UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Natan Gonçalves de Lyra

Álgebra computacional com ênfase em Grupo Simétrico usando o GAP

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Natan Gonçalves de Lyra

Álgebra computacional com ênfase em Grupo Simétrico usando o GAP

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Profa. Dra. Dylene Agda Souza de Barros

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2025

Natan Gonçalves de Lyra

Álgebra computacional com ênfase em Grupo Simétrico usando o GAP

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 31 de outubro de 2025:

Profa. Dra. Dylene Agda S. de Barros
IME UFU

Prof. Dr. Rodrigo Sanches Miani
FACOM UFU

Profa. Dra. Rosemary Miguel Pires
ICEX UFF

Uberlândia, Brasil
2025

Resumo

Este trabalho tem como objetivo investigar e aplicar conceitos de Álgebra Computacional com ênfase na teoria de grupos simétricos, utilizando o sistema algébrico computacional GAP (Groups, Algorithms, Programming). Por meio dessa ferramenta, estudam-se estruturas algébricas e algoritmos capazes de representar e manipular grupos de permutação, com foco na modelagem e resolução de problemas de natureza simétrica, como o Cubo Mágico. Inicialmente, são apresentados os fundamentos teóricos da teoria de grupos, incluindo definições, propriedades, e resultados essenciais sobre subgrupos, homomorfismos, ações e blocos. Em seguida, são explorados aspectos computacionais do GAP e de alguns de seus algoritmos internos. A partir disso, é proposta uma modelagem do Cubo Mágico em termos de grupos de permutação, evidenciando como operações algébricas podem descrever as possíveis configurações do cubo e suas transformações. Conclui-se que o uso de ferramentas computacionais como GAP unidas de fundamentos matemáticos resultam no desenvolvimento de algoritmos inteligentes.

Palavras-chave: Álgebra Computacional, Teoria dos Grupos, Grupos Simétricos, GAP, Cubo Mágico.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

GAP	Groups, Algorithms, Programming
BSGS	Base and Strong Generating Set

Sumário

1 Introdução

A Álgebra Computacional é uma área da matemática que integra conceitos da álgebra e da computação para realizar a manipulação simbólica de expressões matemáticas, desempenhando um papel fundamental nas áreas de matemática aplicada, ciências da computação e engenharia, possibilitando a resolução de problemas complexos que envolvem estruturas algébricas, como os grupos de permutação. Esses grupos, também chamados de grupos simétricos, são estruturas algébricas fundamentais na teoria dos grupos, representando as simetrias de um conjunto finito, o grupo simétrico S_n consiste em todas as permutações possíveis de n elementos, totalizando $n!$ permutações. O estudo das propriedades, regras e teoremas dessas estruturas, aliado à sua aplicação na computação, pode levar a soluções mais otimizadas e eficientes para problemas complexos.

O estudo de estruturas algébricas como os grupos simétricos é bem relevante para otimizar e desenvolver algoritmos de sistemas ligados à criptografia, como a criptografia de curvas elípticas, e para simplificar a análise e solução de problemas de decidibilidade de autômatos na teoria de computação. Nesse contexto, estruturas algébricas estão presentes em diversas áreas da computação, e seu estudo contribui para aprimorar o funcionamento de algoritmos fundamentais para diversos sistemas.

Nos últimos 40 anos foram publicados diversos trabalhos e artigos sobre teoria de grupos em especial a de grupos simétricos, no artigo de Seress (SERESS, 1997) ele diz que grupos de permutação é a área da teoria de grupos computacional onde a análise da complexidade dos algoritmos está mais avançada. Além disso, Seress publicou um livro (SERESS, 2003) com mais conceitos e algortimos fortemente ligados a grupos de permutação. Um trabalho mais recente publicado por Romero (ROMERO, 2018) também utiliza de conceitos da teoria de grupos computacional e do algoritmo SCHREIER-SIMS para otimizar e desenvolver algoritmos que trabalhem com homomorfismo de grupos e encontrem elementos de um grupo mediante pesquisas *backtrack*.

Os objetivos principais deste trabalho englobam o estudo de estruturas algébricas, em especial a de grupos simétricos, para otimizar e desenvolver algoritmos que tenham a capacidade de resolver problemas com características simétricas.

Durante o desenvolvimento desse trabalho, foram utilizadas ferramentas disponíveis no *GAP* (The GAP Group, 2024b), que é um sistema *open source* com linguagem própria de álgebra discreta computacional com ênfase particular na teoria computacional de grupos, que providencia uma biblioteca com milhares de funções que implementam algoritmos algébricos escritos na linguagem do *GAP*.

Esse trabalho foi realizado seguindo a seguinte metodologia: a primeira e segunda

etapas do projeto consistuiram na condução de uma pesquisa abrangente de conceitos matemáticos e algorítmicos, respectivamente. Estas etapas são essenciais para estabelecer uma base teórica sólida e para identificar trabalhos relacionados que servirão como referência. Através dessa investigação, foram reunidos conceitos matemáticos, funções e algoritmos que podem ser implementados no *GAP* que foram cruciais para o desenvolvimento de soluções mais otimizadas e eficientes para problemas complexos que envolvem álgebra computacional e teoria de grupos computacionais.

Dentro da fase de pesquisa da fundamentação matemática, foram conduzidos diversos estudos específicos para destacar os principais tópicos relacionados à teoria de grupos computacionais com ênfase em grupos simétricos, tais como definições e principais formas de representação de grupos simétricos, para melhor compreensão do leitor no decorrer do trabalho.

Na segunda parte do projeto, além da fundamentação matemática, foram realizados estudos de funções e algoritmos do *GAP* que utilizem dos conceitos estudados no desenvolvimento da fundamentação matemática anterior.

A parte final do projeto envolveu a implementação prática daquilo que foi concebido nas fases iniciais da monografia. Com base nos conceitos matemáticos e algoritmos do *GAP* estudados anteriormente, foi possível desenvolver soluções mais otimizadas e eficientes para problemas com cunho matemático que envolvem álgebra computacional e teoria de grupos computacionais, como por exemplo a resolução de um cubo mágico.

2 Fundamentação Teórica

Neste capítulo, será apresentado os principais conceitos matemáticos para a modelagem algébrica e análise do Cubo Mágico. Serão introduzidas as bases da teoria dos grupos, homomorfismos, grupos cíclicos e grupos de permutação, que permitirão representar os movimentos do cubo. Além disso, abordaremos ações de grupos, órbitas e blocos, que serão ferramentas importantes para analisar como essas permutações atuam sobre as peças do cubo. Ademais, discutiremos a noção de grupo livre, fundamental para descrever sequências de movimentos por meio de geradores associados às faces do cubo. Por fim, serão apresentadas as principais funções do *GAP*, que reúnem todos os conceitos descritos anteriormente, para realizarmos a análise e o desenvolvimento do algoritmo que resolve o cubo.

2.1 Grupos

A teoria de grupos computacionais, que é a utilização de conceitos de estruturas algébricas como grupos no desenvolvimento de novos algoritmos, é uma área de pesquisa bastante ativa, como pode ser visto pelo livro publicado pelo matemático húngaro Ákos Seress (SERESS, 2003), onde desenvolveu algoritmos fortemente ligados a grupos de permutação. Além disso, ao longo da última década, os trabalhos de Alexander Hulpke (HULPKE, 2016) com ênfase especial em grupos de permutação dentro da teoria de grupos computacionais, têm contribuído significativamente para essa área.

A seguir na fundamentação teórica do trabalho, serão repassados definições, proposições e teoremas que serão de grande importância para aprofundar o entendimento dos conceitos e motivações por trás dos métodos de resolução de problemas a serem investigados no decorrer do trabalho.

Grande parte do estudo realizado nesta seção foi retirada de (IEZZI, 2003), que apresenta de forma detalhada os conceitos e resultados matemáticos utilizados. Também utilizamos (HERSTEIN, 1975) e (HUNGERFORD, 1980).

Definição 2.1.1 (Grupo)**.** *Um* ***grupo*** *é um par* $(G, \cdot)$*, onde* G *é um conjunto não vazio e* $" \cdot " : G \times G \to G$*,* $(x, y) \in G \mapsto x \cdot y \in G$*, é uma operação binária que satisfaz as seguintes propriedades:*

1. ***Associatividade****: Para todos* $a, b, c \in G$*, temos* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$*;*
2. ***Elemento neutro****: Existe* $e \in G$ *tal que, para todo* $a \in G$*, vale* $a \cdot e = e \cdot a = a$*;*
3. ***Elemento inverso****: Para cada* $a \in G$*, existe* $a^{-1} \in G$ *tal que* $a \cdot a^{-1} = a^{-1} \cdot a = e$*.*

Exemplo 2.1.1. *O conjunto dos inteiros $\mathbb{Z}$ com a operação de adição forma um grupo. O elemento neutro é 0 e o inverso de $n \in \mathbb{Z}$ é $-n$.*

Exemplo 2.1.2. *O conjunto dos racionais $\mathbb{Q}^* = \mathbb{Q} - \{0\}$ com a operação de multiplicação forma um grupo. O elemento neutro é 1 e o inverso de $r \in \mathbb{Q}^*$ é $\frac{1}{r}$.*

Exemplo 2.1.3. *Sejam Ω um conjunto não vazio e $Sym(\Omega) = \{f : \Omega \to \Omega; f$ é uma função bijetora$\}$. Temos que $Sym(\Omega)$ é um grupo com a operação* composição de funções. *De fato, $(f \circ g) \circ h = f \circ (g \circ h)$ para todas as funções f, g, $h \in Sym(\Omega)$, a função $x \in \Omega \longmapsto x \in \Omega$ é elemento neutro e, se f é bijetora, então existe sua função inversa f^{-1}, que também é bijetora. Se Ω é um conjunto finito com n elementos, $Sym(\Omega)$ é um grupo finito com $n!$ elementos.*

Exemplo 2.1.4. *Como caso particular do Exemplo 2.1.3, consideramos $\Omega = \{1, 2, 3\}$ e denotaremos $Sym(\Omega) = S_3$. O grupo S_3 é um grupo de permutações com 6 elementos.*

Um grupo $(G, \cdot)$ também possui as seguintes propriedades:

- Unicidade do elemento neutro $e \in G$;
- Unicidade do inverso a^{-1}, $\forall a \in G$;
- $e^{-1} = e$;
- $(a^{-1})^{-1} = a$, $\forall a \in G$;
- $(a \cdot b)^{-1} = b^{-1} \cdot a^{-1}$;
- Todo elemento de G é regular para a operação $\cdot$, ou seja, para quaisquer $a, x, y \in G$, se $a \cdot x = a \cdot y \quad$ ou $\quad x \cdot a = y \cdot a$, então $x = y$.

