Sempre que possível serão apresentadas referências a conceitos da computação e como eles se relacionam com os conceitos matemáticos apresentados. A principal referência do conteúdo utilizado aqui é (MENEZES, 2010).

Convenções

Os trechos de código apresentados no livro seguem o seguinte padrão:

- **comandos**: devem ser executados no prompt; começam com o símbolo `$`
- **códigos-fontes**: trechos de códigos-fontes de arquivos

A seguir, um exemplo de comando:

```
$ mkdir hello-world
```

O exemplo indica que o comando `mkdir`, com a opção `hello-world`, deve ser executado no prompt para criar uma pasta com o nome `hello-world`.

A seguir, um exemplo de código-fonte:

```
1 class Pessoa:
2     pass
```

O exemplo apresenta o código-fonte da classe `Pessoa`. Em algumas situações, trechos de código podem ser omitidos ou serem apresentados de forma incompleta, usando os símbolos `...` e `#`, como no exemplo a seguir:

```
1 class Pessoa:
2     def __init__(self, nome):
3         self.nome = nome
4 
5     def salvar(self):
6         # executa validação dos dados
```

```
7           ...
8           # salva
9           return ModelManager.save(self)
```

Conhecimentos necessários e desejáveis

Este texto aborda conceitos matemáticos com aplicações em computação. Portanto, conhecimentos básicos dos cursos de computação são necessários, como noções de lógica, algoritmos e programação e estruturas de dados. Além disso, são desejáveis conhecimentos de bancos de dados e orientação a objetos e também podem ser recursos úteis:

- Git (GIT COMMUNITY, [s.d.])
- Visual Studio Code (MICROSOFT, [s.d.])
- TypeScript (MICROSOFT, [s.d.])
- Node.js (NODE.JS FOUNDATION, [s.d.])
- npm (NPM, INC., [s.d.])

Capítulo 1

Introdução

1.1 Matemática Discreta

Conforme (MENEZES, 2010) as Diretrizes Curriculares do MEC para os cursos de computação e informática definem que:

> A matemática, para a área de computação, deve ser vista como uma ferramenta a ser usada na definição formal de conceitos computacionais (linguagens, autômatos, métodos etc.). Os modelos formais permitem definir suas propriedades e dimensionar suas instâncias, dadas suas condições de contorno.

Além disso, afirmam:

> Considerando que a maioria dos conceitos computacionais pertencem ao domínio discreto, a **matemática discreta** (ou também chamada álgebra abstrata) é fortemente empregada.

Desta forma, a **Matemática Discreta** preocupa-se com o emprego de técnicas e abordagens da matemática para o entendimento de problemas a serem resolvidos com computação. Mas o que significa ser **discreto**? A matemática, por si, trata também do domínio **contínuo**. Assim, estes domínios são opostos: contínuo e discreto. Para entender isso melhor, observe a Figura 1.1:

Figura 1.1 representa a função $y = x^2$, com $0 \leq x \leq 5$ em dois gráficos, sendo que o da direita destaca pontos selecionados, que representam 6 amostras (0, 1, 2, 3, 4 e 5).

Aumentando-se o número de amostras em dois instantes, para 10, 100 e 1000 teríamos, como mostra a Figura 1.2:

O que se pode perceber pela Figura 1.2 é que quanto mais se aumenta o número de amostras, mais se aproxima de uma curva perfeita. Entretanto, há um certo limite de percepção da perfeição dessa curva, por assim dizer. Por exemplo, embora a quantidade de amostras do gráfico da esquerda seja menor, a diferença para o gráfico da direita, visualmente falando, é pouco perceptível.

Considere outro exemplo: um computador possui uma capacidade de armazenamento virtualmente infinita. “Virtualmente” porque embora se aceite um limite, ele não é conhecido, já que a quan-

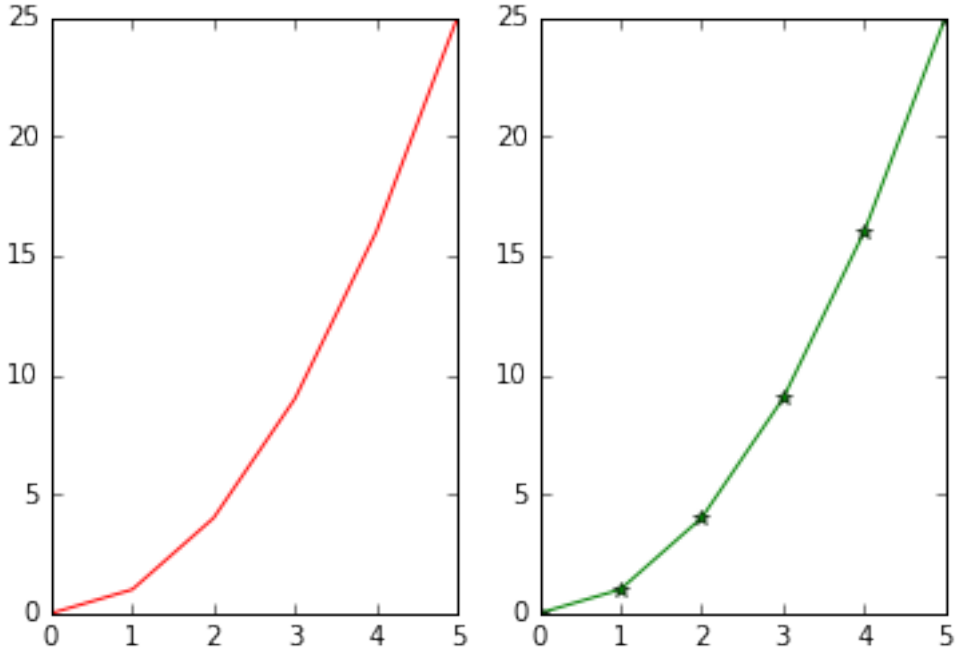

Figura 1.1: Gráfico da $y = x^2$, com $0 \leq x \leq 5$

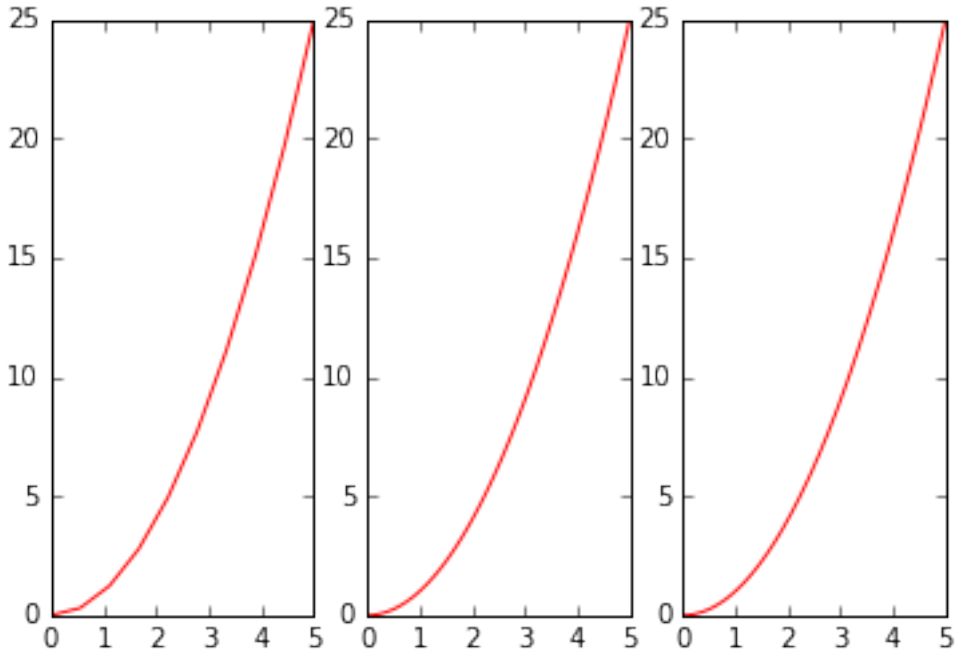

Figura 1.2: Gráfico da $y = x^2$ com mais amostras

tidade de unidades de armazemamento pode ser bastante grande, mas é **contável**. Assim, no contexto da computação, embora algo possa ser considerado finito ou infinito, ele é *contável* ou *discreto* no sentido de que pode ser enumerado ou sequenciado, de forma que não existe um elemento entre quaisquer dois elementos consecutivos da enumeração.

No exemplo do computador, embora a quantidade de unidades de armazenamento não seja conhecida, ela é contável e enumerável e não se pode afirmar que exista, por um exemplo, um disco rídigo desconhecido entre os array de discos composto por D1 e D2. Outro exemplo: na matemática, o conjunto dos números naturais é contável (ou enumerável), equanto o conjunto dos números reais não é contável.

Assim, a matemática discreta possui como ênfase os estudos matemáticos baseados em conjuntos contáveis, sejam eles finitos ou infinitos. De forma oposta, a *matemática do continuum* possui ênfase nos conjuntos não contáveis. Um exemplo disso são o cálculo diferencial e integral.

1.2 Teoria dos Conjuntos

Os **conjuntos** são a base da forma de representação de enumerações de elementos em matemática discreta. Por definição um conjunto é:

> uma estrutura que agrupa objetos e constitui uma base para construir estruturas mais complexas.

Segue uma definição mais formal:

> Um *conjunto* é uma coleção de zero ou mais objetos distintos, chamados *elementos* do conjunto, os quais não possuem qualquer ordem associada.

O fato de não haver uma *ordem associada* não significa que os elementos não possam estar ordenados, num dado contexto, conforme algum critério. Apenas indica que, no geral, isso não é obrigatório.

Há duas formas (notações) de representar conjuntos: notação por extensão e notação por compreensão.

Notação por extensão é quando todos os elementos do conjunto estão enumerados, representados entre chaves e separados por vírgula. Exemplo:

Vogais $= \{a, e, i, o, u\}$.

Entende-se que se um conjunto pode ser representado por extensão, então ele é *finito*. Caso contrário, é *infinito*.

Notação por compreensão representa conjuntos usando propriedades. Os exemplos a seguir usam uma pequena diferença de notação, mas representam a mesma coisa:

- Pares $= \{n \mid n$ é um número par$\}$
- Pares $= \{n : n$ é um número par$\}$

Este conjunto é interpretado como: o conjunto de todos os elementos n tal que n é um número par. A forma geral de representar um conjunto por propriedades é:

$X = \{x : p(x)\}$

Isso quer dizer que x é um elemento de X se a propriedade $p(x)$ for verdadeira.

A notação por propriedades é uma boa forma de representar conjuntos *infinitos*.

Há ainda uma outra forma aceitável de representar conjuntos usando uma representação semelhante à de por extensão. Exemplos:

- Digitos $= \{0, 1, 2, ..., 9\}$
- Pares $= \{0, 2, 4, 6, ...\}$

Embora haja elementos ausentes, substituídos por reticências (...) é completamente aceitável e entendível o que se quer informar com a descrição do conjunto.

O número de elementos de um conjunto A é representado por $|A|$ (isso também é chamado "cardinalidade"). Portanto, se $A = \{1, 2, 3\}$, $|A| = 3$ e $|\emptyset| = 0$.

A seguir, revemos conceitos de algumas relações entre e com conjuntos ou elementos.

1.2.1 Pertinência

Se um elemento a pertence ao conjunto A isso é representado como: $a \in A$. Caso contrário, se a não pertence a A, então representa-se como: $a \notin A$.

Exemplos: Pertence, não pertence

Quanto ao conjunto Vogais $= \{a, e, i, o, u\}$:
a) $a \in$ Vogais
b) $h \notin$ Vogais

Quanto ao conjunto $B = \{x : x$ é brasileiro$\}$:
a) Pele $\in B$
b) Bill Gates $\notin B$

1.2.2 Conjuntos importantes

O **conjunto vazio** é um conjunto sem elementos, representado como $\{\}$ ou $\emptyset$. Exemplos:

- o conjunto de todos os brasileiros com mais de 300 anos;
- o conjunto dos números que são, simultaneamente, ímpares e pares.

O **conjunto unitário** é um conjunto constituído por um único elemento. Exemplos:

- o conjunto constituído pelo jogador de futebol Pelé;
- o conjunto de todos os números que são, simultaneamente, pares e primos, ou seja: $P = \{2\}$;
- um conjunto unitário cujo elemento é irrelevante: $1 = \{*\}$.

O **conjunto universo**, normalmente denotado por U, contém todos os conjuntos considerados em um dado contexto.

Outros conjuntos importantes:

- N: o conjunto dos números naturais (inteiros positivos e o zero)
- Z: o conjunto dos números inteiros (inteiros negativos, positivos e o zero)
- Q: o conjunto dos números racionais (os que podem ser representados na forma de fração)
- I: o conjunto dos números irracionais
- R: o conjunto dos números reais

1.2.3 Alfabetos, palavras e linguagens

Em computação, e mais especificamente em linguagens de programação, um conceito importante é o que define o conjunto de elementos ou termos-chave da linguagem.

Um **alfabeto** um conjunto finito cujos elementos são denominados *símbolos* ou *caracteres*.

Uma **palavra** (cadeia de caracteres ou sentença) sobre um alfabeto é uma sequência finita de símbolos justapostos.

Uma **linguagem** [**formal**] é um conjunto de palavras sobre um alfabeto.

Exemplos: alfabeto, palavra

---

a) Os conjuntos $\emptyset$ e $\{a, b, c\}$ são alfabetos
b) O conjunto N não é um alfabeto
c) ϵ é uma palavra vazia
d) Σ é geralmente usada para representar um alfabeto
e) Σ^* é o conjunto de todas as palavras possíveis sobre o alfabeto Σ
f) ϵ é uma palavra do alfabeto $\emptyset$
g) $\{a, b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, ...\}$

1.2.4 Continência, subconjunto e igualdade de conjuntos

A *continência* permite introduzir os conceitos de *subconjunto* e *igualdade de conjunto*.

Se todos os elementos de um conjunto A também são elementos de um conjunto B, então A está *contido* em B, o que é representado por: $A \subseteq B$. Isso também é lido como A é *subconjunto* de B.

Se $A \subseteq B$, mas há $b \in B$ tal que $b \notin A$, então pode-se dizer que A está *contido propriamente* em B, ou que A é *subconjunto próprio* de B. Isso é denotado por: $A \subset B$.

A negação de *subconjunto* e *subconjunto próprio* é, respectivamente:

- $A \not\subseteq B$ e
- $A \not\subset B$

Exemplos: continência, subconjunto

a) $\{a, b\} \subseteq \{b, a\}$
b) $\{a, b\} \subset \{a, b, c\}$, e $\{a, b\} \subseteq \{a, b, c\}$

A *igualdade de conjuntos* é um conceito baseado em pertinência: se os elementos de A também são elementos de B e vice-versa, então $A = B$. Formalmente, uma condição para $A = B$ é que $A \subseteq B$ e $B \subseteq A$.

Exemplo

$\{1, 2, 3\} = \{3, 3, 3, 2, 2, 1\}$

É importante notar que pertinência ($\in$) é usada entre elementos e conjuntos, enquanto continência ($\subset$ e $\subseteq$) é usada entre conjuntos.

Por definição, um conjunto qualquer é subconjunto de si mesmo, e $\emptyset$ é subconjunto de qualquer conjunto.

Exemplo

Seja $A = \{1, 2\}$ então os subconjuntos de A são: $\emptyset$, $\{1\}$, $\{2\}$ e $\{1, 2\}$.

1.3 Conjuntos, Tuplas e Listas

Uma **Tupla** (ou ênupla) é uma sequência ordenada de n elementos (ou componentes). As principais diferenças para **conjunto** são:

- uma ênupla pode conter um elemento mais de uma vez; e
- os elementos são, obrigatoriamente, ordenados, ou seja, cada elemento está em uma posição diferente.

A notação utilizada é $X = (c_1, c_2, ..., c_n)$ onde:

- X é uma ênupla com n componentes
- c_1 é o primeiro componente da ênupla
- c_n é o último componente da ênupla
- c_i é o i-ésimo componente da ênupla

Exemplos:

- $X = (1, 1, 2, 3)$
- $X = (1_1, 1_2, 2_3, 3_4)$
- $Y = (3_1, 2_2, 1_3)$

As mesmas relações de pertinência entre conjuntos e elementos podem ser aplicadas entre ênuplas e seus componentes.

Uma **Lista** é uma estrutura de dados que implementa o conceito matemático de **Tupla**, então podemos afirmar para Vogais $= (a, e, i, o, u)$:

- Vogais é uma lista com 5 elementos
- a é a primeira vogal
- u é a última vogal

1.4 Exercícios

Questão 1. Para cada conjunto abaixo: a) descreva de forma alternativa (usando outra forma de notação); e b) diga se é finito ou infinito.

a) todos os números inteiros maiores que 10
b) $\{1, 3, 5, 7, 9, 11, ...\}$
c) todos os países do mundo (Terra)
d) a linguagem de programação Python

Questão 2. Para $A = \{1\}$, $B = \{1, 2\}$ e $C = \{\{1\}, 1\}$, marque as afirmações corretas:

a) $A \subset B$
b) $A \subseteq B$
c) $A \in B$
d) $A = B$
e) $A \subset C$
f) $A \subseteq C$
g) $A \in C$
h) $A = C$
i) $1 \in A$
j) $1 \in C$
k) $\{1\} \in A$
l) $\{1\} \in C$
m) $\emptyset \notin C$
n) $\emptyset \subset C$

Questão 3. Sejam $a = \{x \mid 2x = 6\}$ e $b = 3$. É correto afirmar que $a = b$? Por que?