Para simplificar a notação, escreveremos apenas G para um grupo $(G, \cdot)$ e ab para $a \cdot b$, com $a, b \in G$.

2.2 Subgrupos

Para ilustrar a ideia de subgrupo, consideremos o grupo $(\mathbb{R}, +)$. O conjunto dos inteiros $\mathbb{Z}$ é um subconjunto de $\mathbb{R}$ que satisfaz duas propriedades essenciais:

1. $\mathbb{Z}$ é fechado em relação à adição, isto é, a soma de quaisquer dois números inteiros ainda é um número inteiro;
2. $(\mathbb{Z}, +)$, com a operação de adição induzida por $(\mathbb{R}, +)$, também é um grupo, como vimos no Exemplo 2.1.1.

Por isso, dizemos que $(\mathbb{Z},+)$ é um *subgrupo* de $(\mathbb{R},+)$. De maneira similar, o conjunto dos números racionais $(\mathbb{Q},+)$ também é um subgrupo de $(\mathbb{R},+)$.

Definição 2.2.1. *Seja $(G,\cdot)$ um grupo. Um subconjunto não vazio $H \subseteq G$ é um* ***subgrupo*** *de G se, com a operação induzida por G, $(H,\cdot)$ também é um grupo.*

Se e é o elemento neutro de G, então $\{e\}$ e o próprio G são subgrupos de G. Esses são chamados de *subgrupos triviais*.

Proposição 2.2.1. *Seja $(G,\cdot)$ um grupo. Um subconjunto não vazio $H \subseteq G$ é um subgrupo de G se, e somente se, para quaisquer $a, b \in H$, o elemento ab^{-1} também pertence a H, onde b^{-1} é o inverso de b em G.*

Demonstração. Suponha que H seja um subgrupo de G. Seja e o elemento neutro de G e e_H o elemento neutro de H. Como todo elemento é regular em relação a $\cdot$, segue que $e = e_H$. Para $b \in H$, sejam b^{-1} e b_H^{-1} os inversos de b em G e H, respectivamente. Como $b \cdot b^{-1} = e = b \cdot b_H^{-1}$, temos $b^{-1} = b_H^{-1}$. Finalmente, para $a, b \in H$, como $(H,\cdot)$ é um grupo, $a \cdot b \in H$ e $a \cdot b^{-1} \in H$.
Reciprocamente, suponha que $ab^{-1} \in H$, para todos $a, b \in H$. Como H não é vazio, existe $x_0 \in H$. Então, $e = x_0 x_0^{-1} \in H$ e $x_0^{-1} = e x_0^{-1} \in H$, logo H possui elemento neutro e todo elemento de H possui inverso em H. Se $a, b \in H$, então $b^{-1} \in H$ e assim $ab = a(b^{-1})^{-1} \in H$, o que implica que H é fechado para operação de G. Associatividade em H segue diretamente da de G, pois $a(bc) = (ab)c$ para todos $a, b, c \in G$. □

Exemplo 2.2.1. *Considere o conjunto*

$$H = \{x \in \mathbb{R}^* \mid x > 0\},$$

isto é, o conjunto dos números reais estritamente positivos. Verifiquemos que H é um subgrupo de $(\mathbb{R}^*,\cdot)$, o grupo multiplicativo dos números reais diferentes de zero.

Sejam $a, b \in H$, então $a > 0$ e $b > 0$. Como $b > 0$, temos que $b^{-1} > 0$, pois o inverso multiplicativo de um número positivo também é positivo. Consequentemente,

$$ab^{-1} > 0,$$

ou seja, $a \cdot b^{-1} \in H$. Portanto, H é fechado para a operação considerada e, assim, constitui um subgrupo de $(\mathbb{R}^*,\cdot)$.

Sejam G um grupo e H um subgrupo de G. Para $a, b \in G$, defina $a \sim b$ se $a^{-1}b \in H$. É sabido que "$\sim$" é uma relação de equivalência.

Proposição 2.2.2. *Nessas condições, se $a \in G$, a classe de equivalência de a é o conjunto*

$$aH = \{ah \mid h \in H\}.$$

Demonstração. Seja

$$[a] = \{x \in G \mid x \sim a\}$$

a classe de equivalência de a. Vamos mostrar que $[a] = aH$. Se $x \in [a]$, tem-se $a^{-1}x = h \in H$, e portanto $x = ah \in aH$. Logo $[a] \subset aH$.
Reciprocamente, se $x = ah \in aH$ $h \in H$, temos $a^{-1}x = h \in H$ e assim $a \sim x$. Logo $x \in [a]$ e, assim, $aH \subseteq [a]$. Portanto $[a] = aH$. □

Definição 2.2.2. *Seja H um subgrupo de um grupo G. Para um elemento $a \in G$, denomina-se* classe lateral à esquerda de a módulo H *o conjunto*

$$aH = \{ah \mid h \in H\}.$$

Analogamente, a classe lateral à direita de a módulo H é o conjunto

$$Ha = \{ha \mid h \in H\}.$$

Definição 2.2.3. *Seja G um grupo. Um subgrupo N de G é dito um subgrupo normal de G se, para todo $g \in G$ e $n \in N$, tem-se*

$$gng^{-1} \in N.$$

Lema 2.2.1. *Um subgrupo N de G é normal em G se, e somente se, $gNg^{-1} = N$ para todo $g \in G$.*

Demonstração. Se $gNg^{-1} = N$ para todo $g \in G$, certamente $gNg^{-1} \subset N$, de modo que N é normal em G. Reciprocamente, suponha que N é normal em G. Se $g \in G$, tem-se $gNg^{-1} \subset N$ e

$$g^{-1}Ng = g^{-1}N(g^{-1})^{-1} \subset N.$$

Como $g^{-1}Ng \subset N$, então

$$N = g(g^{-1}Ng)g^{-1} \subset gNg^{-1} \subset N,$$

donde $N = gNg^{-1}$. □

Lema 2.2.2. *Um subgrupo N de G é normal em G se, e somente se, toda classe lateral à esquerda de N em G é também uma classe lateral à direita de N em G.*

Demonstração. A demonstração deste lema encontra-se em (HERSTEIN, 1975). □

Lema 2.2.3. *Um subgrupo N de G é normal em G se, e somente se, o produto de duas classes laterais à esquerda de N em G é novamente uma classe lateral à esquerda de N em G.*

Demonstração. Suponha que N seja um subgrupo normal de G e sejam aN, bN duas classes laterais. Então:

$$(aN)(bN) = a(Nb)N = a(bN)N = (ab)NN = abN.$$

Logo, o produto $(aN)(bN)$ é uma classe lateral de N. Reciprocamente, suponha que $(aN)(bN) = xN$. Como $ab = ae \cdot be \in (aN)(bN) = xN$, temos que $xN = abN$ e $(aN)(bN) = abN$, para todos $a, b \in G$.

Fazendo $a = e$, obtemos $NbN = bN$, e assim $Nb = bN$ para todo $b \in G$. Logo, N é um subgrupo normal de G. □

Definição 2.2.4. *Seja G um grupo e N um subgrupo normal de G. Chamamos de* grupo quociente *o conjunto das classes laterais de N em G, denotado por*

$$G/N = \{\, aN \mid a \in G \,\}.$$

A operação em G/N é definida por

$$(aN)(bN) = (ab)N, \quad \textit{para todos } a, b \in G.$$

Esse conjunto, munido dessa operação, forma um grupo, cuja identidade é $N = eN$ e o inverso de aN é $a^{-1}N$.

2.3 Homomorfismos de Grupos

A noção de homomorfismo surge como uma forma de relacionar diferentes grupos. A ideia central é que a estrutura algébrica de um grupo G possa ser transportada para outro grupo H por meio de uma aplicação que respeite a operação dos grupos. Em outras palavras, não é a igualdade de elementos que importa, mas sim a compatibilidade entre as operações.

Definição 2.3.1. *Sejam $(G, *)$ e $(H, \cdot)$. Uma aplicação*

$$f : G \longrightarrow H$$

é um ***homomorfismo de grupos*** *se, para quaisquer elementos $x, y \in G$, vale*

$$f(x * y) = f(x) \cdot f(y). \tag{1}$$

Para facilitar a notação, escreveremos (1) como $f(xy) = f(x)f(y)$ sem fazer diferenciação entre as operações dos grupos G e H.

Definição 2.3.2. *Um homomorfismo de grupos $f : G \longrightarrow H$ bijetor é chamado isomorfismo de grupos.*

Exemplo 2.3.1. *Considere a aplicação*

$$f : \mathbb{Z} \longrightarrow \mathbb{Z}_n, \quad f(k) = \overline{k},$$

onde $\overline{k}$ *denota a classe de equivalência de* k *módulo* n*.*

Para quaisquer $a, b \in \mathbb{Z}$*, temos*

$$f(a+b) = \overline{a+b} = \overline{a} + \overline{b} = f(a) + f(b).$$

Portanto, f *é um homomorfismo de grupos de* $(\mathbb{Z}, +)$ *em* $(\mathbb{Z}_n, +)$*.*

Exemplo 2.3.2. *Para* $a \in \mathbb{Z}$ *um número fixo, definimos a aplicação*

$$f : \mathbb{Z} \longrightarrow \mathbb{Z}, \quad f(m) = a \cdot m.$$

Para quaisquer $m, n \in \mathbb{Z}$*, temos*

$$f(m+n) = a(m+n) = am + an = f(m) + f(n),$$

logo, f *é um homorfismo de grupos.*

Proposição 2.3.1. *Sejam* G *e* J *grupos multiplicativos, com elementos neutros denotados por* e *e* u*, respectivamente. Considere ainda um homomorfismo de grupos* $f : G \to J$*. Então,* $f(e) = u$*.*

Demonstração. Observe que $f(e)f(e) = f(ee) = f(e) = uf(e)$. Assim, segue $f(e) = u$.

□

Proposição 2.3.2. *Sejam* $a \in G$ *e* $f : G \longrightarrow H$ *um homomorfismo de grupos. Então,*

$$f(a^{-1}) = f(a)^{-1}.$$

Demonstração. De fato, $f(a)f(a^{-1}) = f(aa^{-1}) = f(e) = u$. Logo, $f(a^{-1}) = f(a)^{-1}$.

□

Definição 2.3.3. *Seja* $f : G \to H$ *um homomorfismo de grupos. Definimos núcleo de* f *o subconjunto*

$$\ker(f) = \{\, x \in G \mid f(x) = e_H \,\}.$$

Proposição 2.3.3. *Seja* H *um subgrupo de* G *e* $f : G \to J$ *um homomorfismo de grupos. Então a imagem de* H *através de* f*, definida por*

$$f(H) = \{f(x) \mid x \in H\},$$

é um subgrupo de J*.*

Demonstração. Primeiro, como H é um subgrupo de G, temos que $e \in H$, onde e é o elemento neutro de G. Pelo homomorfismo, $f(e) = u$, que é o neutro de J. Portanto, $u \in f(H)$, garantindo que $f(H)$ contém o elemento neutro.

Agora, seja $c, d \in f(H)$. Então existem $a, b \in H$ tais que $c = f(a)$ e $d = f(b)$. Consideremos o produto

$$cd^{-1} = f(a)f(b)^{-1}.$$

Pela Proposição 2.3.2, $f(b^{-1}) = f(b)^{-1}$, então

$$cd^{-1} = f(a)f(b^{-1}) = f(ab^{-1}).$$

Como H é um subgrupo, $ab^{-1} \in H$. Portanto,

$$cd^{-1} = f(ab^{-1}) \in f(H),$$

o que mostra que $f(H)$ é um subgrupo de J. □

Observação 2.3.1. *Seja $f : G \to H$ um homomorfismo de grupos com $\ker(f) = K$. Temos que K é um subgrupo de G e, para $g \in G$, consideremos a classe lateral à esquerda g módulo K, gK. Se g_1, $g_2 \in G$ temos que $g_1K = g_2K$ se, e somente se, $f(g_1) = f(g_2)$. De fato, como classes laterais são classes de equivalências, $g_1K = g_2K$ se, e somente se, $g_1 = g_2k$, para algum $k \in K$, donde $f(g_1) = f(g_2k) = f(g_2)f(k) = f(g_2)e_H = f(g_2)$. Portanto, para qualquer $g \in G$, os elementos da classe lateral gK têm a mesma imagem $f(g)$ em H.*

Proposição 2.3.4. *Sejam $f : G \to H$ um homomorfismo de grupos e N um subgrupo normal de G. Definimos a aplicação $\mu : G \to G/N$ por*

$$\mu(a) = aN.$$

Então, μ é um homomorfismo sobrejetor de grupos, cujo núcleo é exatamente N. O homomorfismo μ é chamado homomorfismo canônico de G sobre G/N.

Demonstração. Para $a, b \in G$, temos

$$\mu(ab) = (ab)N = (aN)(bN) = \mu(a)\mu(b),$$

portanto, μ é de fato um homomorfismo.

Seja $y \in G/N$. Então $y = aN$ para algum $a \in G$. Como $\mu(a) = aN = y$, segue que μ é sobrejetora. Finalmente, provemos que $\ker(\mu) = N$.

O elemento identidade em G/N é a classe N. Se $a \in \ker(\mu)$, então $\mu(a) = aN = N$, o que implica $a \in N$. Logo, $\ker(\mu) \subseteq N$. Por outro lado, se $a \in N$, então $aN = N$, e portanto $\mu(a) = N$, ou seja, $a \in \ker(\mu)$. Assim, concluímos que

$$\ker(\mu) = N.$$

□

Proposição 2.3.5. *Seja $f : G \to H$ um homomorfismo de grupos. Então o núcleo de f, é um subgrupo normal de G.*

Demonstração. Temos que $N = \ker(f)$ é um subgrupo de G pela Proposição 2.3.2. Resta verificar que N é normal. Para isso, provemos que para todo $a \in G$, vale a igualdade $aN = Na$.

Se $x \in aN$, então $x = ah$, para algum $h \in N$. Podemos escrever

$$x = ah = (aha^{-1})a.$$

Como $f(h) = e_H$, segue que

$$f(aha^{-1}) = f(a)f(h)f(a^{-1}) = f(a)e_H f(a)^{-1} = e_H.$$

Portanto, $aha^{-1} \in N$, e consequentemente $x \in Na$. Assim, temos $aN \subset Na$. Analogamente, temos que $Na \subset aN$.

Portanto, $aN = Na$ para todo $a \in G$, o que prova que N é um subgrupo normal de G. □

Proposição 2.3.6 (Teorema do Homomorfismo)**.** *Seja $f : G \to H$ um homomorfismo* sobrejetor *de grupos. Se $N = \ker(f)$, então o grupo quociente G/N é isomorfo a H.*

Demonstração. Nosso objetivo é construir um isomorfismo explícito entre G/N e H.

Note que os elementos de G/N são classes laterais da forma aN, com $a \in G$, enquanto os elementos de H são as imagens $f(a)$, para $a \in G$. Isso sugere considerar a aplicação

$$\varphi : G/N \longrightarrow H, \quad \varphi(aN) = f(a).$$

Se $aN = bN$, então $b^{-1}a \in N = \ker(f)$, o que implica $f(b^{-1}a) = e_H$. Portanto,

$$f(b^{-1}a) = f(b^{-1})f(a) = f(b)^{-1}f(a) = e_H, \text{ logo } f(a) = f(b).$$

Assim, a função $\varphi(aN) = f(a)$ é bem definida.

Suponha que $\varphi(aN) = \varphi(bN)$, isto é, $f(a) = f(b)$, com $a, b \in G$. Então

$$e_H = f(b^{-1})f(a) = f(b^{-1}a),$$

o que mostra que $b^{-1}a \in N$ e, portanto, $aN = bN$. Logo, φ é injetora.

Como f é sobrejetor por hipótese, para todo $y \in H$ existe $a \in G$ tal que $y = f(a)$. Nesse caso, se $x = aN \in G/N$, $\varphi(x) = \varphi(aN) = f(a) = y$. Logo, φ é sobrejetora.

Sejam $aN, bN \in G/N$. Então

$$\varphi\Big((aN)(bN)\Big) = \varphi\Big((ab)N\Big) = f(ab) = f(a)f(b) = \varphi(aN)\varphi(bN).$$

Portanto, φ é um homomorfismo de grupos.

Concluímos que $\varphi : G/N \to H$ é um isomorfismo de grupos. □

2.4 Grupos Cíclicos

Definição 2.4.1. *Seja G um grupo multiplicativo e $a \in G$. Seja $n \in \mathbb{Z}$, definimos a^n por recorrência como:*

$$a^n = \begin{cases} e, & se\ n = 0, \\ a^{n-1}\, a, & se\ n > 0, \\ \left(a^{-1}\right)^{-n}, & se\ n < 0. \end{cases}$$

Sabendo que para quaisquer $m, n \in \mathbb{Z}$, vale $a^m a^n = a^{m+n}$, definimos o subconjunto gerado por a como

$$[a] = \{a^m \mid m \in \mathbb{Z}\},$$

isto é, o conjunto de todas as potências inteiras de a. Sabemos que $[a]$ nunca é vazio, pois contém o elemento neutro e de G, já que $e = a^0$.

Proposição 2.4.1. *Seja G um grupo multiplicativo e $a \in G$.*

(i) O subconjunto $[a]$ é um subgrupo de G.

(ii) Se H é um subgrupo de G tal que $a \in H$, então $[a] \subseteq H$.

Demonstração. (i) Sabendo que $[a]$ nunca é vazio já que $e = a^0$. Sejam $u, v \in [a]$, com $u = a^m$ e $v = a^n$ para alguns $m, n \in \mathbb{Z}$. Então

$$uv^{-1} = a^m (a^n)^{-1} = a^m a^{-n} = a^{m-n} \in [a].$$

Portanto, $[a]$ é um subgrupo de G.

(ii) Como H é subgrupo e $a \in H$, podemos concluir que todas as potências inteiras de a também pertencem a H. Logo, $[a] \subseteq H$. □

Definição 2.4.2. *Um grupo multiplicativo G é chamado de **grupo cíclico** se existe algum elemento $a \in G$ tal que*

$$G = [a] = \{a^m \mid m \in \mathbb{Z}\}.$$

*Nesse caso, o elemento a é denominado **gerador** do grupo G.*

Definição 2.4.3. *Seja $n \in \mathbb{Z}$, definimos a notação aditiva de grupos cíclicos como:*

$$na = \begin{cases} 0, & \text{se } n = 0, \\ a + (n-1)a, & \text{se } n > 0, \\ -n(-a), & \text{se } n < 0. \end{cases}$$

Exemplo 2.4.1. *O grupo aditivo* $(\mathbb{Z}, +)$ *é um grupo cíclico. Isso ocorre porque qualquer inteiro pode ser expresso como múltiplo de* 1 *ou de* -1*.*

Temos

$$\mathbb{Z} = \{m \cdot 1 \mid m \in \mathbb{Z}\} = \{m \cdot (-1) \mid m \in \mathbb{Z}\}.$$

Portanto,

$$\mathbb{Z} = [1] = [-1],$$

e os elementos 1 *e* -1 *são os únicos geradores do grupo* $\mathbb{Z}$*.*

Proposição 2.4.2. *Todo subgrupo de um grupo cíclico é cíclico.*

Demonstração. Seja $G = [a]$ um grupo cíclico multiplicativo com elemento neutro e, e seja H um subgrupo de G. Então cada elemento de H pode ser escrito como uma potência de a, ou seja, $x = a^n$ para algum $n \in \mathbb{Z}$.