Questão 4. Quais todos os subconjuntos dos seguintes conjuntos?

a) $A = \{a, b, c\}$
b) $B = \{a, \{b, c\}, D\}$, dado que $D = \{1, 2\}$

Questão 5. O conjunto vazio está contido em qualquer conjunto, inclusive nele próprio? Justifique.

Questão 6. Todo conjunto possui um subconjunto próprio? Justifique.

Questão 7. Sejam $A = \{0, 1, 2, 3, 4, 5\}$, $B = \{3, 4, 5, 6, 7, 8\}$, $C = \{1, 3, 7, 8\}$, $D = \{3, 4\}$, $E = \{1, 3\}$, $F = \{1\}$ e X um conjunto desconhecido. Para cada item abaixo, determine quais dos conjuntos A, B, C, D, E ou F podem ser iguais a X:

a) $X \subseteq A$ e $X \subseteq B$
b) $X \not\subset B$ e $X \subseteq C$
c) $X \not\subset A$ e $X \not\subset C$
d) $X \subseteq B$ e $X \not\subset C$

Questão 8. Sejam A um subconjunto de B e B um subconjunto de C. Suponha que $a \in A$, $b \in B$, $c \in C$, $d \notin A$, $e \notin B$, $f \notin C$. Quais das seguintes afirmações são verdadeiras?

a) $a \in C$
b) $b \in A$
c) $c \notin A$
d) $d \in B$
e) $e \notin A$
f) $f \notin A$

Questão 9. Bancos de dados relacionais costumam representar conjuntos de dados organizados em tabelas, colunas e registros. É possível considerar alguma semelhança entre o conceito de tabela e o conceito de conjunto? Há recursos de consulta (em linguagem SQL, por exemplo) que permitam identificar, ainda que parcialmente, relações entre tabelas e registros (ou valores) como as expressadas nesse capítulo entre elementos e conjuntos? Explique.

Questão 10. Escolha uma linguagem de programação e demonstre seus recursos para lidar com conjuntos e listas. Utilizando código-fonte (e a sintaxe da linguagem de programação) demonstre e explique as diferenças conceituais e demonstre recursos da linguagem que permitam identificar as relações de pertinência, continência, subconjunto e igualdade.

Capítulo 2

Noções de lógica e técnicas de demonstração

Depois de ver sobre a **Teoria dos conjuntos** fica mais evidente a necessidade de estabelecer uma linguagem lógico-matemática para demonstrações e provas. Este capítulo apresenta uma revisão dos conceitos de lógica e traz técnicas de demonstração úteis nos capítulos seguintes sobre provas.

2.1 Tautologia e contradição

Seja w uma fórmula. Então:

a) w é chamada de **tautologia** se w for **verdadeira** (considerando todas as combinações possíveis de valores de sentenças variáveis – entradas). Por exemplo: a fórmula $p \vee \neg p$ é uma tautologia;
b) w é chamada **contradição** se w for **falsa**. Por exemplo: a fórmula $p \wedge \neg p$ é uma contradição.

A tabela-verdade da Tabela 2.1 resume os valores de entrada possíveis para p e $\neg p$ demonstrando tautologia e contradição das fórmulas de exemplo.

Tabela 2.1: Tabela-verdade demonstrando tautologia e contradição

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
V	F	V	F
F	V	V	F

2.2 Implicação e equivalência

A **relação de implicação** é definida como: sejam p e q duas fórmulas. Então p *implica* q se e somente se $p \rightarrow q$ é uma tautologia. Isso é denotado por:

$$p \Rightarrow q \tag{2.1}$$

Para exemplificar, considere a tabela-verdade a seguir.

p	q	$p \vee q$	$p \rightarrow (p \vee q)$	$p \wedge q$	$(p \wedge q) \rightarrow p$
V	V	V	V	V	V
V	F	V	V	F	V
F	V	V	V	F	V
F	F	F	V	F	V

Vale a implicação chamada **adição**, definida por $p \Rightarrow p \vee q$. Para validar essa afirmação, verificamos que $p \rightarrow (p \vee q)$ é uma tautologia aplicando o condicional (quarta coluna da tabela-verdade).

Vale a implicação chamada **simplificação**, definida por $p \wedge q \Rightarrow q$. Para validar essa afirmação, verificamos que $(p \wedge q) \rightarrow p$ é uma tautologia (sexta coluna da tabela-verdade).

A **relação de equivalência** é definida como: sejam p e q duas fórmulas. Então p *é equivalente a* q se e somente se $p \leftrightarrow q$ é uma tautologia. Isso é denotado por:

$$p \Leftrightarrow q \tag{2.2}$$

Considere a relação de equivalência da fórmula:

$$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$$

Para verificar a validade da equivalência, basta construir a tabela-verdade para demonstrar que $p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee q)$ (chamada de S) é uma tautologia, ou seja:

p	q	r	$q \wedge r$	$p \vee (q \wedge r)$	$p \vee q$	$p \vee r$	$(p \vee q) \wedge (p \vee r)$	S
V	V	V	V	V	V	V	V	V
V	V	F	F	V	V	V	V	V
V	F	V	F	V	V	V	V	V
V	F	F	F	V	V	V	V	V
F	V	V	V	V	V	V	V	V
F	V	F	F	F	V	F	F	V
F	F	V	F	F	F	V	F	V
F	F	F	F	F	F	F	F	V

Também é interessante observar que a fórmula ilustra que a *distributividade* do conectivo **ou** sobre o conectivo **e** é verdadeira sempre, ou seja, é uma tautologia. O mesmo valeria para a distributividade do conectivo **e** sobre o conectivo **ou**? Verifique.

Para verificar a equivalência da fórmula:

$$(p \leftrightarrow q) \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$$

basta demonstrar que $(p \leftrightarrow q) \leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$ (chamada de S) é uma tautologia. Isso é conseguido utilizando a tabela-verdade:

Tabela 2.4: Tabela-verdade: bicondição x condição

p	q	$p \leftrightarrow q$	$p \rightarrow q$	$q \rightarrow p$	$(p \rightarrow q) \wedge (q \rightarrow p)$	S
V	V	V	V	V	V	V
V	F	F	F	V	F	V
F	V	F	V	F	F	V
F	F	V	V	V	V	V

Essa forma em particular demonstra formalmente por que a bicondição pode ser expressa por duas condições: “ida” e “volta”.

Para verificar a equivalência da fórmula:

$$p \rightarrow q \Leftrightarrow \neg q \rightarrow \neg p$$

basta demonstrar que $p \rightarrow q \leftrightarrow \neg q \rightarrow \neg p$ (chamada de S) é uma tautologia. Isso é conseguido utilizando a tabela-verdade:

Tabela 2.5: Tabela-verdade: contraposição

p	q	$\neg p$	$\neg q$	$p \rightarrow q$	$\neg q \rightarrow \neg p$	S
V	V	F	F	V	V	V
V	F	F	V	F	F	V
F	V	V	F	V	V	V
F	F	V	V	V	V	V

Essa relação de equivalência é conhecida como **contraposição**.

Para verificar a equivalência da fórmula:

$$p \rightarrow q \Leftrightarrow p \wedge \neg q \rightarrow F$$

basta demonstrar que $p \rightarrow q \leftrightarrow p \wedge \neg q \rightarrow F$ (chamada de S) é uma tautologia. Isso é conseguido utilizando a tabela-verdade:

Tabela 2.6: Tabela-verdade: redução ao absurdo

p	q	$\neg q$	$p \rightarrow q$	$p \wedge \neg q$	$p \wedge \neg q \rightarrow F$	S
V	V	F	V	F	V	V

p	q	$\neg q$	$p \to q$	$p \wedge \neg q$	$p \wedge \neg q \to F$	S
V	F	V	F	V	F	V
F	V	F	V	F	V	V
F	F	V	V	F	V	V

Essa relação de equivalência é conhecida como **redução ao absurdo**.

2.3 Quantificadores

Seja A um conjunto. Uma proposição $p(x)$ sobre A:

a) descreve alguma propriedade de um elemento $x \in A$; e
b) tem valor lógico dependendo do elemento $x \in A$.

Então:

a) o **conjunto verdade** $p(x) = \{x \in A \mid p(x) \text{ é verdadeira}\}$
b) o **conjunto falsidade** $p(x) = \{x \in A \mid p(x) \text{ é falsa}\}$

Ainda:

a) se $p(x)$ é verdadeira para qualquer $x \in A$ então é uma tautologia, ou seja, o conjunto verdade é A;
b) se $p(x)$ é falsa para qualquer $x \in A$ então é uma contradição, ou seja, o conjunto falsidade é A.

Exemplos: conjuntos verdade e falsidade, tautologia e contradição

Suponha o conjunto universo N, então:
a) para $p(n) = n > 1$:
- conjunto verdade: $\{2, 3, 4, ...\}$
- conjunto falsidade: $\{0, 1\}$
- náo é tautologia e nem contradição. Por exemplo, é verdadeira para $n = 2$, mas falsa para $n = 0$

b) para $p(n) = n! < 10$:
- conjunto verdade: $\{0, 1, 2, 3\}$
- conjunto falsidade: $\{n \in N \mid n > 3\}$
- não é tautologia e nem contradição. Por exemplo, é verdadeira para $n = 0$, mas falsa para $n = 4$

c) para $p(n) = n + 1 > n$:
- conjunto verdade: N
- conjunto falsidade: $\emptyset$
- é uma tautologia

Exemplos: conjuntos verdade e falsidade, tautologia e contradição

d) para $p(n) = 2n$ é ímpar
- conjunto verdade: $\emptyset$
- conjunto falsidade: N
- é uma contradição

Os **quantificadores** são utilizados na lógica para, com relação a uma determinada proposição $p(x)$, quantificar os valores de x que devem ser considerados.

O **quantificador universal**, simbolizado por $\forall$, quando associado a uma proposição $p(x)$, é denotado como qualquer uma das três opções a seguir:

$$(\forall x \in A)(p(x)) \qquad (\forall x \in A)p(x) \qquad \forall x \in A, p(x) \tag{2.3}$$

ou, quando é claro sobre qual conjunto de valores a proposição está definida, pode-se usar:

$$(\forall x)(p(x)) \qquad (\forall x)p(x) \qquad \forall x, p(x) \tag{2.4}$$

A leitura da fórmula $(\forall x \in A)p(x)$ é: "qualquer x, $p(x)$" ou "para todo x, $p(x)$"

O **quantificador existencial**, simbolizado por $\exists$, quando associado a uma proposição $p(x)$, é denotado como qualquer uma das opções a seguir:

$$(\exists x \in A)(p(x)) \qquad (\exists x \in A)p(x) \qquad \exists x \in A, p(x) \tag{2.5}$$

ou, quando é claro sobre qual conjunto de valores a proposição está definida, pode-se usar:

$$(\exists x)(p(x)) \qquad (\exists x)p(x) \qquad \exists x, p(x) \tag{2.6}$$

A leitura da fórmula $(\exists x \in A)p(x)$ é: "existe pelo menos um x tal que $p(x)$" ou "existe x tal que $p(x)$".

O valor-verdade de cada proposição quantificada é:

a) $(\forall x \in A)p(x)$ é **verdadeira**, se $p(x)$ for **verdadeira** para **todos os elementos de** A; e
b) $(\exists x \in A)p(x)$ é **verdadeira**, se $p(x)$ for **verdadeira** para **pelo menos um elemento de** A.

Portanto:

a) **quantificador universal**: a proposição $(\forall x \in A)p(x)$ é:
   - **verdadeira** se o conjunto verdade for igual ao conjunto A
   - **falsa**, caso contrário
b) **quantificador existencial**: a proposição $(\exists x \in A)p(x)$ é:
   - **verdadeira**, se o conjunto verdade for não vazio (ou diferente de $\emptyset$)

- **falsa**, caso contrário (ou igual a $\emptyset$)

A noção de uma proposição $p(x)$ sobre um conjunto pode ser generalizada para descrever alguma propriedade de elementos $x_1 \in A_1, x_2 \in A_2, ..., x_n \in A_n$, sendo denotada da seguinte forma:

$$p(x_1, x_2, ..., x_n) \tag{2.7}$$

Por exemplo, para o conjunto universo N valem:

- $(\forall n)(\exists m)(n < m)$ é **verdadeira**: dado **qualquer** valor n, **existe pelo menos um** m que satisfaz a desigualdade. Por exemplo: tomando $m = n + 1$ é verdadeiro que $n < n + 1$
- $(\exists m)(\forall n)(n < m)$ é **falsa**: **existe** um número natural maior que **qualquer** outro, o que não é verdade.

A **negação da proposição quantificada**:

$$(\forall x \in A)p(x)$$

é definida como:

$$(\exists x \in A)\neg p(x) \tag{2.8}$$

o que significa que existe pelo menos um x tal que não é fato que ou não é verdade que $p(x)$.

Logo:

$$\neg((\forall x \in A)p(x)) \Leftrightarrow (\exists x \in A)\neg p(x)$$

e, também:

$$\neg((\exists x \in A)p(x)) \Leftrightarrow (\forall x \in A)\neg p(x)$$

2.4 Técnicas de demonstração

Um **teorema** é uma proposição que prova-se uma tautologia, do tipo:

$$p \rightarrow q \tag{2.9}$$

ou seja, uma implicação:

$$p \Rightarrow q$$

onde:

- p é chamada de **hipótese** (ou **premissa**) e é, por suposição, **verdadeira**
- q é chamada de **tese** (ou **conclusão**)

Para exemplificar considere o teorema:

> 0 é o único elemento neutro da adição em N

Ele poderia ser reescrito como a seguir, para evidenciar hipótese e tese:

> se 0 é elemento neutro da adição em N, então 0 é o único elemento neutro da adição em N

Desta forma, por causa de $p \Rightarrow q$, a hipótese p é suposta verdadeira e, consequentemente, ela não deve ser demonstrada.

Para um determinado teorema $p \rightarrow q$ destacam-se as seguintes técnicas de prova (demonstração) de que, de fato, $p \Rightarrow q$:

- prova direta
- prova por contraposição
- prova por redução ao absurdo
- prova por indução

Para qualquer técnica de demonstração deve-se dar atenção especial aos quantificadores. Para provar a proposição:

$$(\forall x \in A)p(x)$$

é necessário provar $p(x)$ para todo $x \in A$. Assim, mostrar um determinado elemento $a \in A$ não é uma prova, mas um exemplo, o que não constitui uma prova válida para todos os elementos de A. Já no caso de:

$$(\exists x \in A)p(x)$$

para provar que existe pelo menos um $a \in A$ tal que $p(x)$ é **verdadeira** basta mostrar um exemplo.

2.4.1 Prova direta

Uma prova é direta quando pressupõe verdadeira a hipótese e, a partir dela, prova ser verdadeira a conclusão.

Considere o teorema:

> a soma de dois números pares é um número par

que, reescrito na forma de $p \rightarrow q$, torna-se:

> se n e m são dois números pares quaisquer, então $n + m$ é um número par

Suponha que n e m são números pares. Deve-se mostrar que $n + m$ é par.

Pela definição de **par**:

> qualquer número par n pode ser definido como $n = 2r$, para algum natural r.

podemos afirmar que existem $r, s \in N$ tais que:

$$n = 2r \qquad \text{e} \qquad m = 2s$$

Então, aplicando essas definições em $n + m$:

$$n + m = 2r + 2s = 2(r + s)$$

A expressão $r + s$ é um número que, multiplicado por 2 é um número par. Logo, $n + m$ é um número par e prova-se verdadeira a conclusão.

A estrutura da prova direta é[1]:

1. expresse a afirmação a ser provada na forma $(\forall x \in A)p(x) \rightarrow q(x)$ (pode ser feito mentalmente)
2. comece a prova supondo que x é um elemento específico de A, mas escolhido arbitrariamente para o qual a hipótese $p(x)$ é verdadeira (normalmente abreviado para o formato: suponha $x \in Aep(x)$)
3. mostre que a conslução é verdadeira usando definições, resultados anteriores e as regras de inferência lógica

[1]Notas de aula da disciplina Matemática Discreta, do Departamento de Ciência da Computação da UFMG, ministrada pelo prof. Antonio Alfredo Ferreira Loureiro. On-line: http://homepages.dcc.ufmg.br/~loureiro/md/md_3MetodosDeProva.pdf

Capítulo 3

Álgebra de Conjuntos

Uma **Álgebra** é constituída de operações definidas sobre uma soleção de objetos. Neste contexto, *álgebra de conjuntos* corresponderia às operações definidas sobre todos os conjuntos.

As operações da álgebra de conjuntos podem ser:

- Não reversíveis
  - União
  - Intersecção
  - Diferença
- Reversíveis
  - Complemento
  - Conjunto das partes
  - Produto cartesiano
  - União disjunta

3.1 Operações não reversíveis

3.1.1 União

Sejam A e B dois conjuntos. A união entre eles, $A \cup B$, é definida como:

$$A \cup B = \{x \mid x \in A \vee x \in B\} \tag{3.1}$$

Considerando a lógica, o conjunto A pode ser definido como $x \in A$ e o conjunto B pode ser definido como $x \in B$. Ou seja, a propriedade de pertiência é utilizada para indicar uma proposição lógica.

A união corresponde à operação lógica *disjunção* (símbolo $\vee$).