Se $H = \{e\}$, então H é cíclico, gerado pelo próprio e.

Caso contrário, H contém algum elemento de potência não nula. Como para cada $m \in \mathbb{Z}$, $a^{-m} \in H$ sempre que $a^m \in H$, podemos considerar apenas os expoentes positivos. Seja h o menor inteiro positivo tal que $a^h \in H$. Seja $b = a^h$, será mostrado que $H = [b]$.

Tomemos um elemento genérico $x = a^n \in H$. Pelo algoritmo euclidiano, existem inteiros q e r com $0 \leq r < h$ tais que

$$n = hq + r.$$

Logo,

$$x = a^n = a^{hq+r} = (a^h)^q a^r = b^q a^r.$$

Como $b^q \in H$ e $x \in H$, o elemento $a^r = (b^q)^{-1}x$ também pertence a H. Pela escolha de h como menor expoente positivo em H, devemos ter $r = 0$. Assim,

$$x = b^q \in [b].$$

Portanto, $H \subseteq [b]$. A inclusão oposta é imediata, pois qualquer potência de b pertence a H. Concluímos que

$$H = [b],$$

mostrando que H é cíclico. □

Proposição 2.4.3. *Seja $G = [a]$ um grupo cíclico e suponha que existam inteiros distintos r e s tais que*

$$a^r = a^s.$$

Então existe um inteiro positivo $h > 0$ tal que:

1. $a^h = e$;
2. $a^r \neq e$ *para todo inteiro* r *com* $0 < r < h$.

Neste caso, h é chamado de ***ordem*** *de a, e o grupo pode ser escrito como*

$$G = [a] = \{e, a, a^2, \ldots, a^{h-1}\}.$$

Demonstração. Sem perda de generalidade, suponha $r > s$. Então

$$a^r(a^s)^{-1} = a^{r-s} = e.$$

Portanto, existem potências positivas de a que resultam no elemento neutro e. Seja h o menor inteiro positivo com $a^h = e$.

A partir desse ponto, as potências de a se repetem ciclicamente:

$$a^{h+1} = a^h \cdot a = e \cdot a = a, \quad a^{h+2} = a^h \cdot a^2 = a^2, \ldots$$

Precisamos verificar que não existem repetições antes de h. Suponha, por absurdo, que $a^i = a^j$ com $0 \leq i < j < h$. Então

$$a^{j-i} = a^j(a^i)^{-1} = e,$$

com $0 < j - i < h$, o que contradiz a escolha de h como o menor inteiro positivo para o qual $a^h = e$.

Agora, considere um elemento arbitrário $x \in G$. Por definição, existe um inteiro m tal que $x = a^m$. Pelo algoritmo da divisão, existem inteiros q e r com $0 \leq r < h$ tal que

$$m = hq + r.$$

Assim,

$$x = a^m = a^{hq+r} = (a^h)^q a^r = e^q a^r = a^r,$$

com $0 \leq r < h$. Portanto, qualquer elemento de G é uma das potências

$$e = a^0, a, a^2, \ldots, a^{h-1}.$$

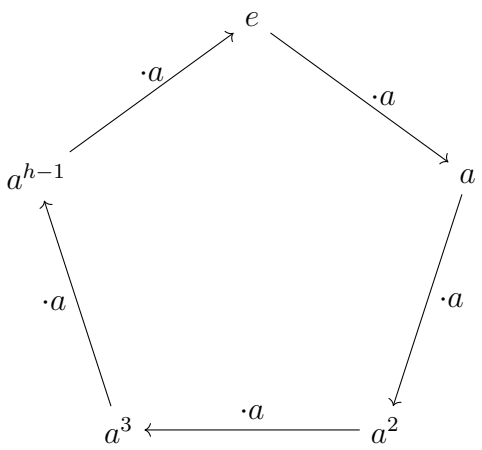
Consequentemente, $G = \{e, a, a^2, \ldots, a^{h-1}\}$ e a ordem do grupo é h. □

Definição 2.4.4. *A **ordem** de um elemento a em um grupo G é o menor inteiro positivo h tal que*

$$a^h = e.$$

Se G é cíclico, a ordem do gerador a coincide com o número de elementos de G.

Exemplo 2.4.2. *Um grupo cíclico de ordem $h = 5$ pode ser observado da seguinte maneira:*



Proposição 2.4.4. *Seja a um elemento de ordem $h > 0$ em um grupo G. Então, para qualquer inteiro m,*

$$a^m = e \quad \textit{se e somente se} \quad h \mid m.$$

Demonstração. Suponha que $a^m = e$. Pelo algoritmo euclidiano, existem inteiros q e r com $0 \leq r < h$ tais que

$$m = hq + r.$$

Então

$$e = a^m = a^{hq+r} = (a^h)^q a^r = e^q a^r = a^r.$$

Como $a^m = e$, segue que $a^r = e$. Pela definição de ordem de a, o menor inteiro positivo h satisfaz $a^h = e$, portanto não pode existir $r > 0$ tal que $a^r = e$. Logo, $r = 0$ e $m = hq$, ou seja, $h \mid m$.

Se $h \mid m$, então existe $q \in \mathbb{Z}$ tal que $m = hq$. Logo,

$$a^m = a^{hq} = (a^h)^q = e^q = e.$$

□

2.5 Grupos Simétricos

Os grupos simétricos, também chamados de grupos de permutação, são estruturas algébricas fundamentais na teoria dos grupos, representando as simetrias de um conjunto finito.

A teoria de grupos simétricos tem origem na segunda metade do século XVIII, com trabalhos de matemáticos como Joseph-Louis Lagrange, que estabeleceu a noção de permutações em um grupo. No entanto, os avanços conceituais mais importantes chegaram em meados do século XIX com Évariste Galois, onde desenvolveu a teoria dos grupos para resolver problemas de solvabilidade de equações polinomiais. Os grupos simétricos surgiram como uma maneira natural de estudar as permutações de raízes de polinômios, permitindo a análise da estrutura dos grupos de permutações, dando espaço posteriormente para o desenvolvimento da teoria de grupos computacionais.

Esse trabalho terá ênfase em resolver problemas que envolvem o uso e manipulação de grupos simétricos, cuja análise será aprofundada no decorrer do trabalho.

Definição 2.5.1. *Sejam $a_1, a_2, \ldots, a_r \in I_n = \{1, 2, \ldots, n\}$ elementos distintos. Dizemos que uma permutação $\sigma \in S_n$ é um* ***ciclo de comprimento*** *r se:*

$$\sigma(a_1) = a_2,\ \sigma(a_2) = a_3,\ \ldots,\ \sigma(a_{r-1}) = a_r,\ \sigma(a_r) = a_1,$$

e, além disso,

$$\sigma(x) = x \quad \text{para todo } x \in I_n \setminus \{a_1, a_2, \ldots, a_r\}.$$

O conjunto $\{a_1, a_2, \ldots, a_r\}$ é chamado de ***suporte*** *do ciclo.*

A notação utilizada para representar essa permutação é:

$$\sigma = (a_1\ a_2\ \ldots\ a_r).$$

No caso particular em que $r = 2$, o ciclo é chamado de ***transposição****.*

Exemplo 2.5.1. *Considere a permutação $\sigma \in S_5$ definida por:*

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 5 & 4 & 1 \end{pmatrix}.$$

Verificamos que:

$$\sigma(1) = 3, \quad \sigma(3) = 5, \quad \sigma(5) = 1,$$

enquanto $\sigma(2) = 2$ e $\sigma(4) = 4$. Portanto, σ é um ciclo de comprimento 3 cujo suporte é $\{1, 3, 5\}$, e pode ser escrito em notação cíclica como:

$$\sigma = (1\ 3\ 5).$$

Proposição 2.5.1. *Seja $\sigma = (a_1\, a_2 \ldots a_r) \in S_n$ um ciclo de comprimento $r > 1$. Então, a ordem de σ é r. Em particular, se e denota a permutação identidade de S_n, então*

$$\langle \sigma \rangle = \{e, \sigma, \sigma^2, \ldots, \sigma^{r-1}\}.$$

Demonstração. Pela definição de ciclo, temos que $\sigma^{i-1}(a_1) = a_i$ para $1 \leq i \leq r$, e $\sigma^r(a_1) = a_1$, enquanto que $\sigma(x) = x$ para todo x fora do suporte $\{a_1, \ldots, a_r\}$. Assim, $\sigma^i(a_1) = a_{i+1}$ para $1 \leq i < r$, de modo que $\sigma^i \neq e$ quando $1 \leq i < r$, dessa forma, $r \leq o(\sigma)$. Além disso, se considerarmos i tal que $1 \leq i \leq r$, logo $\sigma^r(a_i) = \sigma^r(\sigma^{i-1}(a_1)) = \sigma^{i-1}(\sigma^r(a_1)) = \sigma^r(a_1) = a_i$ e, de forma análoga, $\sigma^r(a_i) = a_i$ para todo $1 \leq i \leq r$, além de fixar todos os outros elementos. Logo, $\sigma^r = e$. Portanto, a ordem de σ é r. □

Definição 2.5.2. *Dois ciclos de S_n são chamados disjuntos se seus suportes não possuem elementos em comum.*

Proposição 2.5.2. *Dois ciclos disjuntos comutam, isto é, se σ e μ são ciclos disjuntos, então $\sigma\mu = \mu\sigma$.*

Demonstração. Sejam α e β ciclos disjuntos de S_n, com suportes A e B, respectivamente. Se $x \in A$, então $(\alpha \circ \beta)(x) = \alpha(\beta(x)) = \alpha(x)$, pois β fixa x. Do mesmo modo, $(\beta \circ \alpha)(x) = \beta(\alpha(x)) = \alpha(x)$. Portanto, $(\alpha \circ \beta)(x) = (\beta \circ \alpha)(x)$ em A. Se $x \in B$, a demonstração é análoga. Se $x \notin A \cup B$, temos $\alpha(x) = x$ e $\beta(x) = x$, logo $(\alpha \circ \beta)(x) = (\beta \circ \alpha)(x) = x$. Assim, $\alpha \circ \beta = \beta \circ \alpha$. □

Proposição 2.5.3. *Seja $\sigma \in Sn$, $\sigma \neq e$. Então σ pode ser escrito como produto de ciclos disjuntos. Além disso, a decomposição é única a menor da ordem dos fatores.*

Demonstração. Seja $\sigma \in S_n$ com $\sigma \neq e$. Para iniciar, tomemos o elemento $1 \in I_n = \{1, 2, \ldots, n\}$ e consideremos a sequência obtida aplicando iteradamente σ:

$$1,\ \sigma(1),\ \sigma^2(1),\ \sigma^3(1), \ldots$$

Existe o menor inteiro $r > 0$ tal que $\sigma^r(1) = 1$. Dessa forma, os elementos

$$1, \sigma(1), \sigma^2(1), \ldots, \sigma^{r-1}(1)$$

são todos distintos, o que nos fornece o ciclo

$$\sigma_1 = (1\ \sigma(1)\ \sigma^2(1)\ \ldots\ \sigma^{r-1}(1)),$$

Agora, escolhemos o menor elemento $a \in I_n$ que não pertence ao suporte de σ_1 e satisfaz $\sigma(a) \neq a$. Aplicando o mesmo raciocínio à sequência

$$a,\ \sigma(a),\ \sigma^2(a),\ \sigma^3(a), \ldots,$$

obtemos outro ciclo, que chamaremos σ_2, também correspondente à restrição de σ ao seu suporte.

Mostremos que σ_1 e σ_2 não compartilham elementos. Suponhamos que b pertencesse simultaneamente ao suporte de ambos. Então, teríamos $b = \sigma^t(1) = \sigma^s(a)$ para certos inteiros $t, s \geq 0$. Isso implicaria que a estaria no suporte de σ_1, contrariando a escolha de a. Logo, σ_1 e σ_2 são disjuntos.

Repetindo o processo, após um número finito de passos, obtemos ciclos $\sigma_1, \sigma_2, \ldots, \sigma_m$, todos disjuntos. Como a composição $\sigma_1\sigma_2\cdots\sigma_m$ atua da mesma forma que σ sobre todos os elementos de I_n, concluímos que

$$\sigma = \sigma_1\sigma_2\cdots\sigma_m.$$

□

Proposição 2.5.4. *Seja $n > 1$. Então, qualquer permutação em S_n pode ser escrita como um produto de transposições.*

Demonstração. Basta observar que um ciclo de comprimento r pode ser decomposto em transposições. Para $a_1, a_2, \ldots, a_r \in \{1, 2, \ldots, n\}$ distintos, tem-se:

$$(a_1\ a_2\ a_3\ \ldots\ a_r) \ = \ (a_1\ a_r)(a_1\ a_{r-1})\cdots(a_1\ a_2).$$

Assim, dado $\sigma \in S_n$, pela proposição 2.5.3 sabemos que σ pode ser representada como produto de ciclos disjuntos. Aplicando a decomposição acima a cada um desses ciclos, obtemos uma expressão de σ como produto de transposições. Portanto, toda permutação de S_n pode ser escrita nesse formato. □

Exemplo 2.5.2. *O ciclo* $(1\,2\,3\,4)$ *é o produto* $(1\,4)(1\,3)(1\,2) = (1\,2\,3\,4)$*, pois*

$$1 \overset{(1\,2)}{\longmapsto} 2 \overset{(1\,3)}{\longmapsto} 2 \overset{(1\,4)}{\longmapsto} 2$$

$$2 \overset{(1\,2)}{\longmapsto} 1 \overset{(1\,3)}{\longmapsto} 3 \overset{(1\,4)}{\longmapsto} 3$$

$$3 \overset{(1\,2)}{\longmapsto} 3 \overset{(1\,3)}{\longmapsto} 1 \overset{(1\,4)}{\longmapsto} 4$$

$$4 \overset{(1\,2)}{\longmapsto} 4 \overset{(1\,3)}{\longmapsto} 4 \overset{(1\,4)}{\longmapsto} 1$$

2.6 Ação de Grupos

Seja $\mathrm{Sym}(\Omega)$ o grupo das permutações de um conjunto Ω. Dado $x \in \Omega$ e $g \in \mathrm{Sym}(\Omega)$, indicaremos a aplicação de g sobre x por $g \cdot x$. Observe que, para $g, h \in \mathrm{Sym}(\Omega)$, a compatibilidade da composição com a operação de grupo implica

$$h \cdot (g \cdot x) = (hg) \cdot x.$$

Definição 2.6.1. *Sejam G um grupo e Ω um conjunto. Dizemos que G age em Ω se existe um homomorfismo*

$$\varphi : G \longrightarrow \mathrm{Sym}(\Omega).$$

Para cada $g \in G$, a permutação associada $\varphi(g)$ é denotada por $x \mapsto g \cdot x$, com $x \in \Omega$. Assim, a ação pode ser descrita como

$$\begin{array}{rccl} \varphi: & G & \longrightarrow & \mathrm{Sym}(\Omega) \\ & g & \longmapsto & \varphi(g): \quad \Omega \longrightarrow \Omega \\ & & & \qquad\quad x \longmapsto g \cdot x. \end{array}$$

Uma vez que φ é um homomorfismo de grupos, tem-se que $\varphi(e) = id_\Omega$, em que $id_\Omega : \Omega \to \Omega$ é a função identidade de Ω. Assim, $e \cdot x = x$, para todo $x \in \Omega$.

Definição 2.6.2. *Seja G um grupo que age sobre um conjunto Ω. Podemos definir uma relação $\sim$ em Ω da seguinte maneira: para $\alpha, \beta \in \Omega$, escrevemos $\alpha \sim \beta$ se existir algum $g \in G$ tal que*

$$\beta = g \cdot \alpha.$$

A relação $\sim$ é de equivalência e será chamada de relação induzida pela ação de G.

Definição 2.6.3. *As classes de equivalência da relação $\sim$ são chamadas **órbitas** da ação de G em Ω. Para um elemento fixado $\alpha \in \Omega$, sua órbita é dada por*

$$G \cdot \alpha = \{g \cdot \alpha \mid g \in G\}.$$

Definição 2.6.4. *Seja G um grupo agindo sobre um conjunto Ω. Dizemos que a ação é **transitiva** se existir apenas uma órbita sob a ação, isto é, para quaisquer $\alpha, \beta \in \Omega$, existe $g \in G$ tal que*

$$g \cdot \alpha = \beta.$$

Definição 2.6.5. *Seja G um grupo agindo sobre Ω e seja $\Delta \subseteq \Omega$ um subconjunto não vazio. Dizemos que Δ é um **bloco da ação** se, para todo $g \in G$, vale*

$$g \cdot \Delta = \Delta \quad ou \quad g \cdot \Delta \cap \Delta = \varnothing.$$

Um bloco é dito não trivial se satisfaz $|\Delta| > 1$ e $\Delta \neq \Omega$.

Exemplo 2.6.1. *Considere o grupo $G = \langle (1\ 2) \rangle$, que contém apenas o elemento neutro e a transposição $(1\ 2)$. A ação é definida sobre o conjunto*

$$\Omega = \{1, 2, 3, 4, 5\}.$$

A permutação $(1\ 2)$ atua apenas nos elementos 1 e 2, enquanto os demais permanecem fixos. Assim, as órbitas são:

$$\{1, 2\}, \quad \{3\}, \quad \{4\}, \quad \{5\}.$$

Portanto, a ação não é transitiva, pois há mais de uma órbita.

Podemos identificar blocos da ação observando subconjuntos preservados por G. Como $(1\ 2)$ não altera os elementos $3, 4, 5$, cada um dos conjuntos $\{3\}, \{4\}, \{5\}$ é um bloco trivial. Além disso, como $(1\ 2)$ apenas troca os elementos 1 e 2, o conjunto $\{1, 2\}$ também é um bloco não trivial, pois:

$$(1\ 2) \cdot \{1, 2\} = \{1, 2\}.$$

Portanto, os blocos identificados são:

$$\{1, 2\}, \quad \{3\}, \quad \{4\}, \quad \{5\}, \quad \Omega.$$

Definição 2.6.6. *Uma ação transitiva de G em Ω é chamada **primitiva** quando seus únicos blocos são os triviais, isto é, subconjuntos de tamanho 1 ou todo o conjunto Ω.*

Exemplo 2.6.2. *Considere o grupo S_4 atuando sobre $\Omega = \{1, 2, 3, 4\}$ pela ação natural, isto é, $\sigma \cdot i = \sigma(i)$ para todo $\sigma \in S_4$. Essa ação é **transitiva**, pois para quaisquer $i, j \in \Omega$ existe uma permutação em S_4 enviando i para j. Além disso, os únicos blocos dessa ação são os triviais $\{i\}$, com $i \in \Omega$, e o próprio conjunto Ω, pois qualquer subconjunto com dois elementos pode ser enviado para qualquer outro par por uma permutação de S_4. Assim, a ação é **primitiva**.*

2.7 Grupos Livres

Definição 2.7.1. *Seja X um conjunto. Dizemos que um grupo F é* livre sobre *X quando, para qualquer grupo G e qualquer aplicação*

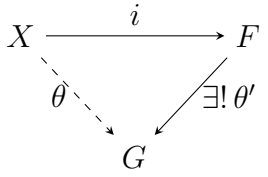
$$\theta : X \longrightarrow G,$$

existe um único homomorfismo de grupos

$$\theta' : F \longrightarrow G$$

tal que $\theta'(x) = \theta(x)$ para todo $x \in X$.