Exemplo: união entre conjuntos

Considere os conjuntos:

Exemplo: união entre conjuntos

a) Digitos = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
b) Vogais = $\{a, e, i, o, u\}$
c) Pares = $\{0, 2, 4, 6, ...\}$

Então:
a) Digitos $\cup$ Vogais = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, e, i, o, u\}$
b) Digitos $\cup$ Pares = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, ...\}$

Suponha os conjuntos:
a) $A = \{x \in N \mid x > 2\}$
b) $B = \{x \in N \mid x^2 = x\}$

Então:
$A \cup B = \{0, 1, 3, 4, 5, 6, ...\}$

É importante observar que o resultado da união é um conjunto sem repetições de elementos.

Vejamos as propriedades da união:

- **Elemento neutro**: $A \cup \emptyset = \emptyset \cup A = A$
- **Idempotência**: $A \cup A = A$
- **Comutativa**: $A \cup B = B \cup A$
- **Associativa**: $A \cup (B \cup C) = (A \cup B) \cup C$

3.1.2 Intersecção

Sejam dois conjuntos A e B. A intersercção entre eles, $A \cap B$ é definida como:

$$A \cap B = \{x \mid x \in A \wedge x \in B\} \tag{3.2}$$

A união corresponde à operação lógica *conjunção* (símbolo $\wedge$).

Exemplo: intersecção

Considere os conjuntos:
a) Digitos = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
b) Vogais = $\{a, e, i, o, u\}$
c) Pares = $\{0, 2, 4, 6, ...\}$

Então:
a) Digitos $\cap$ Vogais = $\emptyset$
b) Digitos $\cap$ Pares = $\{0, 2, 4, 6, 8\}$

Também podemos utilizar **Diagrama de Venn** para demonstrar a Intersecção, como ilustra a Figura 3.1.

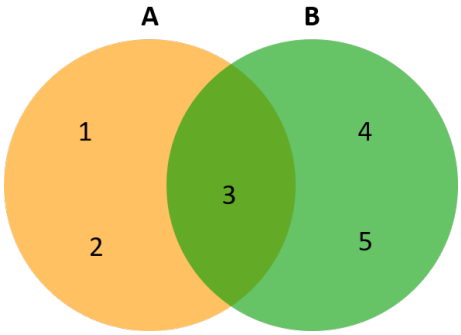

Figura 3.1: Diagrama de Venn demonstrando a intersecção entre conjuntos A e B

A Figura 3.1 considera os conjuntos $A = \{1, 2, 3\}$ e $B = \{3, 4, 5\}$ e mostra o resultado de $A \cap B$. A área mais ao centro, colorida como uma mistura das cores dos conjuntos, corresponde à intersecção (não é necesssário utilizar cores, entretanto).

Sejam A e B dois conjuntos não vazios. Se $A \cap B = \emptyset$, então A e B são chamados *conjuntos disjuntos*, *conjuntos independentes*, ou *conjuntos mutuamente exclusivos*.

Propriedades da intersecção:

- **Elemento neutro**: $A \cap U = U \cap A = A$
- **Idempotência**: $A \cap A = A$
- **Comutativa**: $A \cap B = B \cap A$
- **Associativa**: $A \cap (B \cap C) = (A \cap B) \cap C$

3.2 Propriedades envolvendo união e intersecção

As propriedades a seguir envolvem as operações de união e intersecção:

- **Distributividade da intersecção sobre a união**: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- **Distributividade da união sobre a intersecção**: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- **Absorção**: $A \cap (A \cup B) = A$ e $A \cup (A \cap B) = A$.

3.3 Operações reversíveis

Entende-se por **operação reversível** uma operação a partir de cujo resultado pode-se recuperar os operandos originais.

3.3.1 Complemento

Considere o conjunto universo U. O complemento de um conjunto $A \subseteq U$, denotado por $\sim A$ é definido como:

$$\sim A = \{x \in U \mid x \notin A\} \tag{3.3}$$

Exemplos: complemento

1. Considere o conjunto universo definido por Digitos $= \{0, 1, 2, ..., 9\}$. Seja $A = \{0, 1, 2\}$. Então, $\sim A = \{3, 4, 5, 6, 7, 8, 9\}$.

2. Suponha conjunto universo $\mathbb{N}$. Seja $A = \{0, 1, 2\}$. Então $\sim A = \{x \in \mathbb{N} \mid x > 2\}$.

3. Para qualquer conjunto universo U, valem:
a) $\sim \emptyset = U$
b) $\sim U = \emptyset$

4. Suponha que U é o conjunto universo. Então, para qualquer conjunto $A \subseteq U$, valem:
a) $A \cup \sim A = U$
b) $A \cap \sim A = \emptyset$

No último exemplo, observe que:

- a união de um conjunto com seu complemento sempre resulta no conjunto universo ($p \vee \sim p =$ Verdadeiro); e
- a intersecção de um conjunto com seu complemento sempre resulta no conjunto vazio ($p \wedge \sim p =$ Falso).

Também vale a noção de *duplo complemento* (ou *dupla negação*):

$$\sim\sim A = A \tag{3.4}$$

A propriedade denominada **DeMorgan** vale-se do complemento, envolvendo as operações de união e interseção (Tabela 3.4).

Tabela 3.4: DeMorgan na álgebra de conjuntos e na Lógica

DeMorgan na Álgebra de conjuntos	Propriedade DeMorgan na Lógica
$\sim (A \cup B) = \sim A \cap \sim B$	$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$
$\sim (A \cap B) = \sim A \cup \sim B$	$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$

3.3.2 Diferença

Sejam os conjuntos A e B. A diferença dos conjuntos A e B, denotada por $A - B$ é definida como:

$$A - B = A \cap \sim B \tag{3.5}$$

ou

$$A - B = \{x \mid x \in A \wedge x \notin B\} \tag{3.6}$$

Exemplos: diferença

1. Suponha os conjuntos: Digitos $= \{0, 1, 2, ..., 9\}$, Vogais $= \{a, e, i, o, u\}$ e Pares $= \{0, 2, 4, 6, ...\}$. Utilizando a diferença, temos:
a) Digitos $-$ Vogais $=$ Digitos
b) Digitos $-$ Pares $= \{1, 3, 5, 7, 9\}$

2. Para qualquer conjunto universo U e qualquer $A \subseteq U$, valem:
a) $\emptyset - \emptyset = \emptyset$
b) $U - \emptyset = U$
c) $U - A = \sim A$
d) $U - U = \emptyset$

3.3.3 Conjunto das partes

Para qualquer conjunto A sabe-se que:

- $A \subseteq A$
- $\emptyset \subseteq A$
- Para qualquer elemento $a \in A$, é visível que $\{a\} \subseteq A$

A operação unária chamada *conjunto das partes*, ao ser aplicada ao conjunto A, resulta no conjunto de todos os subconjuntos de A. Suponha um conjunto A. O conjunto das partes de A (ou conjunto potência), denotado por $\mathrm{P}(A)$ ou 2^A, é definido por:

$$\mathrm{P}(A) = \{X \mid X \subseteq A\} \tag{3.7}$$

Exemplos: conjunto das partes

Sejam os conjuntos $A = \{a\}$, $B = \{a, b\}$, $C = \{a, b, c\}$ e $D = \{a, \emptyset, \{a, b\}\}$, então:
1. $\mathrm{P}(\emptyset) = \{\emptyset\}$
2. $\mathrm{P}(A) = \{\emptyset, \{a\}\}$
3. $\mathrm{P}(B) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
4. $\mathrm{P}(C) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$
5. $\mathrm{P}(D) = \{\emptyset, \{a\}, \{\emptyset\}, \{\{a, b\}\}, \{a, \emptyset\}, \{a, \{a, b\}\}, \{\emptyset, \{a, b\}\}, \{a, \emptyset, \{a, b\}\}\}$

3.3.4 Produto Cartesiano

A operação **produto cartesiano** é uma operação binária que, quando aplicada a dois conjuntos A e B, resulta em um conjunto constituído de sequências de duas componentes (tuplas), sendo que a primeira componente de cada sequência é um elemento de A, e a segunda componente, um elemento de B.

Uma sequência de n componentes, denominada *n-upla ordenada* (lê-se: ênupla ordenada), consiste de n objetos (não necessariamente distintos) em uma ordem fixa. Por exemplo, uma 2-upla (tupla) ordenada é denominada *par ordenado*. Um par ordenado no qual a primeira componente é x e a segunda é y é definido como $\langle x, y\rangle$ ou (x, y).

Uma n-upla ordenada é definida como:

$$\langle x_1, x_2, x_3, ..., x_n\rangle \tag{3.8}$$

Uma n-upla ordenada não deve ser confundida com um conjunto, pois a ordem das componentes é importante. Assim:

$$\langle x, y\rangle \neq \langle y, x\rangle \tag{3.9}$$

O **produto cartesiano** dos conjuntos A e B, denotado por $A \times B$ é definido por:

$$A \times B = \{\langle a, b\rangle \mid a \in A \wedge b \in B\} \tag{3.10}$$

O produto cartesiano de um conjunto com ele mesmo é definido por $A \times A = A^2$

Exemplos: produto cartesiano

Sejam os conjuntos $A = \{a\}$, $B = \{a, b\}$ e $C = \{0, 1, 2\}$. Então:
1. $A \times B = \{\langle a, a\rangle, \langle a, b\rangle\}$
2. $B \times C = \{\langle a, 0\rangle, \langle a, 1\rangle, \langle a, 2\rangle, \langle b, 0\rangle, \langle b, 1\rangle, \langle b, 2\rangle\}$
3. $A^2 = \{\langle a, a\rangle\}$

3.3.5 União disjunta

Diferentemente da *união*, que desconsidera repetições de elementos no conjunto resultante, a *união disjunta* permite que os elementos do conjunto resultante sejam duplicados, uma vez que seja identificada a sua fonte. A *união disjunta* dos conjuntos A e B, denotada por $A + B$ ou $A\dot{\cup}\, B$ é definida como:

$$A + B = \{\langle a, A\rangle \mid a \in A\} \cup \{\langle b, B\rangle \mid b \in B\} \tag{3.11}$$

ou

$$A + B = \{a_A \mid a \in A\} \cup \{b_B \mid b \in B\} \tag{3.12}$$

Exemplo: união disjunta

Suponha os conjuntos Silva = {João, Maria, José} e Souza = {Pedro, Ana, José}. Então:
1. Silva + Souza =
{⟨João, Silva⟩, ⟨Maria, Silva⟩, ⟨José, Silva⟩, ⟨Pedro, Souza⟩, ⟨Ana, Souza⟩, ⟨José, Souza⟩}

ou

2. Silva + Souza = $\{\text{João}_{\text{Silva}}, \text{Maria}_{\text{Silva}}, \text{José}_{\text{Silva}}, \text{Pedro}_{\text{Souza}}, \text{Ana}_{\text{Souza}}, \text{José}_{\text{Souza}}\}$

3.4 Relações entre a Lógica e as operações sobre conjuntos

É possível estabelecer uma relação entre a lógica e as operações da álgebra de conjuntos, como mostra a Tabela 3.9.

Tabela 3.9: Conectivos lógicos x operações sobre conjuntos

Propriedade	Lógica	Teoria dos conjuntos
idempotência	$p \wedge p \Leftrightarrow p$	$A \cap A = A$
	$p \vee p \Leftrightarrow p$	$A \cup A = A$
comutativa	$p \wedge q \Leftrightarrow q \wedge p$	$A \cap B = B \cap A$
	$p \vee q \Leftrightarrow q \vee p$	$A \cup B = B \cup A$
associativa	$p \wedge (q \wedge r) \Leftrightarrow (p \wedge q) \wedge r$	$A \cap (B \cap C) = (A \cap B) \cap C$
	$p \vee (q \vee r) \Leftrightarrow (p \vee q) \vee r$	$A \cup (B \cup C) = (A \cup B) \cup C$
distributiva	$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
	$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
negação ou	$\neg\neg p \Leftrightarrow p$	$\sim \sim A = A$
complemento	$p \wedge \neg p \Leftrightarrow F$	$A \cap \sim A = \emptyset$
	$p \vee \neg p \Leftrightarrow V$	$A \cup \sim A = U$
DeMorgan	$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$	$\sim (A \cup B) = \sim A \cap \sim B$
	$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$	$\sim (A \cap B) = \sim A \cup \sim B$
elemento	$p \wedge V \Leftrightarrow p$	$A \cap U = A$
neutro	$p \vee F \Leftrightarrow p$	$A \cup \emptyset = A$

Propriedade	Lógica	Teoria dos conjuntos
elemento	$p \wedge F \Leftrightarrow F$	$A \cap \emptyset = \emptyset$
absorvente	$p \vee V \Leftrightarrow V$	$A \cup U = U$
absorção	$p \wedge (p \vee q) \Leftrightarrow p$	$A \cap (A \cup B) = A$
	$p \vee (p \wedge q) \Leftrightarrow p$	$A \cup (A \cap B) = A$

Ainda, as relações lógicas e as relações sobre conjuntos são análogas:

- **implicação/contigência**: na lógica: $p \Rightarrow q$, na teoria dos conjuntos: $A \subseteq B$
- **equivalência/igualdade**: na lógica: $p \Leftrightarrow q$, na teoria dos conjuntos: $A = B$

Como já visto, $p(x)$ é uma proposição que descreve uma propriedade de um elemento $x \in A$. Assim, a continência e a igualdade de dois conjuntos $A = \{x \mid p(x)\}$ e $B = \{x \mid q(x)\}$ pode ser vista assim:

- **contingência**: $A \subseteq B$ se e somente se $(\forall x \in U)(p(x) \Rightarrow q(x))$
- **igualdade**: $A = B$ se e somente se $(\forall x \in U)(p(x) \Leftrightarrow q(x))$

Exemplos:

- $A = U$ se e somente se $(\forall x \in U)(p(x) \Leftrightarrow V)$
- $A = \emptyset$ se e somente se $(\forall x \in U)(p(x) \Leftarrow F)$

3.5 Provando propriedades

3.5.1 Prova da propriedade *elemento neutro da união*

Elemento neutro é definido como:

$$A \cup \emptyset = \emptyset \cup A = A \tag{3.13}$$

Assim, há duas igualdades, que podem ser analisadas considerando a validade da transitividade [da igualdade]. Assim, temos que observar alguns casos:

(A) Para provar $A \cup \emptyset = \emptyset \cup A$:

O primeiro caso (1): Seja $x \in A \cup \emptyset$. Então devemos provar que $A \cup \emptyset \subseteq \emptyset \cup A$:

- $x \in A \cup \emptyset \Rightarrow$ (definição de união)
- $x \in A \vee x \in \emptyset \Rightarrow$ (comutatividade da disjunção)
- $x \in \emptyset \vee x \in A \Rightarrow$ (definição de união)
- $x \in \emptyset \cup A$

Portanto, $A \cup \emptyset \subseteq \emptyset \cup A$.

O segundo caso (2): Seja $x \in \emptyset \cup A$. Então devemos provar que $\emptyset \cup A \subseteq A \cup \emptyset$:

- $x \in \emptyset \cup A \Rightarrow$ (definição de união)
- $x \in \emptyset \vee x \in A \Rightarrow$ (comutatividade da disjunção)
- $x \in A \vee x \in \emptyset \Rightarrow$ (definição de união)
- $x \in A \cup \emptyset$

Portanto, $\emptyset \cup A = A \cup \emptyset$.

Terceiro caso (3): De (1) e (2) concluímos que $A \cup \emptyset = \emptyset \cup A$.

(B) Para provar $A \cup \emptyset = A$:

Quarto caso (4): Seja $x \in A \cup \emptyset$. Então devemos provar que $A \cup \emptyset \subseteq A$:

- $x \in A \cup \emptyset \Rightarrow$ (definição de união)
- $x \in A \vee x \in \emptyset \Rightarrow$ ($x \in \emptyset$ é sempre *false*)
- $x \in A$

Portanto, $A \cup \emptyset \subseteq A$.

Quinto caso (5): Seja $x \in A$. Então devemos provar que $A \subseteq A \cup \emptyset$:

- $x \in A \Rightarrow$ ($x \in A$ é sempre *true*, portanto podemos considerar $p \Rightarrow p \vee q$)
- $x \in A \vee x \in \emptyset$ (definição de união)
- $x \in A \cup \emptyset$

Portanto, $A \subseteq A \cup \emptyset$.

Sexto caso (6): De (4) e (5) concluímos que $A \cup \emptyset = A$.

Sétimo caso (7): Por fim, de (3) e (6) e pela transitividade da igualdade, concluímos que $A \cup \emptyset = \emptyset \cup A = A$ e provamos a propriedade do *elemento neutro* da união.

3.6 Exercícios

Exercício 1: Suponha o conjunto universo $S = \{p, q, r, s, t, u, v, w\}$ bem como os seguintes conjuntos:

- $A = \{p, q, r, s\}$
- $B = \{r, t, v\}$
- $C = \{p, s, t, u\}$

Determine:

a) $B \cap C$

b) $A \cup C$

c) $\sim C$

d) $A \cap B \cap C$

e) $\sim (A \cup B)$

f) $(A \cup B) \cap \sim C$

Exercício 2: Considere os conjuntos $A = \{a\}$, $B = \{a, b\}$ e $C = \{0, 1, 2\}$. Calcule os seguintes produtos cartesianos:

1. $(A \times B) \times C$
2. $A \times (B \times C)$
3. B^2
4. C^2

Exercício 3: Considere os conjuntos $D = \{0, 1, 2, ..., 9\}$, $V = \{a, e, i, o, u\}$, e $P = \{0, 2, 4, 6, ...\}$. Então, encontre:

1. $D + V$
2. $D + P$
3. $V + V$
4. $V + \emptyset$

Exercício 4: Utilizando um banco de dados relacional, crie duas tabelas: *Palavras1* e *Palavras2*, respectivamente. Utilizando linguagem SQL, crie e apresente o resultado de uma consulta que realiza o produto cartesiano entre as duas tabelas.

Exercício 5: Crie um programa que lê n arquivos de entrada. Cada arquivo contém uma palavra em cada linha. O programa deve ler os arquivos e gerar um arquivo de saída chamado *pc.txt* contendo o produto cartesiano entre as palavras dos arquivos de entrada. Cada linha do arquivo de saída deve representar um elemento do produto cartesiano (uma n-upla) cujos componentes devem estar separados por um espaço [em branco]. **Exemplo**: Para os arquivos de entrada:

palavras1.txt
jose
maria

palavras2.txt
silva
santos

palavras3.txt
moreira
aires

o arquivo resultante seria:

pc.txt
jose silva moreira

pc.txt
jose silva aires
jose santos moreira
jose santos aires
maria silva moreira
maria silva aires
maria santos moreira
maria santos aires

Exercício 6: Faça uma pesquisa sobre as provas das propriedades das operações da álgebra de conjuntos e, em formato técnico-científico, apresente cada uma. Não se esqueça de apresentar as referências.

3.7 Projeto do capítulo

Este capítulo deu continuidade à Seção 1.2 e apresentou operações e propriedades sobre conjuntos que constituem a **Álgebra de conjuntos**.

Como forma de fundamentar os conceitos apresentados este capítulo traz o **projeto do capítulo**, uma atividade prática que envolve computação e escrita de textos.

O projeto deste capítulo deve alcançar os objetivos:

1. utilizar uma linguagem de programação para criar uma estrutura de dados **Conjunto**, com as funcionalidades:
   - adicionar elemento
   - remover elemento
   - verificar pertinência
   - verificar continência
   - realizar união (com outro conjunto)
   - realizar intersecção (com outro conjunto)
   - realizar diferença (com outro conjunto)
   - realizar complemento (em relação a outro conjunto)
   - gerar o conjunto das partes
   - realizar o produto cartesiano (com outro conjunto)
   - realizar a união disjunta (com outro conjunto)
2. escrever um texto didático explicando e demonstrando a implementação e os cocneitos utilizados. Para cada funcionalidade, apresentar: o conceito (definição), a operação (como funciona), a demonstração (com exemplos). Esse texto deve usar notação matemática (o resultado é um arquivo **PDF**).

Importante: No *Objetivo 1* não pode ser utilizado recurso da linguagem que já implemente recursos de conjuntos e opreações sobre conjuntos. Utilize lista, vetor ou outra estrutura de dados semelhante.

Por fim, o conteúdo deve ser disponibilizado em um repositório do Github. Deve haver um arquivo `README.md` identificando e apresentando o trabalho, bem como descrevendo os procedimentos adotados.

Capítulo 4

Relações

Considere as relações a seguir:

- parentesco
- maior ou igual
- igualdade
- lista telefônica, que associa assinante e seu número de telefone
- fronteira (entre países)

Suponha os conjuntos A e B. Uma relação R de A em B é um subconjunto de um produto cartesiano $A \times B$, ou seja:

$$R \subseteq A \times B \tag{4.1}$$

ou

$$R : A \to B \tag{4.2}$$

onde:

- A é o **domínio** (origem ou conjunto de partida de R)
- B é o **contradomínio** (destino ou conjunto de chegada de R)

Os elementos de R são pares (ou tuplas) dos elementos de A e B, ou seja:

$$(a, b) \in R$$

ou

$$aRb$$

Uma relação pode ser caracterizada por uma propriedade. Neste caso, na notação $R : A \to B$, R assume uma propriedade. Por exemplo:

$$=: A \to B$$

define a relação de igualdade entre os conjuntos A e B tal que os elementos de A sejam iguais aos elementos de B. Em outras palavras, os componentes das tuplas da relação R terão que respeitar a propriedade de igualdade.

Exemplo: Considere os seguintes conjuntos: $A = \{a\}$, $B = \{a, b\}$ e $C = \{0, 1, 2\}$. É válido afirmar que:

- $\{\}$ é uma relação de A em B pois o conjunto vazio é subconjunto de qualquer conjunto
- $A \times B = \{(a, a), (a, b)\}$ é uma relação (qualquer) de A em B
- $=: A \to A = \{(a, a)\}$ (relação de igualdade de A em A)
- $=: A \to B = \{(a, a)\}$ (relação de igualdade de A em B)
- $<: C \to C = \{(0, 1), (1, 2)\}$ (relação "menor que" de C em C)

4.1 Endorrelação

Endorrelação é um tipo de relação que considera o mesmo conjunto, ou seja, $R : A \to A$ (domínio e contradomínio são o mesmo conjunto). Uma endorrelação $R : A \to A$ pode ser denotada por (A, R), por exemplo:

a) $(N, <=)$
b) $(P(A), \subseteq)$

4.2 Representações gráficas de relações

As relações podem ser representadas graficamente como diagramas de Venn, gráficos em um plano cartesiano ou em grafos.

4.2.1 Relações como diagramas de Venn

Diagramas de Venn podem ser utilizados para representar relações visualmente. Na Figura 4.1 o diagrama da esquerda representa a relação $=: A \to A$ e o da direita a relação $<=: C \to C$.

Em Diagramas de Venn o conjunto da esquerda é comumente chamado de *domínio* e o da direita de *imagem*.

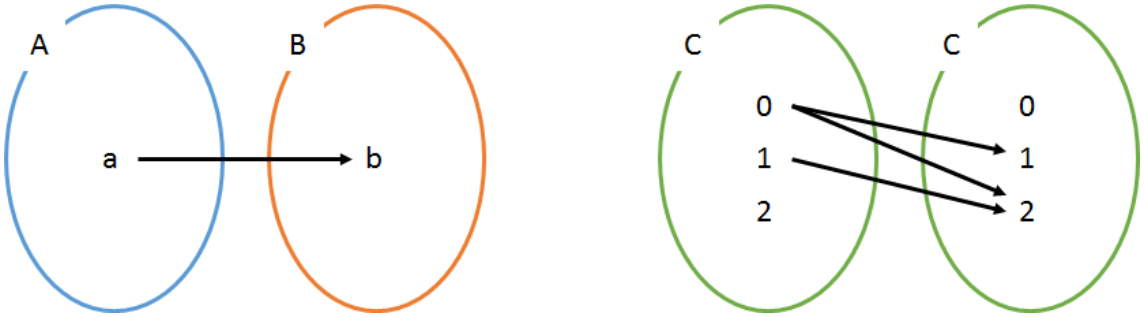

Figura 4.1: Diagrama de Venn demonstrando relações entre conjuntos

4.2.2 Relações como planos cartesianos

Gráficos em planos cartesianos podem representar relações. Em um plano cartesiano de duas dimensões, os eixos (coordenadas) representam os conjuntos. Por exemplo a relação $R = \{(x, y) \mid y = x^2\}$ pode ser representada pelo gráfico da Figura 4.2.

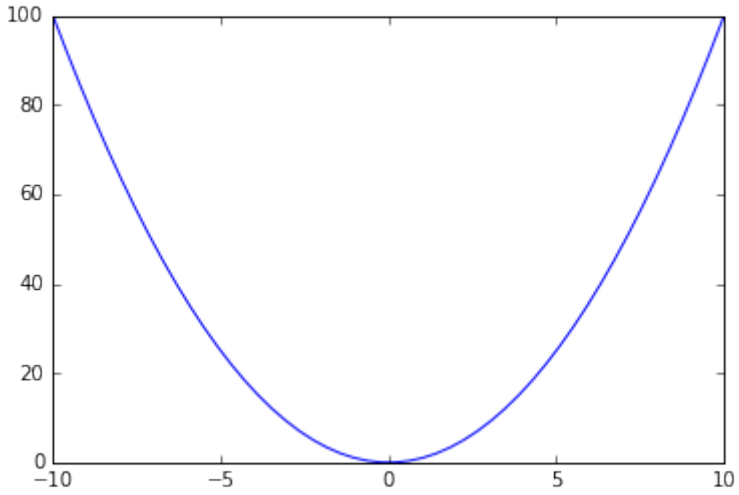

Figura 4.2: Gráfico demonstrando relações entre conjuntos

4.2.3 Relações como grafos

A representaçao de grafos aplica-se apenas a endorrelações. Grafos podem ser representados como matrizes e como diagramas e vértices e arestas.

Considere os conjuntos $A = \{a\}$, $B = \{a, b\}$ e $C = \{0, 1, 2\}$.

A relação $=: B \to B$, definida por $\{(a, a), (b, b)\}$ seria representada pela matriz e pelo grafo da Figura 4.3.

A relação $<=: C \to C$ seria representada pela matriz e pelo grafo da Figura 4.4:

=	a	b
a	1	0
b	0	1

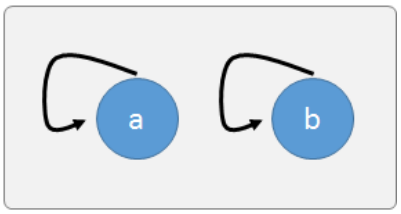

Figura 4.3: Grafo 1 - demonstrando relações entre conjuntos

<=	0	1	2
0	1	1	1
1	0	1	1
2	0	0	1

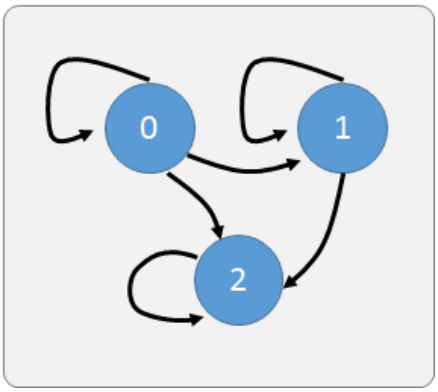

Figura 4.4: Grafo 2 - demonstrando relações entre conjuntos

4.3 Exercícios

1. Considere os seguintes conjuntos: $A = \{a\}$, $B = \{a, b\}$ e $C = \{0, 1, 2\}$. Defina as relações a seguir:

a) $\neq : B \to C$
b) $\neq : A \to B$

2. Considere o conjunto $A = \{1, 2, 3, 4, 5\}$. Represente as relações a seguir na forma de Diagrama de Venn, Plano cartesiano e Grafo:

a) $= : A \to A$
b) $(A, =)$
c) $(A, <=)$
d) $(A, >=)$
e) $R = \{(x, y) \mid y = x + y\}$

4.4 Propriedades de endorrelações

4.4.1 Reflexiva, irreflexiva

Seja $R : A \to A$, então R é uma:

a) **relação reflexiva**, se $(\forall a \in A)(a\, R\, a)$: para todo elemento a de A existe uma tupla (a, a)

b) **relação irreflexiva** (ou **antirreflexiva**), se $(\forall a \in A)(\neg(a\,R\,a))$: para todo elemento a de A não existe uma tupla (a, a).

Uma forma rápida de identificar a relação R no conjunto A como reflexiva ou irreflexiva é realizar os seguintes passos:

1. Definir o produto cartesiano de A
2. Definir $C_i \subseteq R$ no qual todas as tuplas tenham componentes iguais (primeiro igual ao segundo)
3. Definir $R \subseteq C_d$ no qual todas as tuplas tenham componentes diferentes (primeiro é diferente do segundo)

Assim, podemos definir que:

- a relação R é reflexiva se $C_i \subseteq R$
- a relação R é irreflexiva se $R \subseteq C_d$

Exemplo: Seja $A = \{a, b\}$. Conjuntos importantes:

- $A^2 = \{(a, a), (a, b), (b, a), (b, b)\}$.
- $C_i = \{(a, a), (b, b)\}$
- $C_d = \{(a, b), (b, a)\}$

As seguintes relações são:

a) Reflexivas:

- $R_1 = \{(a, a), (b, b)\}$
- $R_2 = \{(a, a), (a, b), (b, b)\}$, pois $C_i \subseteq R_2$
- $R_3 = \{(a, a), (b, b), (b, a)\}$, pelo mesmo motivo de R_2
- $R_4 = \{(a, a), (a, b), (b, a), (b, b)\}$, idem

b) Irreflexivas:

- $R_5 = \{(a, b), (b, a)\}$
- $R_6 = \{(a, b)\}$, pois $R_6 \subseteq C_d$
- $R_7 = \{(b, a)\}$, pelo mesmo motivo de R_6
- $R_8 = \{\}$, idem

Exemplo: Seja $S = \{0, 1, 2\}$. Conjuntos importantes:

- $S^2 = \{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)\}$
- $C_i = \{(0, 0), (1, 1), (2, 2)\}$
- $C_d = \{(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)\}$

A seguinte relação não é reflexiva e nem irreflexiva:

$$(S, R) = \{(0, 2), (2, 0), (2, 2)\}$$

Não é reflexiva porque $C_i \nsubseteq (S, R)$ e não é irreflexiva porque $(S, R) \nsubseteq C_d$.

4.4.2 Reflexividade de relações em matrizes e grafos

Para um conjunto A é possível verificar se uma endorrelação $R : A \to A$ é reflexiva ou irreflexiva analisando a sua representação como matriz ou grafo:

a) **Matriz**

- **Reflexiva**: a diagonal da matriz contém somente o valor verdadeiro
- **Irreflexiva**: a diagonal da matriz contém somente o valor falso

b) **Grafo**

- **Reflexiva**: qualquer nó tem um arco com origem e destino nele mesmo
- **Irreflexiva**: qualquer nó não tem um arco com origem e destino nele mesmo

4.4.3 Simétrica, antissimétrica

Seja $R : A \to A$ uma endorrelação em A, então R é uma:

a) **relação simétrica**, se $(\forall a, b \in A)(a\,R\,b \to b\,R\,a)$: para todo par de elementos a e b de A, sendo $a = b$ ou não, se há uma tupla (a, b) então há uma tupla (b, a)

b) **relação antissimétrica**, $(\forall a, b \in A), (a\,R\,b \wedge a \neq b) \implies (\neg b\,R\,a)$: para todo par de elementos a e b de A, se há uma tupla (a, b) e $a \neq b$, então não há a tupla (b, a).

Exemplos: Seja $A = \{a, b\}$. Conjuntos importantes:

- $A^2 = \{(a, a), (a, b), (b, a), (b, b)\}$
- $C_s = A^2$
- $C_a = \{(a, a), (b, b)\}$

As seguintes relações são:

a) Simétricas:

- $R_1 = \{(a, a), (b, b)\}$
- $R_4 = \{(a, a), (a, b), (b, a), (b, b)\}$
- $R_5 = \{(a, b), (b, a)\}$
- $R_8 = \{\}$
- $R_9 = \{(a, a)\}$
- $R_{10} = \{(b, b)\}$
- $R_{11} = \{(a, a), (a, b), (b, a)\}$
- $R_{12} = \{(a, b), (b, a), (b, b)\}$

b) Antissimétricas:

- $R_1 = \{(a, a), (b, b)\}$
- $R_2 = \{(a, a), (a, b), (b, b)\}$
- $R_3 = \{(a, a), (b, b), (b, a)\}$
- $R_6 = \{(a, b)\}$
- $R_8 = \{\}$

4.4.4 Simétrica e antissimétrica em matrizes e grafos

Para um conjunto A uma forma de entender e verificar se uma endorrelação $R : A \to A$ é simétrica ou antissimétrica é analisar a sua representação como matriz ou grafo:

a) **Matriz**

- Simétrica: a metade acima ou abaixo da diagonal é espelhada na outra
- Antissimétrica: se uma célula em uma metade da diagonal for verdadeira, a correspondente na outra metade é falsa

b) **Grafo**

- Simétrica: entre dois nós: a) não existe seta; ou b) existem duas setas, uma em cada sentido
- Antissimétrica: há no máximo uma seta entre dois nós quaisquer

4.4.5 Transitiva, não transitiva

Seja A um conjunto e R uma endorrelação em A, então R é uma:

a) **transitiva** se $(\forall a, b, c \in A), (a\,R\,b \wedge b\,R\,c \to a\,R\,c)$; e
b) **não-transitiva** é o contrário da relação transitiva

4.5 Atividade de pesquisa

1. Uma relação pode ser classificada nos seguintes tipos (que não são mutuamente exclusivos):

a) funcional
b) injetora
c) total
d) sobrejetora
e) monomorfismo
f) epimorfismo
g) isomorfismo

Escolha três dos tipos e, para cada um, apresente:

a) definição (incluindo representação formal)
b) dois exemplos

2. As *Redes de Petri* são o modelo computacional do tipo concorrente mais utilizado em computação, informática e engenharia. Faça uma pesquisa sobre Redes de Petri e apresente:

a) definição (conceito)
b) significado dos símbolos da representação gráfica da rede
c) dois exemplos
d) como definir uma Rede de Petri utilizando Relação (utilizando exemplos)

Capítulo 5

Contagem

Considere o seguinte algoritmo, em pseudocódigo:

```
1  Algoritmo ordena(A)
2  n = tamanho(A)
3  para i de 1 até n - 1
4      para j de i + 1 até n
5          se (A[i] > A[j]) então
6              troca(A, i, j)
```

Os algoritmos `tamanho(lista)` e `troca(lista, i, j)` representam, respectivamente:

- retorna o tamanho da `lista`
- troca os valores dos índices `i` e `j` da `lista`

No algoritmo `ordena(lista)`, quantas vezes é executado o condicional?