O diagrama abaixo ilustra essa situação:



Assim, um grupo livre sobre o conjunto X pode ser visto como o grupo que se pode construir a partir de X, sem impor relações adicionais além das necessárias para ser um grupo. No que segue, vamos mostrar a existência de um grupo livre sobre um conjunto X.

Seja X um conjunto. Vamos construir um grupo F livre sobre o conjunto X. Se $X = \emptyset$, então F é o grupo trivial $\langle e \rangle$. Se $X \neq \emptyset$, consideremos um conjunto X^{-1} disjunto de X, contendo o mesmo número de elementos, isto é, $|X| = |X^{-1}|$. Denotamos uma bijeção $x \mapsto x^{-1}$ de X em X^{-1}, e dizemos ainda que a imagem de $x \in X$ é x^{-1}. Introduzimos ainda o elemento 1, distinto dos elementos de $X \cup X^{-1}$, que atuará como o elemento neutro.

Uma **palavra** sobre X é uma sequência $(a_1, a_2, \dots)$, com $a_i \in X \cup X^{-1} \cup \{1\}$, tal que existe um número $n \in \mathbb{N}^*$ com $a_k = 1$ para todo $k \geq n$. A sequência constante $(1, 1, 1, \dots)$ é chamada de *palavra vazia* e é denotada por 1.

Uma palavra $(a_1, a_2, \dots)$ sobre X é dita *reduzida* se:

(i) Para todo $x \in X$, x e x^{-1} não aparecem adjacentes, isto é,

$$a_i = x \Rightarrow a_{i+1} \neq x^{-1} \quad \text{e} \quad a_i = x^{-1} \Rightarrow a_{i+1} \neq x,$$

para todo $i \in \mathbb{N}^*$.

(ii) Se $a_k = 1$, então $a_i = 1$ para todo $i \geq k$.

A palavra vazia 1 é reduzida, e toda palavra reduzida não vazia é da forma

$$(x_1^{\lambda_1} x_2^{\lambda_2} \cdots x_n^{\lambda_n}, 1, 1, \dots),$$

onde $n \in \mathbb{N}^*$, $x_i \in X$, e $\lambda_i = \pm 1$. Por convenção, $x^1 = x$ e x^{-1} é o inverso formal de x. Por simplificação, denotaremos essa palavra simplesmente por $x_1^{\lambda_1} x_2^{\lambda_2} \cdots x_n^{\lambda_n}$.

Observa-se ainda que duas palavras reduzidas

$$x_1^{\lambda_1} \cdots x_m^{\lambda_m} \quad \text{e} \quad y_1^{\delta_1} \cdots y_n^{\delta_n}$$

são iguais se, e somente se, $m = n$ e $x_i = y_i$, $\lambda_i = \delta_i$ para cada $i = 1, 2, \dots, n$.

Assim, podemos identificar X com sua imagem e considerar $X \subseteq F(X)$, onde $F(X)$ é conjunto de todas as palavras reduzidas sobre X.

Definimos agora uma operação binária em $F = F(X)$. A palavra vazia 1 atua como elemento identidade, ou seja, $w1 = 1w = w$, para toda palavra $w \in F$.

Queremos que o produto de palavras reduzidas não vazias seja dado por justaposição, isto é,

$$(x_1^{\lambda_1} \cdots x_m^{\lambda_m})(y_1^{\delta_1} \cdots y_n^{\delta_n}) = x_1^{\lambda_1} \cdots x_m^{\lambda_m} y_1^{\delta_1} \cdots y_n^{\delta_n}.$$

No entanto, essa palavra resultante pode não ser reduzida, por exemplo, se $x_m^{\lambda_m} = y_1^{-\delta_1}$. Portanto, definimos o produto como a justaposição seguida de possíveis cancelamentos de termos adjacentes da forma xx^{-1} ou $x^{-1}x$.

Sejam

$$x_1^{\lambda_1} \cdots x_m^{\lambda_m} \quad \text{e} \quad y_1^{\delta_1} \cdots y_n^{\delta_n}$$

palavras reduzidas não vazias sobre X, com $m \leq n$. Seja k o maior inteiro $0 \leq k \leq m$ tal que

$$x_{m-j}^{\lambda_{m-j}} = y_{j+1}^{-\delta_{j+1}}, \quad \text{para } j = 0, 1, \ldots, k-1.$$

Então definimos

$$(x_1^{\lambda_1} \cdots x_m^{\lambda_m})(y_1^{\delta_1} \cdots y_n^{\delta_n}) = \begin{cases} x_1^{\lambda_1} \cdots x_{m-k}^{\lambda_{m-k}} y_{k+1}^{\delta_{k+1}} \cdots y_n^{\delta_n}, & \text{se } k < m, \\ y_{m+1}^{\delta_{m+1}} \cdots y_n^{\delta_n}, & \text{se } k = m < n, \\ 1, & \text{se } k = m = n. \end{cases}$$

Se $m > n$, o produto é definido de maneira análoga. Assim o produto de palavras reduzidas é uma palavra reduzida.

Teorema 2.7.1. *Seja X um conjunto não vazio e $F = F(X)$ o conjunto de todas as palavras reduzidas em X. Então F, munido da operação binária previamente definida, forma um grupo. Além disso, tem-se que $F = \langle X \rangle$.*

Demonstração. A demonstração deste resultado pode ser encontrada em (HUNGERFORD, 1980). □

Teorema 2.7.2. *Seja F o grupo livre gerado por um conjunto X e seja $\varphi : X \to F$ a aplicação de inclusão. Se G é um grupo e $f : X \to G$ é uma aplicação de conjuntos, então existe um único homomorfismo de grupos $\hat{f} : F \to G$ tal que $\hat{f} \circ \varphi = f$.*

Demonstração. A demonstração deste resultado pode ser encontrada em (HUNGERFORD, 1980). □

2.8 O GAP

O *GAP* (The GAP Group, 2024b) (*Groups, Algorithms, Programming*) é um sistema algébrico computacional desenvolvido principalmente para pesquisas em álgebra discreta, com ênfase em teoria de grupos. Sua linguagem de programação e extensa biblioteca de funções o tornam uma ferramenta poderosa para manipular grupos, subgrupos, homomorfismos, anéis e outros objetos algébricos.

A história do *GAP* remonta aos anos 1980, quando foi criado no Lehrstuhl D für Mathematik da Universidade RWTH Aachen, na Alemanha. Inicialmente, o foco era em

fornecer suporte para a teoria de grupos computacional, mas logo evoluiu para abarcar uma série de áreas, como álgebra e combinatória.

O *GAP* é mantido e desenvolvido pela comunidade científica, sendo um software de código aberto amplamente utilizado por pesquisadores pelo mundo, sendo a principal ferramenta de pesquisa no trabalho de Romero (ROMERO, 2018), onde o *GAP* e conceitos da teoria de grupos computacionais são utilizados para otimizar e desenvolver algoritmos que trabalhem com homomorfismo de grupos. Outro exemplo de uso do *GAP* é no caso do trabalho de Hulpke (HULPKE, 2020) em que o *GAP* é utilizado no cálculo de subgrupos de grupos finitos. Não só isso, Hulpke também ajudou na manutenção e criação de alguns algoritmos presentes na biblioteca do *GAP*, alguns deles envolvendo homomorfismo, dos quais um deles foi utilizado nesse trabalho.

Nesta seção, destacamos os principais algoritmos que serão essenciais para mostrar características de problemas e resolve-los de forma inteligente. O manual do *GAP* com informações fundamentais de uso como estrutura léxica, atribuição de variáveis, estruturas de condições e laços e outros podem ser encontrados em (The GAP Group, 2024a). Em Listing 2.1 está o código fonte da função `GroupHomomorphismByImages` que se encontra no pacote *hom* da bliblioteca do *GAP*, responsável por construir homomorfismos de grupos a partir da definição explícita de imagens de geradores.

```
1  #  GroupHomomorphismByImages( <G>, <H>, <Ggens>, <Hgens> )
2  #  GroupHomomorphismByImages( <G>, <H>, <Hgens> )
3  #  GroupHomomorphismByImages( <G>, <H> )
4
5  InstallGlobalFunction( GroupHomomorphismByImages,
6
7  function( arg )
8
9      local  hom, G, H, Ggens, Hgens, arrgh;
10
11     arrgh := arg;
12
13     #verifica se a quantidade de parametros esta correta
14     if not Length(arrgh) in [2..4]
15        or not IsGroup(arrgh[1]) then
16        Error("for usage, see ?GroupHomomorphismByImages");
17     fi;
18
19     #verifica se os geradores em arrg[2] formam um grupo
20     if not IsGroup(arrgh[2]) then
21       arrgh := Concatenation([arrgh[1],Group(arrgh[Length(arrgh)
           ])],
```

```
22                arrgh{[2..Length(arrgh)]});
23      fi;
24
25      G := arrgh[1]; H := arrgh[2];
26
27      #gera os grupos se nao foram passados os geradores como
           parametro
28      if Length(arrgh) = 2 then
29         Ggens := GeneratorsOfGroup(G);
30         Hgens := GeneratorsOfGroup(H);
31      elif Length(arrgh) = 3 then
32         Ggens := GeneratorsOfGroup(G);
33         Hgens := arrgh[3];
34      elif Length(arrgh) = 4 then
35         Ggens := arrgh[3];
36         Hgens := arrgh[4];
37      fi;
38
39      #verifica se os geradores de G sao validos
40      if Length(Ggens) > 0 then
41        if not (IsDenseList(Ggens) and IsHomogeneousList(Ggens)
42        and FamilyObj(Ggens) = FamilyObj(G)) then
43          Error("The generators do not all belong to the source");
44        fi;
45      fi;
46
47      #verifica se os geradores de H sao validos
48      if Length(Hgens) > 0 then
49        if not (IsDenseList(Hgens) and IsHomogeneousList(Hgens)
50           and FamilyObj(Hgens) = FamilyObj(H)) then
51          Error("The images do not all belong to the range");
52        fi;
53      fi;
54
55      hom := GroupGeneralMappingByImages(G, H, Ggens, Hgens);
56
57      #vefirica se o homomorfismo criado respeita as propriedades
           de um mapeamento
58      if IsMapping(hom) then
59        return hom;
60      else
61        return fail;
```

```
62      fi;
63
64  end );
```

Listing 2.1 – Código fonte da função `GroupHomomorphismByImages` no GAP

O objetivo principal dessa função é criar um homomorfismo de grupos $\varphi : G \to H$ especificando apenas como os geradores de G são enviados em H. A ideia fundamental é que, dado um conjunto de geradores, a imagem de todo o grupo é determinada pelas imagens desses geradores.

- A função aceita entre 2 e 4 argumentos: os grupos G e H, uma lista de geradores de G (opcional) e suas imagens em H (opcional).
- Se apenas G e H forem passados, o *GAP* utiliza os geradores padrões de ambos os grupos.
- Caso três argumentos sejam fornecidos, os geradores de G são os padrões e apenas as imagens em H são especificadas.
- Se forem passados quatro argumentos, tanto os geradores de G quanto suas imagens em H devem ser dados explicitamente.
- A função realiza algumas verificações de consistência como: conferir se os grupos fornecidos são válidos;testar se os elementos especificados como geradores realmente pertencem aos grupos correspondentes; garantir que listas de geradores e imagens sejam homogêneas, que evita erros de interpretação pelo *GAP*.

Por fim, a função invoca o procedimento interno `GroupGeneralMappingByImages`, que, resumidamente, constrói a aplicação entre os grupos. Caso essa construção seja bem-sucedida, o resultado é um homomorfismo de grupos válido; caso contrário, a função retorna `fail`.

A função `GroupHomomorphismByImages` é fundamental para o desenvolvimento de soluções porque permite, de maneira prática, definir homomorfismos sem precisar descrevê-los de forma exaustiva em todos os elementos do grupo. Como todo homomorfismo é completamente determinado pelas imagens dos geradores, essa abordagem é eficiente e natural em teoria de grupos computacional.

Exemplo 2.8.1 (Homomorfismo entre grupos de permutação no GAP)**.** *Considere os grupos G e H definidos como subgrupos do grupo simétrico S_3. Seja $G = \langle (1\ 2), (2\ 3) \rangle$ e $H = \langle (1\ 2\ 3) \rangle$, onde desejamos construir um homomorfismo enviando os geradores de G para geradores de H.*

No ambiente GAP, fazemos:

```
gap> G := Group( (1,2), (1,2,3) );;   # G = S3
gap> H := Group( (1,2) );;            # H = C2
# (1,2) -> (1,2); (1,2,3) -> identity
gap> Himgs := [ (1,2), Identity(H) ];
[ (1,2), () ]

gap> hom := GroupHomomorphismByImages(G, H, GeneratorsOfGroup(G), Himgs);
[ (1,2), (1,2,3) ] -> [ (1,2), () ]

gap> IsMapping(hom);
true
gap> Image(hom, (1,2));
(1,2)
gap> Image(hom, (1,2,3));
()
```

A função ***IsMapping*** *verifica se o homomorfismo criado é válido, e as funções* ***Image*** *retornam a imagem de um elemento passando o elemento e o homomorfismo como parâmetros. A aplicação definida acima é um homomorfismo de grupos porque as imagens escolhidas respeitam as relações entre os geradores de G.*

Outra função fundamental no *GAP* é a `PreImagesRepresentative` que se encontra no pacote *hom* da bliblioteca do *GAP*, que permite determinar um elemento do grupo original que seja mapeado em um elemento específico da imagem de um homomorfismo. O código fonte dessa função é apresentado em Listing 2.2.

```
1  #M  PreImagesRepresentative( <hom>, <elm> ) . . . . . .
       via images
2
3  InstallMethod( PreImagesRepresentative, "for PBG-Hom",
       FamRangeEqFamElm,
4    [ IsPreimagesByAsGroupGeneralMappingByImages,
5      IsMultiplicativeElementWithInverse ], 0,
6  function( hom, elm )
7    if HasIsHandledByNiceMonomorphism(Source(hom)) then
8      # if we use the 'AsGGMBI' directly, it will be a composite
           through
9      # a big group
10     return ImagesRepresentative( RestrictedInverseGeneralMapping(
           hom ),
11             elm );
```

```
12    else
13      return PreImagesRepresentative( AsGroupGeneralMappingByImages
          ( hom ),
14              elm );
15    fi;
16  end );
```

Listing 2.2 – Código fonte da função `PreImagesRepresentative` no GAP

O objetivo principal desta função é encontrar um representante no grupo fonte que seja enviado em um elemento específico da imagem de um homomorfismo. Em outras palavras, dado um homomorfismo $\varphi : G \to H$ e um elemento $h \in H$, o algoritmo busca um elemento $g \in G$ tal que $\varphi(g) = h$.

Em suma, a função funciona da seguinte forma:

- Ela verifica inicialmente se a origem do homomorfismo (`Source(hom)`) possui uma estrutura tratada por um monomorfismo (`HasIsHandledByNiceMonomorphism`).
  - Se sim, a função utiliza a `RestrictedInverseGeneralMapping` para restringir a inversa do homomorfismo, garantindo que o cálculo seja feito dentro do subgrupo relevante e evitando composições desnecessárias com grupos grandes, ganhando eficiência em tempo e espaço.
  - Em seguida, chama `ImagesRepresentative` para recuperar o elemento do grupo fonte correspondente ao elemento da imagem.
- Caso contrário, o algoritmo transforma o homomorfismo em uma versão geral via `AsGroupGeneralMappingByImages` e aplica recursivamente `PreImagesRepresentative`.

A função `AsGroupGeneralMappingByImages` é um procedimento interno do *GAP* que transforma um homomorfismo específico em uma representação genérica baseada em imagens de geradores. Essa conversão permite que operações de inversão ou busca de pré-imagens sejam feitas de forma sistemática e consistente, independentemente da forma original do homomorfismo. Essa função não altera a semântica do homomorfismo, mas oferece uma estrutura conveniente e uniforme para que os algoritmos do *GAP* realizem operações de consulta, inversão e manipulação de elementos com eficiência.

Dessa forma, o algoritmo permite a recuperação eficiente de pré-imagens em grupos, sendo uma ferramenta essencial para explorar relações entre elementos de grupos fonte e imagem sem precisar enumerar todas as combinações possíveis.

Exemplo 2.8.2. *A seguir, é apresentado um exemplo simples de utilização da função* `PreImagesRepresentative` *no GAP, cujo objetivo é determinar um representante da*

pré-imagem de um elemento por meio de um homomorfismo entre grupos considerando os resultados do exemplo 2.8.1.

```
# Escolhemos um elemento de H
gap> h := (1,2);;

# Buscando um representante de g em G tal que f(g) = h
gap> g := PreImagesRepresentative(hom, h);;

gap> g;
(1,2)

gap> Image(hom, g);   # Confirma que f(g) = h
(1,2)
```

Ao executar o código, o GAP retorna o elemento $(1\ 2) \in S_3$ *como um representante da pré-imagem de* $(1\ 2) \in C_2$*, verificando que sua imagem sob o homomorfismo realmente coincide com o elemento especificado em* H*.*

Outro algoritmo interessante é a função `Orbit` que se encontra no pacote *orb* da bliblioteca do *GAP*, que permite calcular a órbita de um elemento ou conjunto de elementos sob a ação de uma sequência de geradores de grupo. Como a implementação original localizada no pacote *orb* tem centenas de linhas, em Listing 2.3 está uma implementação alternativa para o algoritmo que mantêm o contexto e escopo do trabalho.

```
1  orbit := function(a, x)
2      local delta, r, b, i;
3      r := Length(x);
4      delta := [a];
5      for b in delta do
6          for i in [1..r] do
7              if not b^x[i] in delta then
8                  Add(delta, b^x[i]);
9              fi;
10         od;
11     od;
12     return delta;
13 end;
```

Listing 2.3 – Implementação alternativa da função `orbit` no GAP

O objetivo desta função é determinar o conjunto de elementos que podem ser alcançados a partir de um elemento inicial a ao aplicar todas as combinações possíveis de uma lista de elementos do grupo $x = [x_1, x_2, \ldots, x_r]$.

Em suma, o algoritmo funciona da seguinte maneira:

- Inicialmente, a lista `delta` é criada contendo apenas o elemento a, que será a base da órbita.
- O algoritmo percorre cada elemento b atualmente em `delta`.
- Para cada b, aplica-se a ação de cada gerador $x[i]$ utilizando a operação $b^{x[i]}$, que é a representação da ação do grupo.
- Se o resultado $b^{x[i]}$ ainda não estiver presente em `delta`, ele é adicionado à lista.
- Este processo é repetido até que não haja mais elementos novos a serem adicionados, garantindo que todas as combinações possíveis de aplicações dos geradores tenham sido consideradas.
- Por fim, o algoritmo retorna `delta`, que contém todos os elementos da órbita de a sob a ação do conjunto de geradores x.

A função `orbit` constitui uma ferramenta eficiente para explorar a estrutura de um grupo por meio de suas órbitas, permitindo identificar de forma sistemática todos os elementos que podem ser gerados a partir de um elemento inicial.

Exemplo 2.8.3. *No exemplo a seguir, considere o grupo S_3, agindo sobre o próprio conjunto por meio da aplicação das permutações em cada elemento.*

```
gap> G := Group( (1,2), (1,2,3) );;

# Calculando a órbita do elemento 1 sob a ação do grupo G
gap> orb := Orbit(G, 1);
[ 1, 2, 3 ]
```

Isso mostra que o grupo S_3 pode mover o elemento 1 para qualquer outro elemento do conjunto $\{1, 2, 3\}$ por meio de suas permutações. Isso significa que a ação é transitiva.