Analisamos isso da seguinte forma:

- da **primeira** vez: para $i = 1$, $j = \{2, ..., n\}$, executa $n - 1$ vezes
- da **segunda** vez: para $i = 2$, $j = \{3, ..., n\}$, executa $n - 2$ vezes
- da **terceira** vez: para $i = 3$, $j = \{4, ..., n\}$, executa $n - 3$ vezes
- da **quarta** vez: para $i = 4$, $j = \{5, ..., n\}$, executa $n - 4$ vezes
- da **i-ésima** vez, executa $n - i$ vezes.

Assim, o condicional é executado tantas vezes conforme indica a Eq. 5.1, para $n > 0, i = [1, (n-1)]$:

$$(n-1) + (n-2) + ... + (n-i) \tag{5.1}$$

Para $n = 5$, por exemplo, temos:

$$
\begin{aligned}
(n-1)+(n-2)+(n-3)+(n-4) = \\
(5-1)+(5-2)+(5-3)+(5-4) = \\
4+3+2+1 = \\
10
\end{aligned}
$$

O código a seguir, em Python, demonstra esse comportamento:

```
lista = [1, 2, 3, 4, 5]
n = len(lista)
k = 0
for i in range(n - 1): # range(n) gera n números, de 0...(n-1)
    for j in range(i + 1, n): # range(i, n) gera números de i...(n-1)
        print('Comparando A[%d] > A[%d]' % (i, j))
        k = k + 1
print('O condicional executou {} vezes'.format(k))
```

A saída seria algo como o seguinte:

```
Comparando A[0] > A[1]
Comparando A[0] > A[2]
Comparando A[0] > A[3]
Comparando A[0] > A[4]
Comparando A[1] > A[2]
Comparando A[1] > A[3]
Comparando A[1] > A[4]
Comparando A[2] > A[3]
Comparando A[2] > A[4]
Comparando A[3] > A[4]
O condicional executou 10 vezes
```

5.1 Utilizando a teoria dos conjuntos

Podemos melhorar o raciocínio para interpretar a contagem da quantidade de comparações utilizando a linguagem de conjuntos. Suponha que o conjunto S contenha todas as comparações feitas pelo algoritmo, representadas por uma tupla (i, j) em que i representa o primeiro índice da lista e j, o segundo. Considere $n = |S|$. Dividimos o conjunto S em $n-1$ partes (subconjuntos):

- o conjunto S_1 de comparações quando $i = 1$
- o conjunto S_2 de comparações quando $i = 2$
- o conjunto S_{n-1} de comparações quando $i = n-1$

Assim, descobrimos o número de comparações em cada conjunto S_i e adicionamos para obter o

tamanho do conjunto S.

Exemplo:

Para $n = 5$:

- $S_1 = \{(1,2),(1,3),(1,4),(1,5)\}$; $|S_1| = n - 1 = 5 - 1 = 4$
- $S_2 = \{<2,3>,<2,4>,<2,5>\}$; $|S_2| = n - 2 = 5 - 2 = 3$
- $S_3 = \{<3,4>,<3,5>\}$; $|S_3| = n - 3 = 5 - 3 = 2$
- $S_4 = \{<4,5>\}$; $|S_4| = n - 4 = 5 - 4 = 1$

5.2 Princípio da adição

Cada conjunto S_i é um **conjunto disjunto** dos demais porque as comparações que cada um contém são diferentes das presentes nos outros conjuntos. Em outras palavras, conjuntos são chamados disjuntos quando não possuem elementos em comum. Assim, $S = \{S_1, S_2, ..., S_m\}$ (com $m = n-1$) é uma família de conjuntos mutuamente disjuntos. Seguindo esse pensamento podemos estabelecer o **Princípio da adição**:

> **Princípio da adição**
>
> O tamanho da união de uma família de conjuntos finitos mutuamente disjuntos é a soma dos tamanhos dos conjuntos.

Podemos aplicar o princípio da adição para resolver o problema em questão. Considere que $|S|$ indique o tamanho do conjunto S, então:

$$|S_1 \cup S_2 \cup ... \cup S_m| = |S_1| + |S_2| + ... + |S_m|$$

ou, usando a notação de união:

$$\left|\bigcup_{i=1}^{m} S_i\right| = \sum_{i=1}^{m} |S_i|$$

Exemplo:

Para $n = 5$:

$$\begin{aligned}\left|\bigcup_{i=1}^{m} S_i\right| &= |S_1| + |S_2| + |S_3| + |S_4| \\ &= 4 + 3 + 2 + 1 \\ &= 10\end{aligned}$$

Quando $S = \bigcup_{i=1}^{m} S_i$, então temos um S particionado nos conjuntos $S_1, S_2, ..., S_m$ e estes formam uma **partição** de S.

Exemplo:

- $S = \{1, 2, 3, 4, 5\}$
- O conjunto $P_1 = \{\{1\}, \{2, 3\}, \{4, 5\}\}$ é uma partição de S
- S pode ser particionado nos conjuntos $\{1\}$, $\{2, 3\}$ e $\{4, 5\}$, elementos de P_1, que são chamados **blocos da partição** P_1.

Isso permite redefinir o **Princípio da adição**:

> **Princípio da adição** (*redefinido*)
>
> Se um conjunto finito S foi dividido em *blocos*, então o tamanho de S é a soma do tamanho dos blocos.

5.3 Soma de inteiros consecutivos

Voltando para a Eq. 5.1, ela pode ser escrita como:

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

Demonstramos assim, para $n = 5$:

$$\sum_{i=1}^{n-1}(n-i) = (5-1) + (5-2) + (5-3) + (5-4) = 4 + 3 + 2 + 1 = 10$$

Isso mostra que, na prática:

$$\sum_{i=1}^{n-1}(n-i) = 4 + 3 + 2 + 1 = 1 + 2 + 3 + 4 = \sum_{i=1}^{n-1} i$$

Entretanto, podemos tentar encontrar uma forma de reduzir ou simplificar essa equação. Vamos experimentar o seguinte:

1. Somar $n-1$ números, de 1 a $n-1$: $s_1 = 1 + 2 + ... + (n-1)$
2. Somar $n-1$ números, de $n-1$ a 1: $s_2 = (n-1) + ... + 2 + 1$
3. Somar n números $n-1$ vezes: $s_3 = n + n + ... + n$

Exemplo, para $n = 5$:

1. $s_1 = 1 + 2 + 3 + 4$
2. $s_2 = 4 + 3 + 2 + 1$
3. $s_3 = 5 + 5 + 5 + 5$

Esses artifícios mostram que $s_3 = s_1 + s_2$ e esse raciocínio leva a algumas conclusões:

1. Podemos escrever $s_3 = n \times (n-1)$
2. $s_3 = s_1 + s_2$
3. Logo, podemos afirmar que $\frac{s_3}{2} = s_1 = s_2$

A conclusão final desse raciocínio é que:

$$\sum_{i=1}^{n-1}(n-i) = \sum_{1=1}^{n-1} i = \frac{n \times (n-1)}{2}$$

Esse artifício foi utilizado por *Carl Friedrich Gauss* e, por fim, fornece uma forma útil para situações que envolvam encontrar a soma de uma série de valores ou variáveis.

Exemplo: Para $n = 5$, encontrar $S = 1 + 2 + ... + (n-1)$:

$$S = \frac{5 \times (5-1)}{2} = \frac{20}{2} = 10$$

5.4 Princípio do produto

Seguindo a mesma abordagem para entender o princípio da adição, veja o seguinte algoritmo `multiplica_matrizes()`, que encontra o produto de duas matrizes ($A \times B = C$):

```
1  Algoritmo multiplica_matrizes(A, B)
2  r, n = dimensoes(A) # A tem r = linhas, n = colunas
3  n, m = dimensoes(B) # B tem n = linhas, m = colunas
4  C = matriz(r, m)    # C tem r = linhas, m = colunas
5  para i de 1 até r
6      para j de 1 até m
7          s = 0
8          para k de 1 até n
9              s = s + A[i][k] * B[k][j]
10         C[i][j] = s
11 retorne C
```

Onde os algoritmos `dimensoes(A)` e `matriz(m,n)` representam, respectivamente:

- as dimensões (linha x coluna) da matriz `A`
- criação de uma matriz com as dimensões `m` x `n`

Quantas multiplicações (`A[i][k] * B[k][j]`) o algoritmo executa ao final de todas as iterações (em termos de `r`, `m`, e `n`)?

Vejamos:

- o laço `para` mais interno (linha 8) executa n multiplicações
- o laço `para` anterior (linha 6), repete o laço mais interno m vezes. Assim, executa $n \times m$ multiplicações

Pensando no contexto de conjuntos, como foi feito na abstração do princípio da adição, temos que para todo $i = 1, ..., r$ o conjunto de multiplicações pode ser dividido em:

- conjunto S_1 de multiplicações quando $j = 1$
- conjunto S_2 de multiplicações quando $j = 2$
- conjunto S_j de multiplicações para todo j.

Seja T_i o conjunto das multiplicações para todo i. Este conjunto é a união dos conjuntos S_j:

$$T_i = \bigcup_{j=1}^{m} S_j$$

Usando o princípio da adição, o tamanho de T_i é igual à soma dos tamanhos de cada conjunto S_j. A soma de m números, cada um igual a n, é $m \times n$, como mostra a Eq. 5.2, que define o **Princípio do produto**:

$$|T_i| = \left| \bigcup_{j=1}^{m} S_j \right| = \sum_{j=1}^{m} |S_j| = \sum_{j=1}^{m} n = m \times n \tag{5.2}$$

Princípio do produto

O tamanho da união de conjuntos disjuntos m, sendo cada um de tamanho n, é $m \times n$.

O laço `para` mais externo (linha 5) executa uma vez para cada valor de $i = 1, ..., r$, por isso o podemos afirmar que existem r conjuntos T_i. Portanto, pelo princípio do produto, concluímos que o algoritmo do produto entre duas matrizes executa $r \times m \times n$ multiplicações.

5.5 Subconjunto com dois elementos

Voltemos ao algoritmo usado no princípio da adição. Algumas considerações:

- quando $i = 1$, $j = 2, ..., n$
- quando $i = 2$, $j = 3, ..., n$

Para cada número de i e j, é feita a comparação entre $A[i]$ e $A[j]$ exatamente uma vez. Desta forma, o número de comparações é o mesmo que o número de subconjuntos de dois elementos do conjunto $\{1, 2, ..., n\}$. Uma questão importante é: **de quantas maneiras diferentes podemos escolher dois elementos desse conjunto?**

Antes de responder essa pergunta é importante apresentar (ou relembrar) o conceito de **par ordenado**: um par ordenado contém um elemento na primeira posição e outro na segunda. Por exemplo, temos o par ordenado $(2, 5)$ – que é diferente do par ordenado $(5, 2)$, pois a ordem dos elementos é importante aqui.

O problema se torna descobrir a quantidade de pares ordenados que podem ser criados com os elementos do conjunto em questão.

O raciocínio é este: ao escolhermos determinados primeiro e segundo elemento, existem n maneiras

de se escolher o primeiro elemento, e para cada escolha dele, há $n-1$ maneiras de se escolher o segundo. Numericamente, para $n = 5$, fazemos:

- ao escolher 1 para primeiro elemento do par são formados os pares $(1,2), (1,3), (1,4), (1,5)$
- ao escolher 2 para primeiro elemento do par são formados os pares $(2,1), (2,3), (2,4), (2,5)$
- ao escolher 3 para primeiro elemento do par são formados os pares $(3,1), (3,2), (3,4), (3,5)$
- ao escolher 4 para primeiro elemento do par são formados os pares $(4,1), (4,2), (4,3), (4,5)$
- ao escolher 5 para primeiro elemento do par são formados os pares $(5,1), (5,2), (5,3), (5,4)$

Se representarmos essas escolhas usando conjuntos $(C_1, ..., C_5)$ podemos afirmar que o conjunto C de todas as escolhas é igual à união de n conjuntos de tamanho $n-1$, ou seja:

$$C = \bigcup_{i=1}^{n} C_i$$

Pelo princípio do produto, poderíamos concluir que a quantidade de pares ordenados é $n \times (n-1)$. Portanto, para $n = 5$, teríamos $5 \times (5-1) = 20$.

Ao escolher os elementos para formar os pares, entretanto, chegamos ao seguinte raciocínio:

- escolher 2 e depois 5, para formar o par $(2,5)$ **ou**
- escolher 5 e depois 2, para formar o par $(5,2)$

Essa é uma consideração importante porque as duas escolhas são mutuamente exclusivas.

Portanto, o número de subconjuntos com dois elementos em $\{1, 2, ..., n\}$ é $n \times (n-1)/2$. Isso também pode ser representado usando a notação $\binom{n}{2}$, que pode ser lida como n **termos,** 2 **a** 2 ou n **escolha** 2. Assim:

$$1 + 2 + ... + (n-1) = \binom{n}{2} = \frac{n \times (n-1)}{2}$$

O seguinte programa Python demonstra esses conceitos na prática:

```
n = 5

S = [i for i in range(1, n + 1)]
print('S = {', ', '.join([str(i) for i in S]), '}')

Pares = []
for i in range(n):
    for j in range(i + 1, n):
        Pares.append('(%d, %d)' % (i + 1, j + 1))

print('Pares = {', ', '.join([i for i in Pares]), '}')

print(len(Pares), ' Pares')
```

A saída do programa seria:

```
S = { 1, 2, 3, 4, 5 }
Pares = { (1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4),
    (3, 5), (4, 5) }
10  Pares
```

5.6 Contagem de listas, permutações e conjuntos

Para utilizar os princípios da adição e do produto na prática, considere o seguinte problema:

> Uma senha de um certo sistema de computador deveria ter de quatro a oito caracteres e conter letras minúsculas e/ou maiúsculas. Quantas senhas são possíveis?

Para resolver o problema, o raciocínio pode partir do princípio de que as senhas podem ter quatro, cinco, seis, sete ou oito caracteres. Como o conjunto de todas as senhas é a união dos conjuntos com senhas de tamanho quatro a oito, podemos usar o princípio da adição.

Considere P_i como o conjunto das senhas com i caracteres e P é o conjunto com todas as senhas.

$$P = P_4 \cup P_5 \cup P_6 \cup P_7 \cup P_8$$

Assim:

$$|P| = \sum_{i=4}^{8} |P_i|$$

Qual o valor de $|P_i|$? Para isso precisamos considerar que para uma senha com i letras existem 52 escolhas para a primeira letra, 52 para a segunda e assim por diante. Pelo princípio do produto:

$$|P_i| = 52^i$$

Portanto, o número total de senhas, $|P|$ é:

$$|P| = 52^4 + 52^5 + 52^6 + 52^7 + 52^8$$

A porcentagem de senhas com quatro letras é obtida como:

$$\frac{52^4}{|P|} \times 100$$

Esse raciocínio é útil para chegar à conclusão de que é mais fácil encontrar uma senha que já sabemos ter quatro caracteres do que entre quatro e oito caracteres e leva a uma redefinição (ou segunda versão) do princípio do produto.

Princípio do produto (versão 2)

Se um conjunto S de listas de comprimento m tem as seguintes propriedades:

- existem i_1 primeiros elementos diferentes das listas em S, e
- para cada $j > 1$ e cada escolha dos $j-1$ elementos de uma lista em S, existem i_j escolhas de elementos na posição j daquelas listas, então, existem $i_1 \times i_2 \times ... \times i_m = \prod_{k=1}^{m} i_k$ listas em S.

Ao aplicar esse princípio ao problema da contagem das senhas temos: como uma senha com m letras é uma lista com m elementos, e por existirem 52 primeiros elementos diferentes da senha e 52 escolhas para cada outra posição da senha, temos que $i_1 = 52, i_2 = 52, ..., i_m = 52$, assim, o número de senhas de comprimento m é $i_1 \times i_2 \times ... \times i_m = 52^m$.

5.7 Listas e funções

Definição formal de lista:

> Uma **lista** de k coisas escolhidas a partir de T consiste em um primeiro membro de T até o k-ésimo membro de T. Por exemplo, uma lista de 3 coisas escolhidas de um conjunto T consiste nos elementos t_1, t_2, t_3, todos membros de T. Os elementos da lista não são, necessariamente, diferentes uns dos outros. Se a lista for escrita em uma ordem diferente, será uma lista diferente.

Definição formal de função:

> Uma **função** de um conjunto S (o *domínio*) para um conjunto T (o *contradomínio*) é um relacionamento entre os elementos de S e T.

Por exemplo:

$$f(1) = Samuel \qquad f(2) = Maria \qquad f(3) = Sara$$

indica que f é uma função que descreve uma lista de três nomes. Assim, temos uma definição: **uma lista de k elementos a partir de um conjunto T é uma função de $\{1, 2, ..., k\}$ para T.**

Exemplo: Quais são as funções do conjunto $\{1, 2\}$ para o conjunto $\{a, b\}$?

Para resolver a questão, precisamos especificar $f_i(1)$ e $f_i(2)$, enumerando-as:

$f_i(1)$	$f_i(2)$
$f_1(1) = a$	$f_1(2) = b$
$f_2(1) = b$	$f_2(2) = a$
$f_3(1) = a$	$f_3(2) = a$
$f_4(1) = b$	$f_4(2) = b$

O conjunto de todas as funções do conjunto $\{1, 2\}$ para o conjunto $\{a, b\}$ é a união entre as funções f_i com $f_i(1) = a$ e aquelas com $f_i(1) = b$. O conjunto de funções com $f_i(1) = a$ tem dois elementos, um para cada escolha de $f_i(2)$:

- o conjunto $f_i(1) = a = \{f_1(2) = b, f_3(2) = a\}$
- o conjunto $f_i(1) = b = \{f_2(2) = a, f_4(2) = b\}$

Pelo princípio do produto, a resposta para a questão é $2 \times 2 = 4$.

O número de funções do conjunto $\{1, 2\}$ para o conjunto $\{a, b, c\}$ é a união dos três conjuntos, um para cada escolha de $f_i(1)$, cada qual possuindo três elementos, um para cada escolha de $f_i(2)$. Assim, pelo princípio do produto, temos: $3 \times 3 = 9$.

O número de funções do conjunto $\{1, 2, 3\}$ para o conjunto $\{a, b\}$ é a união de quatro conjuntos, um para cada escolha de $f_i(1)$ e $f_i(2)$, cada qual possuindo dois elementos, um para cada escolha de $f_i(3)$. Assim, pelo princípio do produto, temos $4 \times 2 = 8$.

Uma função f de um conjunto S para um conjunto T é chamada **um a um**, ou **injetora**, se, para cada $x \in S$ e $y \in S$, com $x \neq y$, $f(x) \neq f(y)$. No exemplo anterior as funções f_1 e f_2 são injetoras, mas f_3 e f_4 não são.

Uma função f de um conjunto S para um conjunto T é chamada **sobrejetora** se, para cada elemento $y \in T$ (no contradomínio), existe um $x \in S$, tal que $f(x) = y$. No exemplo anterior, as funções f_1 e f_2 são sobrejetoras, mas f_3 e f_4 não são.

5.8 Permutações de k elementos de um conjunto

Uma lista de k elementos distintos de um conjunto N é chamada **permutação de k elementos** de N.

Existem n escolhas para o primeiro elemento da lista, para cada uma existem $(n-1)$ escolhas para o segundo. Para cada escolha dos dois primeiros elementos da lista, há $(n-2)$ formas de escolher o terceiro elemento, e assim por diante.

Por exemplo, se $N = \{1, 2, 3, 4\}$, as permutações de $k = 3$ elementos são:

$$L = \{123, 124, 132, 134, 142, 143, \tag{5.3}$$
$$213, 214, 231, 234, 241, 243, \tag{5.4}$$
$$312, 314, 321, 324, 341, 342, \tag{5.5}$$
$$412, 413, 421, 423, 431, 432\} \tag{5.6}$$

Ou seja, existem $4 \times 3 \times 2 = 24$ listas com $k = 3$ elementos em N. Pelo princípio do produto (versão 2) isso é $n \times (n-1) \times (n-2)$.

Dados os primeiros $i-1$ elementos da lista, temos $n - (i-1) = n - i + 1$ escolhas para o i-ésimo elemento da lista.

Pelo princípio do produto (versão 2) temos $n \times (n-1) \times ... \times (n-k+1)$ maneiras de escolher permutações de k elementos de N. De outra forma:

$$n \times (n-1) \times ... \times (n-k+1) = \prod_{i=0}^{k-1}(n-i) \tag{5.7}$$

Teorema 2.1

O número de permutações de k elementos de um conjunto de n elementos é:

$$n \times (n-1) \times ... \times (n-k+1) = \prod_{i=0}^{k-1}(n-i) = \frac{n!}{(n-k)!} \tag{5.8}$$

5.9 Contando subconjuntos de um conjunto

Usamos anteriormente $\binom{n}{3}$, que lemos "n termos, 3 a 3", para representar o número de subconjuntos com três elementos do conjunto $\{1, 2, ..., n\}$.

O número de subconjuntos de k elementos do conjunto $\{1, 2, ..., n\}$ é $\binom{n}{k}$, que lemos como "n termos, k a k".

Como o número de permutações de k elementos de um conjunto com k elementos é $k!$ temos:

$$\frac{n!}{(n-k)!} = \binom{n}{k} \times k! \tag{5.9}$$

Isso dá o **Teorema 2.2**:

Para inteiros n e k, com $0 \leq k \leq n$, o número de subconjuntos de k elementos de um conjunto de n elementos é:

$$C_{n,k} = \binom{n}{k} = \frac{n!}{k! \times (n-k)!} \tag{5.10}$$

A notação $C_{n,k}$ gera os chamados **coeficientes binomiais**.

5.10 Atividade prática

Estude a relação entre o coeficiente binomial e o Triângulo de Pascal e use esse conhecimento para criar um programa que crie o Triângulo de Sierpinski (use, por exemplo, a cor preta para número ímpar e a cor branca para número par). Exemplo:

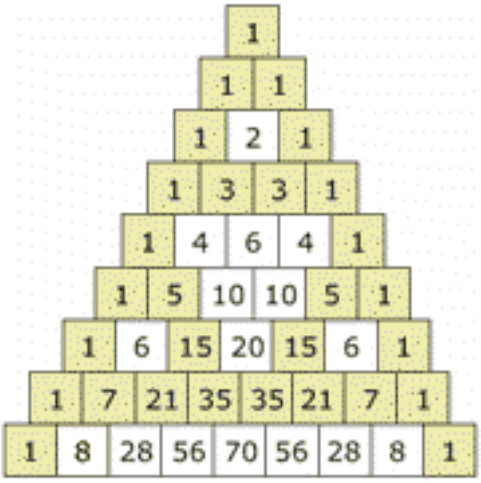

Figura 5.1: Triângulo de Sierpinski

Capítulo 6

Probabilidade

6.1 Definições iniciais

Experimento Uma ocorrência com um **resultado** incerto. Também pode ser chamado "processo".

Exemplos:

- rolar um dado
- um usuário utilizar uma interface com vários botões que podem ser pressionados sem uma ordem pré-estabelecida

Resultado O resultado de um **experimento**; um "estado" particular do mundo (ambiente). Também pode ser chamado "caso".

Exemplos:

- 4 (como resultado do experimento "rolar um dado");
- usuário pressionou o botão "Sair" (como resultado do experimento "usuário utilizar uma interface com vários botões")

Espaço amostral O conjunto de todos os **resultados** possíveis para um experimento.

Exemplos:

- $lados = \{1, 2, 3, 4, 5, 6\}$;
- $botoes = \{\text{Salvar}, \text{Sair}, \text{Cancelar}, \text{Ajuda}\}$

Evento Um subconjunto do **espaço amostral**.

Exemplos:

- o evento "dado com face par" é o conjunto dos resultados $\{2, 4, 6\}$
- o evento "usuário pressionou botão do formulário" é o conjunto dos resultados $botoes = \{\text{Salvar}, \text{Cancelar}\}$

Probabilidade A probabilidade de um **evento** em relação a um **espaço amostral** é a quantidade de **resultados** no evento dividida pela quantidade de resultados no espaço amostral. Como é uma razão, o valor de uma probabilidade será **sempre** um número entre 0 (um evento impossível) e 1 (um evento certo). Entretanto, é importante considerar que é bastante comum encontrar a porcentagem apresentada entre 0 e 100, ou seja, é o valor entre 0 e 1 multiplicado por 100.

Exemplo:

- a probabilidade do evento "dado com face par" é $3/6 = 1/2 = 0.5$ ou 50 %
- a probabilidade do evento "usuário pressionou botão do formulário" é o conjunto é $2/4 = 0.5$

6.2 A função $P()$

$P()$ é o nome tradicional para a função **Probabilidade**:

```
1  from fractions import Fraction
2
3  def P(evento, espaco):
4      "A probabilidade de um `evento`, dado o `espaco` amostral"
5      return Fraction(len(evento & espaco), len(espaco))
```

O código implementa o conceito "Probabilidade é simplemente uma fração cujo numerador é o número de casos favorávels e cujo denominador é o número de todos os casos possíveis" (Laplace). Os parâmetros `evento` e `espaco` são conjuntos.

6.3 Aquecimento: Rolar um dado

Qual é a probabilidade de rolar um dado e a face superior ser um número par? Consideramos, para isso, um dado com seis faces.

Podemos definir o espaço amostral A (o conjunto das faces do dado), o evento *par* (o conjunto dos resultados que atendem ao **predicado** "dado com face par") e calcular a probabilidade:

```
1  A = {1, 2, 3, 4, 5, 6}
2  par = {2, 4, 6}
3  print(P(par, A))
```

Resultado:

```
1  1/2
```

O tipo do resultado de `P(par, A)` é `Fraction`, portanto o resultado do código também poderia ser `Fraction(1, 2)`. O código utiliza o tipo `Fraction` para que o resultado seja apresentado como fração, não como número real. Obviamente, $1/2 = 0.5$.

Você pode perguntar: "por que não usar `len(evento)` ao invés de `len(evento & espaco)`? Isso é necessário porque não podem ser considerados resultados que não estejam presentes no espaço amostral. Veja só:

```
par = {2, 4, 6, 8, 10, 12}
print(P(par, A))
```

Teria como resultado:

```
1/2
```

Se considerasse apenas `len(evento)`, o retorno seria outro:

```
Fraction(len(par), len(A))
```

Resultado:

```
Fraction(1, 1)
```

Os "casos favoráveis", da definição de Laplace, resultam da interseção evento ∩ espaco. No código da função `p()`, considerando que os parâmetros sejam conjuntos, isso é feito usando o operador `&`, por isso `len(evento & espaco)`.

6.3.1 Sobre o tipo `Fraction` em Python

O tipo `Fraction` do Python é usado para representar valores na forma de frações (racional). Por exemplo:

```
print(Fraction(0.5))
print(Fraction(0.3))

# usa limit_denominator() para encontrar a fração mais aproximada
print(Fraction(0.3).limit_denominator())

print(Fraction(3, 10))
print(Fraction(0.3333333))

# limitando o denominador (padrão é max=1000000)
print(Fraction(0.3333333).limit_denominator())

```

```
# convertendo o valor de Fraction() para float()
print(float(Fraction(0.3333333)))
```

Resultado:

```
1/2
5404319552844595/18014398509481984
3/10
3/10
6004798902680711/18014398509481984
1/3
0.3333333
```

6.4 Problemas com urnas

Problemas com urnas surgiram por volta do ano 1700, com o matemático *Jacob Bernoulli*. Por exemplo:

- Uma urna contém 23 bolas: 8 brancas, 6 azuis e 9 vermelhas. Selecionamos seis bolas aleatoriamente (cada seleção com a mesma chance de acontecer). Qual é a probabilidade desses três resultados possíveis:

1. todas as bolas são vermelhas
2. 3 são azuis, 2 são vermelhas e 1 é branca
3. 4 são brancas

Então, um resultado é um **conjunto** com 6 bolas, enquanto o espaço amostral é o conjunto de todas as possíveis combinações (também conjuntos) com 6 bolas. Antes de continuar a solução, duas questões:

1. há múltiplas bolas da mesma cor (ex: 8 bolas brancas)
2. um resultado é um **conjunto** de bolas, então a ordem dos seus elementos não importa (diferentemente da **lista** – ou sequência – para a qual a ordem importa)

Para resolver a primeira questão, vamos nomear as bolas usando uma letra e um número, assim:

- as 8 bolas brancas serão chamadas `W1` até `W8`
- as 6 bolas azuis serão chamadas `B1` até `B6`
- as 9 bolas vermelhas serão chamadas `R1` até `R9`

Para resolver a segunda questão teremos que trabalhar com permutações e então encontrar combinações, dividindo o número de permutações por $c!$, onde c é o número de bolas em uma combinação.

Por exemplo, se eu precisar escolher 2 bolas de 8 disponíveis, há 8 formas de escolher a primeira, e 7 formas de escolher a segunda. Sendo assim, há $8 \times 7 = 56$ permutações, mas $52/2 = 26$ combinações. Tudo isso porque $\{W1, W2\} = \{W2, W1\}$.

Vamos lá. O código a seguir define a função `cross()` e o conteúdo da urna:

```
def cross(A, B):
    "O conjunto de formas de concatenar os itens de A e B (produto
        cartesiano)"
    return {a + b
            for a in A for b in B
           }

urna = cross('W', '12345678') | cross('B', '123456') | cross('R', '
    123456789')
```

Os parâmetros `A` e `B` devem ser iteráveis, ou seja, podem ser conjuntos, listas ou... strings.

O código para definir a variável `urna` utiliza a função `cross()` para gerar três conjuntos, um para cada cor de bola, e depois gera a união entre eles (utilizando o operador `|`).

A variável `urna` é um conjunto de 23 elementos $\{B1, B2, ..., B6, R1, R2, ..., R9, W1, W2, ..., W8\}$.

Agora, vamos definir o espaço amostral, chamado `U6`, o conjunto de todas as combinações com 6 bolas. O código a seguir define a função `combos()`:

```
import itertools

def combos(items, n):
    "Todas as combinações de n items; cada combinação concatenada em uma
        string"
    return {' '.join(combo)
            for combo in itertools.combinations(items, n)
           }

U6 = combos(urna, 6)
```

Os parâmetros são:

- `items`: deve ser uma lista ou um conjunto
- `n`: deve ser um número inteiro que representa o tamanho de cada combinação

A função `combos()` utiliza o pacote `itertools` e a função `combinations()` e concatena o resultado dela para gerar um conjunto de strings como resultado final.

A variável `U6` contém o conjunto de todas as combinações (resultados possíveis) com 6 bolas. O tamanho dessa variável é 100947 (a quantidade de combinações). O código a seguir usa o pacote `random` para apresentar uma amostra de tamanho 10 (função `sample()`) dessa variável.

```
import random

print(random.sample(U6, 10))
```

Resultado:

```
['W2 W3 W6 R7 W1 W8',
 'B4 R9 W5 B2 R4 R5',
 'W5 R6 R3 W8 W7 R5',
 'W4 W5 R6 W7 R5 B6',
 'B1 W5 R8 W8 R4 W7',
 'B4 R9 B1 B5 R3 B2',
 'W2 W6 R7 R3 W8 B2',
 'W1 R3 W8 R4 R5 B6',
 'B4 R9 W1 B2 B3 B6',
 'R9 W3 W5 R3 B2 B3']
```

Mas... será que o 100.947 é a quantidade correta de formas de escolher 6 de 23 bolas? Bem, o raciocínio é esse:

- Escolha uma de 23 (sobram 22)
- Escolha uma de 22 (sobram 21)
- ...
- Escolha seis de 18

Como não ligamos para a ordem (enfim, é um **conjunto**), então dividimos por 6!. Isso dá:

$$23 \text{ escolha } 6 = \frac{23 \times 22 \times 21 \times 20 \times 19 \times 18}{6!} = 100947 \tag{6.1}$$

Note que $23 \times 22 \times 21 \times 20 \times 19 \times 18 = 23!/(23-6)!$, então, podemos generalizar:

$$n \text{ escolha } c = \frac{n!}{c! \times (n-c)!} = \binom{n}{c} \tag{6.2}$$

Podemos traduzir isso para código, definindo a função `escolha()`:

```
from math import factorial

def escolha(n, c):
    "Número de formas de escolher c itens de uma lista com n items"
    return factorial(n) // (factorial(c) * factorial(n - c))
```

Os parâmetros são:

- `n`: tamanho da lista
- `c`: quantos itens da lista (tamanho da combinação)

O código utiliza o pacote `math` e a função `factorial()` para encontrar o resultado do binomial. Em Python, o operador `/` é utilizado para uma divisão resultando em um número real, enquanto `//` resulta em um número inteiro.

Por exemplo:

```
escolha(23, 6)
```

Resulta em `100947`.

Agora temos uma API que permite resolver vários problemas. Relembrando:

- `P(evento, espaco)`: calcula a probabilidade de um `evento` dado um `espaco`
- `escolha(n, c)`: calcula o número de forma de escolher `c` itens de uma lista com `n` itens

Os problemas a seguir estão no contexto urna do enunciado anterior.

Problema 1: qual a probabilidade de selecionar 6 bolas vermelhas?

O problema considera a mesma definição da `urna` apresentada anteriormente. A resolução envolve o código a seguir.

```
red6 = {s for s in U6 if s.count('R') == 6}
prob_red6 = P(red6, U6)
```

O código define o conjunto `red6`, que resulta de um **filtro** na variável `U6`: utiliza a função `count(n)` para identificar quais os elementos de `U6` possuem 6 ocorrências da letra `R`, ou seja, 6 bolas vermelhas.

O resultado desse raciocínio: a probabilidade de escolher 6 bolas vermelhas é `4/4807 = 0.0008` (0.08 %) ou `len(red6)/ len(U6)= 84 / 100.947` (`len(red6)= 84` indica que há 84 escolhas possíveis).

Por que 84 escolhas possíveis? Por que há 9 bolas vermelhas na urna, então estamos querendo saber quantas são as formas de escolher 6 delas:

```
escolha(9, 6)
```

O resultado é `84`.

Portanto, a probabilidade de selecionar 6 bolas vermelhas da urna é `escolha(9, 6)` dividido pelo tamanho do espaço amostral:

```
P(red6, U6) == Fraction(escolha(9, 6), len(U6))
```

O código verifica se o valor de `P(red6, U6)` é igual a `Fraction(escolha(9, 6), len(U6))`. O resultado é: `True`.

Importante: Reveja a definição da modelagem do problema e do raciocínio da sua resolução quantas vezes forem necessárias para fixar o entendimento.

Problema 2: qual a probabilidade de escolher 3 azuis, 2 brancas e 1 vermelha?

De forma semelhante ao problema anterior, primeiro é necessário criar um modelo que representa o ennciado (o evento), ou seja: no espaço amostral, quais os possíveis resultados têm três bolas azuis, duas brancas e uma vermelha? Para isso, temos o código:

```
b3w2r1 = {s for s in U6 if s.count('B') == 3 and s.count('W') == 2 and s.
    count('R') == 1}

prob_b3w2r1 = P(b3w2r1, U6)
```

O código encontra os resultados que têm três letras "B", duas "W" e 1 "R" e, a partir dele, encontra a probabilidade desejada. O resultado é: $240/4807 = 0.05$ ou 5 %.

Podemos encontrar o mesmo resultado contando de quantas formas podemos escolher 3 de 6 azuis, 2 de 8 brancas, 1 de 9 vermelhas e dividindo o resultado pelo tamanho do espaço amostral. Assim, o código a seguir verifica essa igualdade:

```
P(b3w2r1, U6) == Fraction(escolha(6, 3) * escolha(8, 2) * escolha(9, 1),
    len(U6))
```

O resultado é `True`.

O raciocínio seguinte também é válido:

- há 6 bolas azuis, então...
  - há 6 formas de encontrar a primeira bola azul
  - há 5 formas de encontrar a segunda azul
  - há 4 formas de encontrar a terceira azul
- há 8 bolas brancas, então...
  - há 8 formas de encontrar a primeira bola branca
  - há 7 formas de encontrar a segunda branca
- há 9 bolas vermelhas, então...
  - há 9 formas de encontrar a bola vermelha

Como $\{B1, B2, B3\} = \{B3, B2, B1\}$ e $\{W1, W2\} = \{W2, W1\}$, teríamos:

$$\frac{(6 \times 5 \times 4) \times (8 \times 7) \times (9)}{3! \times 2! \times |A|} \tag{6.3}$$

Ou, em código:

```
P(b3w2r1, U6) == Fraction((6 * 5 * 4) * (8 * 7) * 9,
                          factorial(3) * factorial(2) * len(U6))
```

O resultado é `True`.

Problema 3: Qual a probabilidade de termos exatamente 4 bolas brancas?

Podemos encontrar a resposta usando os mesmos raciocínios anteriores:

```
w4 = {s for s in U6 if
      s.count('W') == 4}

prob_w4 = P(w4, U6)
```

O resultado é `350/4807 = 0.07` ou `7.3%`. Para checar o resultado, temos:

```
P(w4, U6) == Fraction(escolha(8, 4) * escolha(15, 2), len(U6))
```

O resultado é `True`.

Ou, de outra forma:

```
P(w4, U6) == Fraction((8 * 7 * 6 * 5) * (15 * 14), factorial(4) *
    factorial(2) * len(U6))
```

O resultado é `True`.

Esse último raciocínio, em particular, é interpretado assim:

- há 8 bolas brancas, então...
  - 8 escolhas para a primeira bola
  - 7 escolhas para a segunda
  - 6 escolhas para a terceira
  - 5 escolhas para a segunda
- há $23 - 8 = 15$ bolas não brancas, então...
  - 15 escolhas para as outras bolas restantes (não brancas)
  - 14 escolhas para as outras bolas restantes (não brancas restantes)

Ou seja:

$$\frac{(8 \times 7 \times 6) \times (15 \times 14)}{4! \times 2! \times |A|} \tag{6.4}$$

6.5 Revisando a função P, com eventos mais gerais

Até o momento, para calcular a probabilidade de um dado com face par, usamos:

```
par = {2, 4, 6}
```

Entretanto, embora seja fácil enumerar um conjunto pequeno, é difícil enumerar um conjunto maior. Assim o código a seguir apresenta uma redefinição da função `P()` e a função `tal_que()`:

```
def P(evento, espaco):
    """A probabilidade de um evento, dado um espaco amostral.
    evento pode ser um conjunto ou um predicado"""
    if callable(evento):
        evento = tal_que(evento, espaco)
    return Fraction(len(evento & espaco), len(espaco))

def tal_que(predicado, colecao):
    "O subconjunto de elementos da colecao para os quais o predicado é
        verdadeiro"
    return {e for e in colecao if predicado(e)}
```

O código da função `P()` verifica se o parâmtro `evento` é uma função (aqui chamamos de "predicado"). Se sim, então executa a função `tal_que()` usando o predicado.

A função `tal_que()` aceita dois parâmetros:

- `predicado`: uma expressão que retorna um valor booleano
- `colecao`: o iterável (conjunto ou lista, por exemplo) sobre o qual o predicado será aplicado

O retorno da função é um conjunto com os elementos de `colecao` que satisfazem `predicado` (ou seja, o tornam `True`).

Vamos verificar como isso se comporta. O código a seguir define a função `eh_par()`, um predicado:

```
def eh_par(n):
    return n % 2 == 0
```

O resultado de `P(eh_par, A)` é `1/2`. Então, aplicando esse conceito, temos o conjunto `D12` (um conjunto com 12 números inteiros, de 1 a 12):

```
D12 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

pares = tal_que(eh_par, D12)
```

O resultado é que `pares` é igual a o conjunto $\{2, 4, 6, 8, 10, 12\}$, ou seja, os elementos de `D12` que satisfazem ao predicado (são pares). A probabilidade é calculada assim:

```
P(eh_par, D12)
```

O resultado é `1/2`, ou seja, a probabilidade de que um número entre 1 e 12 seja par é 0.5 ou 50%.

Isso permite coisas interessantes. Por exemplo, como determinar a probabilidade de que a soma de três dados é um número primo?

```
A3 = {(a1, a2, a3) for a1 in A for a2 in A for a3 in A}

```

```
def soma_eh_primo(r):
    return eh_primo(sum(r))

def eh_primo(n):
    return n > 1 and not any(n % i == 0 for i in range(2, n))

P(soma_eh_primo, A3)
```

O resultado é `73/216 = 0.34` ou 34%.

6.6 Problemas com cartas

Considerando as cartas de um baralho, definimos o problema assim:

- 4 naipes:
  - copas (C) (ou *hearts* - H)
  - ouro (O) (ou *diamond* - D)
  - paus (P) (ou *clubs* - C) e
  - espadas (E) (ou *spades* - S)
- 13 graus: 1 (Ás), 2, 3, 4, 5, 6, 7, 8, 9, 10 (ou T), 11 (J), 12 (Q), 13 (K)

O baralho possui 52 cartas. O código a seguir define o problema:

```
naipes = 'SHDC'
graus = 'A23456789TJQK'
baralho  = cross(graus, naipes)
```

O resultado de `len(baralho)` é 52.

Quais as jogadas (mãos) com 5 cartas?

```
jogadas5 = combos(baralho, 5)
```

Então podemos verificar que `len(jogadas5)== escolha(52, 5)` e podemos apresentar uma amostra do conjunto:

```
random.sample(jogadas, 5)
```

Resulta em:

```
['3C 3H 8H 2H AC',
 '8C KS 5C 2H 3S',
 'JS 6C JD KH JC',
 '7H 7C 9D 9H KH',
 '5S TC 5D 5C 4C']
```

Problema 1: Qual a probabilidade de dar *flush* (5 cartas do mesmo naipe)?

```
def flush(jogada):
    return any(jogada.count(n) == 5 for n in naipes)

prob_flush = P(flush, jogadas)
```

O resultado, o valor de `prob_flush`, é: `33/16660 = 0.002` ou 0.2%.

Problema 2: Qual a probabilidade de dar *four of a kind* (4 cartas iguais, mesmo com naipes diferentes) ?

```
def four_kind(jogada):
    return any(jogada.count(g) == 4 for g in graus)

prob_fk = P(four_kind, jogadas)
```

O resultado, o valor de `prob_fk`, é: `1/4165 = 0.0002` ou 0.02%.

```
Fraction(1, 4165)
```

Curioso sobre as jogadas do Pôker? Veja: https://pt.wikipedia.org/wiki/Tabela_de_jogadas_do_p%C3%B4quer.

6.7 Fermat e Pascal: Apostas, Triângulos e o nascimento da Probabilidade

Considere um jogo de apostas, consistindo do fato de jogar uma moeda. Dois jogadores: **H** aposta em cara; e **T** aposta em coroa. Ganha o jogador que conseguir 10 acertos primeiro. O que fazer se o jogo for interrompido quando H tiver acertado 8 e T, acertado 7? Como dividir o pote de dinheiro? Fermat e Pascam trocaram correspondências sobre esse problema, o que pode ser lido aqui.

Podemos resolver o problema com as ferramentas definidas anteriormente. Primeiro, alguns predicados:

```
def ganhar_jogo_incompleto(h_pontos, t_pontos):
    "A probabilidade de que H vai ganhar o jogo não terminado, dados os
        pontos necessários para H e T ganharem."
    def h_ganha(r): return r.count('h') >= h_pontos
    return P(h_ganha, continuacoes(h_pontos, t_pontos))

def continuacoes(h_pontos, t_pontos):
    "Todas as continuações possíveis quando H precisa de `h_pontos` e T
        precisa de `t_pontos`"
```

```
    rodadas = ['ht' for _ in range(h_pontos + t_pontos - 1)]
    return set(itertools.product(*rodadas))
```

O valor de `continuacoes(2, 3)` (os possíveis retornos restantes para os jogadores) contém:

```
{('h', 'h', 'h', 'h'),
 ('h', 'h', 'h', 't'),
 ('h', 'h', 't', 'h'),
 ('h', 'h', 't', 't'),
 ('h', 't', 'h', 'h'),
 ('h', 't', 'h', 't'),
 ('h', 't', 't', 'h'),
 ('h', 't', 't', 't'),
 ('t', 'h', 'h', 'h'),
 ('t', 'h', 'h', 't'),
 ('t', 'h', 't', 'h'),
 ('t', 'h', 't', 't'),
 ('t', 't', 'h', 'h'),
 ('t', 't', 'h', 't'),
 ('t', 't', 't', 'h'),
 ('t', 't', 't', 't')}
```

O resultado de `ganhar_jogo_incompleto(2, 3)` é `11/16 = 0.6875` ou a probabilidade de H ganhar o joogo é 68,75%.

O resultado confirma o encontrado por Pascal e Fermat.

6.8 Resultados não equiprováveis: Distribuições de probabilidade

Até aqui, lidamos com casos em que a probabilidade de um retorno no espaço amostral é a mesma (uniforme). No mundo real, isso não é sempre verdade. Por exemplo, a probabilidade de uma criança ser uma menina não é exatamente 1/2 (50%), e a probabilidade é um pouco diferente para uma segunda criança. Uma pesquisa encontrou as seguintes contagens de famílias com dois filhos na Dinamarca (**GB** significa uma família em que o primeiro filho é uma garota e o segundo filho é um menino):

```
GG: 121801     GB: 126840
BG: 127123     BB: 135138
```

Vamos introduzir mais três definições:

Frequência Um número que descreve o quão frequente um resultado ocorre. Pode ser uma contagem, como 121801, ou uma razão, como 0,515

Distribuição Um mapeamento de resultado para frequência, para cada resultado no espaço amostral.

Distribuição de Probabilidade Uma distribuição que foi *normalizada* de tal forma que a soma das frequências é 1

Seguindo essas definições, declaramos a classe `ProbDist` (um subtipo do `dict` do Python):

```
class ProbDist(dict):
    "Uma distribuição de probablidade; um mapeamento {resultado:
        probabilidade}"
    def __init__(self, mapping=(), **kwargs):
        self.update(mapping, **kwargs)
        total = sum(self.values())
        for outcome in self:
            self[outcome] = self[outcome]/total
            assert self[outcome] >= 0
```

A classe `ProbDist` cria um dicionário cujas chaves são resultados possíveis e os valores são probabilidades. Os nomes das chaves são definidos manualmente, mas as propriedades são criadas a partir de frequências:

```
ProbDist(sexo_f=1, sexo_m=1)
```

O trecho de código demonstra o uso da classe `ProbDist`, criando uma distribuição de probabilidade com dois resultados (`sexo_m` e `sexo_f`). Os valores de cada chave são informados como suas frequências e o código calcula a frequência normalizada (mantendo o valor entre 0 e 1). Desta forma, a classe representa que:

- frequência normalizada de `sexo_m`: $1/2 = 0.5$
- frequência normalizada de `sexo_f`: $1/2 = 0.5$

Também redefinimos as funções `P()` e `tal_que()`:

```
def tal_que(predicado, espaco):
    """Os resultados no espaço amostral para os quais o predicado é
        verdadeiro.
    Se espaco é um conjunto, retorna um subconjunto {resultado, ...}
    Se espaco é ProbDist, retorna um ProbDist{resultado, frequencia}"""
    if isinstance(espaco, ProbDist):
        return ProbDist({o:espaco[o] for o in espaco if predicado(o)})
    else:
        return {o for o in espaco if predicado(o)}


def P(evento, espaco):
    """A probabilidade de um evento, dado um espaço amostral de resultados
         equiprováveis.
```

```
    evento: uma coleção de resultados, ou um predicado.
    espaco: um conjunto de resultados ou a distribuicao de probabilidade
        na forma de pares {resultado: frequencia}.
    """
    if callable(evento):
        evento = tal_que(evento, espaco)
    if isinstance(espaco, ProbDist):
        return sum(espaco[o] for o in espaco if o in evento)
    else:
        return Fraction(len(evento & espaco), len(espaco))
```

A função `tal_que()` considera que o `espaco` pode ser `ProbDist` e, neste caso, retorna uma nova instância de `ProbDist` que é uma cópia de `espaco` desde que o `predicado` seja verdadeiro para cada chave.

A função `P()` leva em consideração que o `espaco` pode ser uma instância de `ProbDist` e, neste caso, o cálculo da probabilidade de o evento ocorrer é baseado em:

- se o `evento` for uma função (um predicado) então usa `tal_que()` para aplicar o predicado sobre o `espaco` e determinar o conjunto `evento`
- se o `espaco` for uma instância de `ProbDist` então calcula a soma dos valores (as frequências) das chaves que também estão em `espaco`

Vamos ver alguns exemplos práticos para fixar esses conceitos.

Aqui está a distribuição de probabilidade para as famílias Dinamarquesas com dois filhos:

```
DK = ProbDist(GG=121801, GB=126840, BG=127123, BB=135138)
```

O resultado, o valor de `DK`:

```
{
    'BB': 0.2645086533229465,
    'BG': 0.24882071317004043,
    'GB': 0.24826679089140383,
    'GG': 0.23840384261560926
}
```

Perceba que cada chave do dicionário `DK` não possui a frequência, mas a probabilidade.

Vamos entender o resultado em partes. Para isso, alguns predicados:

```
def primeiro_menina(r): return r[0] == 'G'
def primeiro_menino(r): return r[0] == 'B'
def segundo_menina(r): return r[1] == 'G'
def segundo_menino(r): return r[1] == 'B'
def duas_meninas(r): return r == 'GG'
def dois_meninos(r): return r == 'BB'
```

Estes predicados serão aplicados a cada chave para calcular as probabilidades:

- a probabilidade de que o primeiro filho seja uma menina é `P(primeiro_menina, DK)` com valor aproximado de `0.487`.
- a probabilidade de que o segundo filho seja uma menina é `P(segundo_menina, DK)` com valor aproximado de `0.487`.

Isso indica que a probabilidade de uma criança ser menina está entre 48% e 49%, mas isso é um pouco diferente entre o primeiro e o segundo filhos:

```
P(segundo_menina, tal_que(primeiro_menina, DK))
P(segundo_menina, tal_que(primeiro_menino, DK))
```

A primeira linha calcula a probilidade de o segundo filho ser menina, dado que o primeiro é uma menina. O resultado é `0.4898669165584115`. A segunda linha calcula a probabilidade de o segundo filho ser menina, dado que o primeiro é um menino. O resultado é `0.48471942072973107`.

E, para:

```
P(segundo_menino, tal_que(primeiro_menina, DK))
P(segundo_menino, tal_que(primeiro_menino, DK))
```

A primeira linha calcula a probabilidade de o segundo filho ser menino, dado que o primeiro é uma menina. O resultado é `0.5101330834415885`. A segunda linha calcula a probabilidade de o segundo filho ser menino, dado que o primeiro é um menino. O resultado é `0.5152805792702689`.

Isso diz que é mais provável que o sexo do segundo filho seja igual ao do primeiro (cerca de 50%).

6.9 Mais problemas de Urnas: M&Ms e Bayes

Outro problema de urna (Allen Downey):

> O M&M azul foi introduzido em 1995. Antes disso, a mistura de cores em um pacote de M&Ms era formado por: 30% marrom, 20% amarelo, 20% vermelho, 10% verde, 10% laranja, 10% tostado. Depois, ficou: 24% azul, 20% verde, 16% laranja, 14% amarelo, 14% vermelho, 13% marrom. Um amigo meu possui dois pacotes de M&Ms e ele me diz que um é de 1994 e outro é de 1996. Ele não me diz qual é qual, mas me dá um M&M de cada pacote. Um é amarelho e outro é verde. Qual a probabilidade de que o M&M amarelo seja do pacote de 1994?

Para resolver esse problema, primeiro representamos as distribuições de probabilidade de cada pacote:

```
bag94 = ProbDist(brown=30, yellow=20, red=20, green=10, orange=10, tan=10)
bag96 = ProbDist(blue=24, green=20, orange=16, yellow=14, red=13, brown
    =13)
```

A seguir, definimos `MM` como a *probabilidade conjunta* – o espaço amostral para escolher um M&M de cada pacote. O resultado `yellow green` significa que um M&M amarelo foi selecionado do pacote de 1994 e um verde, do pacote de 1996.

```
def joint(A, B, sep=''):
    """A probabilidade conjunta de duas distribuições de probabilidade
       independentes.
    Resultado é todas as entradas da forma {a+sep+b: P(a)*P(b)}"""
    return ProbDist({a + sep + b: A[a] * B[b]
                     for a in A
                     for b in B})

MM = joint(bag94, bag96, ' ')
```

O resultado, o valor de `MM`, é:

```
{
    'brown blue': 0.07199999999999998,
    'brown brown': 0.03899999999999999,
    'brown green': 0.059999999999999984,
    'brown orange': 0.04799999999999999,
    'brown red': 0.03899999999999999,
    'brown yellow': 0.041999999999999996,
    'green blue': 0.023999999999999994,
    'green brown': 0.012999999999999998,
    'green green': 0.02,
    'green orange': 0.015999999999999997,
    'green red': 0.012999999999999998,
    'green yellow': 0.013999999999999999,
    'orange blue': 0.023999999999999994,
    'orange brown': 0.012999999999999998,
    'orange green': 0.02,
    'orange orange': 0.015999999999999997,
    'orange red': 0.012999999999999998,
    'orange yellow': 0.013999999999999999,
    'red blue': 0.04799999999999999,
    'red brown': 0.025999999999999995,
    'red green': 0.04,
    'red orange': 0.031999999999999994,
    'red red': 0.025999999999999995,
    'red yellow': 0.027999999999999997,
    'tan blue': 0.023999999999999994,
    'tan brown': 0.012999999999999998,
    'tan green': 0.02,
    'tan orange': 0.015999999999999997,
    'tan red': 0.012999999999999998,
    'tan yellow': 0.013999999999999999,
    'yellow blue': 0.04799999999999999,
```

```
    'yellow brown': 0.025999999999999995,
    'yellow green': 0.04,
    'yellow orange': 0.031999999999999994,
    'yellow red': 0.025999999999999995,
    'yellow yellow': 0.027999999999999997
}
```

A seguir definimos o predicado que trata do evento "um é amarelho e o outro é verde":

```
def yellow_and_green(r):
    return 'yellow' in r and 'green' in r
```

O resultado de `tal_que(yellow_and_green, MM)` é:

```
{
    'green yellow': 0.25925925925925924,
    'yellow green': 0.7407407407407408
}
```

Agora podemos responder a pergunta: dado que tivemos amarelo e verde (mas não sabemos sabemos qual vem de qual pacote), qual é a probabilidade de que o amarelo tenha vido do pacote de 1994?

```
def yellow94(r):
    return r.startswith('yellow')
```

O resultado de `P(yellow94, tal_que(yellow_and_green, MM))` é, aproximadamente, `0.74`. Então, há 74% de chance de que o amarelo tenha vindo do pacote de 1994.

A forma de resolver o problema foi semelhante ao que já vínhamos fazendo: criar um espaço amostral e usar `P()` para escolher a probabilidade do evento em questão, dado que sabemos sobre o retorno.

Poderíamos usar o *Teorema de Bayes*, mas por que? Porque queremos saber a probabilidade de um evento dada uma evidência, que não está imediatamente disponível; entretanto, a probabilidade da evidência dado o evento está disponível.

Antes de ver as cores dos M&Ms, há duas hipóteses, A e B, ambas com igual probabilidade:

- **A**: primeiro M&M do pacote de 1994, segundo do pacote de 1996
- **B**: primeiro M&M do pacote de 1996, segundo do pacote de 1994

$$P(A) = P(B) = 0.5 \tag{6.5}$$

Então, temos uma evidência:

- **E**: primeiro M&M amarelo, depois verde

Queremos saber a probabilidade da hipótese **A**, dada a evidência **E**: $P(A \mid E)$.

Isso não é fácil de calcular (exceto numerando o espaço amostral), mas o Teorema de Bayes diz:

$$P(A \mid E) = \frac{P(E \mid A) \times P(A)}{P(E)} \tag{6.6}$$

As quantidades do lado direito são mais fáceis de calcular:

$$P(E \mid A) = P(Yellow94) \times P(Green96) \qquad = 0.20 \times 0.20 = 0.04 \tag{6.7}$$

$$P(E \mid B) = P(Yellow96) \times P(Green94) \qquad = 0.10 \times 0.14 = 0.014 \tag{6.8}$$

$$P(A) = 0.5 \tag{6.9}$$

$$P(B) = 0.5 \tag{6.10}$$

$$P(E) = P(E \mid A) \times P(A) + P(E \mid B) \times P(B) \tag{6.11}$$

$$= 0.04 \times 0.5 + 0.014 \times 0.5 = 0.027 \tag{6.12}$$

O resultado final:

$$P(A \mid E) = \frac{P(E \mid A) \times P(A)}{P(E)} \tag{6.13}$$

$$= \frac{0.4 \times 0.5}{0.027} \tag{6.14}$$

$$= 0.7407407407 \tag{6.15}$$

Então é isso. Você tem uma escolha: O Teorema de Bayes permite fazer menos cálculos, mas usa mais álgebra; é melhor custo-benefício se você estiver trabalhando com lápis e papel. Por outro lado, enumerar o espaço amostrar usa menos álgebra, pelo custo de requerer mais cálculos; é melhor custo-benefício se você estiver usando um computador. Idependentemente da abordagem utilizada, é importante conhecer o Teorema de Bayes e como ele funciona.

Mais importante ainda: você comeria M&Ms de 20 anos?

Capítulo 7

Bayes, Bayes, Baby

O capítulo anterior apresentou conceitos fundamentais de Probabilidade e uma das principais ferramentas, o *Teorema de Bayes*. Este capítulo contribui com o anterior, trazendo uma leitura e uma aplicação prática.

7.1 Dados

Considere um conjunto de dados com informações sobre pessoas (sexo, idade e renda familiar):

```
sexo;idade;rendafamiliar
F;20;2000
M;19;3000
M;19;3000
M;20;2500
M;21;3000
```

O conjunto de dados está no formato CSV. Cada linha do conjunto de dados, com exceção da primeira, que representa o cabeçalho, representa um indivíduo (uma pessoa) no conjunto de dados. A partir desse conjunto de dados, vamos fazer algumas análises.

Imagine que uma nova pessoa respondeu a pesquisa, mas não se sabe nada dela. Qual a probabilidade de essa nova pessoa ser do sexo masculino? Para resolver essa questão, vamos utilizar as ferramentas anteriores. A primeira delas é a distribuição de probabilidade, enumerando o conjunto de dados e a quantidade de ocorrências de cada sexo.

```
PDSexo = ProbDist(
    Sexo_M=4,
    Sexo_F=1
)
```

O valor de `PDSexo` é um dicionário com duas chaves, `Sexo_F` e `Sexo_M`, e seus valores indicam que:

- frequência do sexo feminino (`Sexo_F`): $1/5 = 0.2$
- probabilidade de ser do sexo masculino (`Sexo_M`): $4/5 = 0.8$

Na sequência, definimos dois predicados:

```
1  def sexo_m(r):
2      return 'Sexo_M' in r
3
4  def sexo_f(r):
5      return 'Sexo_F' in r
```

Os predicados representados pelas funções `sexo_m()` e `sexo_f()` são aplicados ao dicionário `PDSexo` e retornam `True` se a chave correspondente estiver presente no dicionário.

Assim, podemos calcular a probabilidade de um novo indivíduo ser do sexo masculino usando a função `P()`, o predicado `sexo_m()` e `PDSexo`:

```
1  P(sexo_m, PDSexo)
```

O resultado é: `0.8`.

O que $0,8$ (ou 80%) quer dizer? Chamamos isso de **probabilidade independente**, pois estamos considerando que a **variável** "sexo" não depende de outra variável do conjunto de dados. Variável? Isso, esse é o termo usado, mas, para simplificar, é uma coluna no conjunto de dados.

Outra forma de enxergar isso é perceber que a frequência (aqui diretamente relacionada à probabilidade) é $\frac{4}{5} = 0.8$:

$$A = \text{conjunto das pessoas do sexo masculino} \tag{7.1}$$

$$S = \text{conjunto das pessoas} \tag{7.2}$$

$$P(Sexo = M) = P(A) = \frac{|\ A\ |}{|\ S\ |} = \frac{4}{5} = 0.8 \tag{7.3}$$

Perceba que se fôssemos fazer uma manchete disso, ficaria mais interessante algo como:

> **8 em cada 10 pessoas são homens**

Ao invés de

> **4 de 5 pessoas são homens**

Claro, a relação se mantém. Verifique.

Imagine que você continuou olhando para os dados e ficou curioso. De repente, vem a pergunta: **e como fazer para a idade?**. Talvez sua pergunta venha do fato de que tentou imaginar que seria pouco produtivo gerar a probabilidade de cada idade possível. Considerando idades entre 20 e 59 anos, por exemplo, não seria difícil gerar as probabilidades individuais para cada idade, mas isso não seria prático.

Nesses casos, a técnica é usar uma **categorização** da variável "idade". Assim:

- até 20 anos: categoria A
- 21 a 30 anos: categoria B
- 31 a 50 anos: categoria C
- 51 anos ou mais: categoria D

Isso começa a deixar a situação mais interessante, pois podemos encontrar as quantidades das idades conforme as categorias e gerar a distribuição de probabilidade:

```
PDIdades = ProbDist(
    Idade_A=4,
    Idade_B=1,
    Idade_C=0,
    Idade_D=0
)
```

O valor de `PDIdades` significa que:

- probabilidade de a idade estar na categoria A: $4/5 = 0.8$
- probabilidade de a idade estar na categoria B: $1/5 = 0.2$
- probabilidade de a idade estar na categoria C: $0/5 = 0.0$
- probabilidade de a idade estar na categoria D: $0/5 = 0.0$

Alguns predicados:

```
def idade_A(r):
    return 'Idade_A' in r

def idade_B(r):
    return 'Idade_B' in r

def idade_C(r):
    return 'Idade_C' in r

def idade_D(r):
    return 'Idade_D' in r
```

Com estas ferramentas podemos responder a pergunta: qual a probabilidade de uma pessoa estar na categoria de idade B (faixa de 21 a 30 anos):

```
P(idade_B, PDIdades)
```

O resultado é 0.2.

O raciocínio é o mesmo se usarmos a contagem:

$$A = \text{conjunto das pessoas com idade entre 21 e 30 anos (faixa B)} \tag{7.4}$$

$$S = \text{conjunto de todas as pessoas} \tag{7.5}$$

$$P(Idade = B) = P(A) = \frac{|\,A\,|}{|\,S\,|} = \frac{1}{5} = 0.2 \tag{7.6}$$

Ou seja, 2 em cada 10 pessoas estão na faixa de idade B (21 a 30 anos).

Suponha que alguém tenha se questionado como responder a pergunta: dado que uma pessoa está na faixa etária B, qual a probabilidade de ser homem? Para responder isso, criamos uma **distribuição de probabilidade conjunta**:

```
PDSexoIdade = joint(PDSexo, PDIdades, ' ')
```

O valor de `PDSexoIdade` é:

```
{
    'Sexo_F Idade_A': 0.16,
    'Sexo_F Idade_B': 0.04,
    'Sexo_F Idade_C': 0.0,
    'Sexo_F Idade_D': 0.0,
    'Sexo_M Idade_A': 0.64,
    'Sexo_M Idade_B': 0.16,
    'Sexo_M Idade_C': 0.0,
    'Sexo_M Idade_D': 0.0
 }
```

Daí respondemos a pergunta calculando:

```
P(sexo_m, tal_que(idade_B, PDSexoIdade))
```

O resultado é, aproximadamente, 0.8; então a resposta para a pergunta anterior é 80%.

Outra informação importante é que a distribuição de probabilidade conjunta (resultado da função `joint()`) representa a distribuição de probabilidade de dois eventos ocorrerem sequencialmente, por isso usamos `joint(PDSexo, PDIdades)`.

> O que seria `joint(PDIdades, PDSexo)`? Por que os valores são iguais?

Com base nessas informações podemos interpretar o resultado da seguinte forma:

```
PA = P(sexo_m, PDSexoIdade)
PE = P(idade_B, PDSexoIdade)
prob = PA * PE / PE
```

O valor de `prob` é `0.8000000000000002` ou, aproximadamente, 0.8.

Perceba que há um padrão nesse tipo de pergunta: "dado que" é chamado de **evidência** e representa a informação dada no enunciado (ex.: "dado que uma pessoa está na faixa etária B").

Em outras palavras, essa técnica para encontrar a probabilidade de um evento dada a ocorrência [anterior] de uma evidência é chamada **probabilidade condicional** ou **Teorema de Bayes**.

Usando o Teorema de Bayes na sua forma clássica temos:

```
# Evidência
PE = P(idade_B, PDSexoIdade)
print('P(E) = P(Idade=B) = %.1f%%' % (PE * 100))

# Hipótese A (Sexo = F)
PA = P(sexo_f, PDSexoIdade)
print('P(A) = P(Sexo=F) = %.1f%%' % (PA * 100))

# Hipótese B (Sexo = M)
PB = P(sexo_m, PDSexoIdade)
print('P(B) = P(Sexo=M) = %.1f%%' % (PB * 100))

# Evidência, dada Hipótese A
PEA = P(idade_B, tal_que(sexo_f, PDSexoIdade))
print('P(E|A) = P(Idade=B|Sexo=F) = %.1f%%' % (PEA * 100))

# Evidência, dada Hipótese B
PEB = P(idade_B, tal_que(sexo_m, PDSexoIdade))
print('P(E|B) = P(Idade=B|Sexo=M) = %.1f%%' % (PEB * 100))

# Outra forma de encontrar P(E)
PE2 = PEA * PA + PEB * PB
print('P(E) = P(Idade=B) = %.1f%%' % (PE2 * 100))

# probabilidade desejada (Sexo = M), dada a Evidência -> P(B|E)
PBE = P(sexo_m, tal_que(idade_B, PDSexoIdade))
print('P(B|E) = P(Sexo=M|Idade=B) = %.1f%%' % (PBE * 100))

# outra forma de encontrar P(B|E)
PBE2 = PEB * PB / PE
print('P(B|E) = P(Sexo=M|Idade=B) = %.1f%%' % (PBE2 * 100))

# outra forma de encontrar P(B|E)
PBE3 = PB * PE / PE
print('P(B|E) = P(Sexo=M|Idade=B) = %.1f%%' % (PBE3 * 100))
```

Os resultados seriam:

```
P(E) = P(Idade=B) = 20.0%
P(A) = P(Sexo=F) = 20.0%
```

```
P(B) = P(Sexo=M) = 80.0%
P(E|A) = P(Idade=B|Sexo=F) = 20.0%
P(E|B) = P(Idade=B|Sexo=M) = 20.0%
P(E) = P(Idade=B) = 20.0%
P(B|E) = P(Sexo=M|Idade=B) = 80.0%
P(B|E) = P(Sexo=M|Idade=B) = 80.0%
P(B|E) = P(Sexo=M|Idade=B) = 80.0%
```

Exercícios

- dado que uma pessoa é do sexo feminino, qual a probabilidade da sua renda estar na faixa B? (considere as faixas: A (até 1000), B (1001 a 3000), C (3001 a 5000) e D (acima de 5000)
- dado que uma pessoa está na faixa etária B, qual a probabilidade da sua renda estar na faixa A?

Referências

GIT COMMUNITY. **Git**, [s.d.]. Disponível em: <https://git-scm.com/>. Acesso em: 22 jul. 2018

MENEZES, P. B. **Matemática discreta para computação e informática**. Traducao. 3. ed. Porto Alegre: Bookman, 2010.

MICROSOFT. **Visual Studio Code - Code Editing. Redefined**, [s.d.]. Disponível em: <https://code.visualstudio.com/>. Acesso em: 22 jul. 2018a

MICROSOFT. **TypeScript - JavaScript that scales**, [s.d.]. Disponível em: <https://www.typescriptlang.org/index.html>. Acesso em: 24 jul. 2018b

NODE.JS FOUNDATION. **Node.js**, [s.d.]. Disponível em: <https://nodejs.org>. Acesso em: 23 jul. 2018

NPM, INC. **npm**, [s.d.]. Disponível em: <https://www.npmjs.com/>. Acesso em: 23 jul. 2018

THE JQUERY FOUNDATION. **jQuery**, [s.d.]. Disponível em: <http://jquery.com/>. Acesso em: 22 jul. 2018

W3SCHOOLS. **JavaScript Tutorial**, [s.d.]. Disponível em: <https://www.w3schools.com/js/default.asp>. Acesso em: 22 jul. 2018a

W3SCHOOLS. **JavaScript and HTML DOM Reference**, [s.d.]. Disponível em: <https://www.w3schools.com/jsref/default.asp>. Acesso em: 22 jul. 2018b

W3SCHOOLS. **HTML5 Tutorial**, [s.d.]. Disponível em: <https://www.w3schools.com/html/default.asp>. Acesso em: 22 jul. 2018c

W3SCHOOLS. **CSS Tutorial**, [s.d.]. Disponível em: <https://www.w3schools.com/css/default.asp>. Acesso em: 22 jul. 2018d

W3SCHOOLS. **CSS Reference**, [s.d.]. Disponível em: <https://www.w3schools.com/cssref/default.asp>. Acesso em: 22 jul. 2018e

W3SCHOOLS. **HTML Element Reference**, [s.d.]. Disponível em: <https://www.w3schools.com/tags/default.asp>. Acesso em: 22 jul. 2018f