Exemplo 2.8.4. *Podemos também considerar ações que não sejam transitivas. No exemplo a seguir, o grupo $G = \langle (1\,2) \rangle$ atua sobre o conjunto $\{1, 2, 3, 4\}$ porém apenas troca os elementos 1 e 2, mantendo 3 e 4 fixos.*

```
# Definindo um grupo com apenas a transposição (1,2)
gap> G := Group( (1,2) );;

# Calculando as órbitas dos elementos de {1,2,3,4}
gap> orb1 := Orbit(G, 1);
[ 1, 2 ]
gap> orb3 := Orbit(G, 3);
[ 3 ]
```

O que indica que a ação do grupo G não é transitiva, pois nem todos os elementos do conjunto podem ser alcançados a partir de qualquer elemento por meio das permutações do grupo.

Outra função que será fundamental em futuras análises é a função `Blocks`, que é uma função interna do *GAP*, que, no contexto desse trabalho, terá como objetivo calcular todos os subconjuntos $B \subseteq \Omega$ que satisfazem a seguinte propriedade: para todo $g \in G$, ou $g \cdot B = B$ ou $g \cdot B \cap B = \varnothing$, ou seja, calculará os subconjuntos que ou são enviados a si mesmos ou a um conjunto disjunto. A demonstração detalhada original da função `Blocks` pode ser consultada no capítulo 41, seção 11 do manual do *GAP* (The GAP Group, 2024a).

Exemplo 2.8.5. *Consideremos o grupo $G = \langle (1\ 2)(3\ 4), (1\ 3)(2\ 4) \rangle$ que age sobre o conjunto $\Omega = \{1, 2, 3, 4\}$. Vamos utilizar a função `Blocks` no GAP para determinar todos os blocos da ação deste grupo.*

```
gap> G := Group( (1,2)(3,4), (1,3)(2,4) );
gap> Blocks(G, [1..4]);
[ [ 1, 2, 3, 4 ], [ 1, 2 ], [ 3, 4 ] ]
```

A saída indica que existem três blocos na ação:

- $\{1, 2, 3, 4\}$ *é o bloco trivial que corresponde ao conjunto completo;*
- $\{1, 2\}$ *é um bloco não trivial;*
- $\{3, 4\}$ *é outro bloco não trivial.*

Esses blocos mostram que os elementos são particionados em subconjuntos preservados pela ação de G. Assim, a ação não é primitiva.

3 Trabalhos Relacionados

O trabalho de Angie Tatiana Suárez Romero (ROMERO, 2018), intitulado "Computação em Grupos de Permutação Finitos com *GAP*", aborda algoritmos para grupos de permutação e sua implementação no sistema *GAP*, com foco no estudo de ações de grupos em conjuntos finitos. O objetivo principal é descrever métodos para representar grupos em computadores, calcular órbitas e estabilizadores, e explorar conceitos como o conjunto gerador forte (BSGS) por meio do algoritmo Schreier-Sims. Os resultados incluem algoritmos eficientes para identificar grupos transitivos, primitivos e elementos de um grupo usando pesquisas de retrocesso (backtrack). Relaciona-se diretamente com o tema de álgebra computacional, aplicando os recursos do *GAP* ao estudo de grupos simétricos e suas propriedades, destacando como ferramentas algorítmicas podem simplificar e sistematizar cálculos complexos em teoria de grupos.

O objetivo do artigo de Ákos Seress (SERESS, 1997) é apresentar uma introdução abrangente à Teoria Computacional de Grupos, destacando como algoritmos e ferramentas computacionais, como o *GAP*, podem ser usados para resolver problemas em álgebra de grupos. O artigo explora métodos fundamentais, como o procedimento Todd-Coxeter e técnicas de reescrita de termos, para investigar propriedades estruturais de grupos. Os principais resultados incluem a descrição de algoritmos eficientes para manipular grupos de permutação, reconhecer grupos matriciais e calcular tabelas de caracteres, oferecendo uma base sólida para aplicações práticas em áreas como teoria dos grafos e codificação. Este artigo se relaciona diretamente com a álgebra computacional com ênfase em grupos simétricos, já que o *GAP*, um dos softwares destacados, é amplamente utilizado para lidar com representações de grupos simétricos, explorando bases, subconjuntos estabilizadores e estruturas subjacentes para resolver problemas matemáticos e computacionais.

Já o trabalho de Gregory Butler (BUTLER, 1991) tem como principal objetivo apresentar algoritmos fundamentais para a teoria de grupos de permutações, abordando tanto a formulação quanto a análise de complexidade desses algoritmos. O trabalho se concentra no desenvolvimento incremental das técnicas, introduzindo conceitos teóricos de grupos e aplicando-os na construção e otimização de algoritmos específicos. O método utilizado por G. Butler envolve a dedução passo a passo dos algoritmos, partindo de fundamentos básicos da teoria dos grupos até alcançar métodos avançados, como o algoritmo de Schreier-Sims. Os principais resultados incluem algoritmos otimizados para listar elementos de grupos, calcular órbitas e vetores de Schreier, verificar primitividade e regularidade de grupos, e determinar cadeias de estabilizadores e bases geradoras fortes. Este trabalho se relaciona diretamente com o tema, pois o *GAP* é amplamente utilizado para implementar os algoritmos apresentados. Esses métodos são cruciais para resolver proble-

mas de simetria e combinatória em álgebra computacional, especialmente no contexto de grupos simétricos e suas representações em aplicações matemáticas e computacionais.

4 Desenvolvimento

O presente capítulo tem como objetivo descrever o processo de aplicação prática dos conceitos teóricos abordados anteriormente, utilizando o sistema computacional *GAP*. Nesta etapa, busca-se demonstrar como os fundamentos teóricos da álgebra abstrata podem ser implementados de forma computacional, permitindo a verificação, exploração e experimentação de propriedades algébricas de maneira sistemática e reprodutível.

Neste trabalho, o problema central a ser abordado consiste na resolução do *Cubo Mágico* (também conhecido como *Rubik's Cube*) por meio da teoria dos grupos de permutação. O cubo pode ser interpretado como um conjunto de elementos submetidos a transformações que preservam sua estrutura, sendo cada movimento uma permutação de peças. Assim, o *GAP* será empregado para modelar essas permutações, gerar o grupo associado ao cubo e investigar propriedades como órbitas, blocos, elementos geradores e possíveis soluções por meio de representantes e homomorfismo.

Dessa forma, o desenvolvimento a seguir busca demonstrar como o *GAP* pode ser utilizado na representação e resolução de um problema combinatório complexo através de técnicas algébricas. O uso dessa abordagem evidencia a integração entre a teoria dos grupos e a computação, permitindo compreender a estrutura interna do cubo e construir algoritmos capazes de conduzir à sua solução de forma eficiente.

O repositório com os algoritmos criados no decorrer desse trabalho podem ser encontrados em (LYRA, 2025).

4.1 Modelagem do Cubo Mágico no GAP

Para abordar a resolução do Cubo Mágico por meio da teoria computacional de grupo, iniciamos o processo de modelagem no *GAP* pela decomposição estrutural do cubo. Essa decomposição e as técnicas de resolução são baseadas em um curso de álgebra computacional usando o *GAP* que pode ser encontrado em (SCHNEIDER C., LIMA I., 2024). Nesse sentido, resolver o cubo será equivalente a encontrar a sequência de rotações de cada face que leve a configuração atual ao seu estado neutro, ou seja, com as peças de cada cor em suas respectivas faces. A estratégia consiste em destrinchar o cubo em suas seis faces e numerar individualmente cada uma de suas peças, de modo que cada número represente uma posição específica no estado inicial resolvido. Essa numeração é fundamental para representar de forma precisa o estado do cubo e as transformações que ocorrem durante os movimentos. A Figura [1] ilustra essa numeração adotada para as peças do cubo.

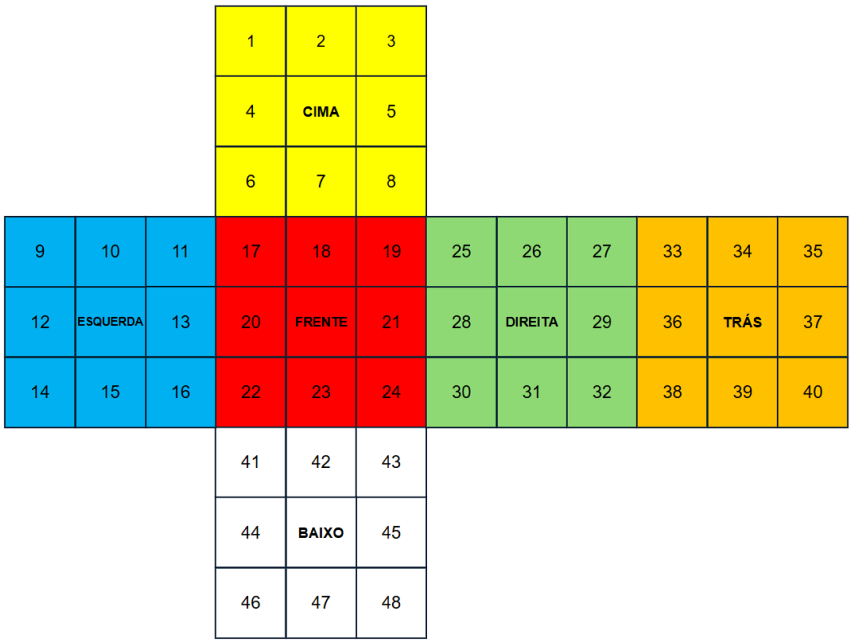


Figura 1 – Numeração das peças do Cubo Mágico utilizada para modelagem no *GAP* (Autoria própria).

Com essa convenção estabelecida, cada movimento possível do cubo pode ser representado como uma permutação dos números, ou seja, uma reordenação dos elementos que descreve o deslocamento das peças. Dessa forma, as rotações de cada face são expressas como permutações cíclicas, permitindo o uso de conceitos algébricos para analisar e manipular as transformações do cubo.

No ambiente do *GAP*, definimos as permutações correspondentes às rotações no sentido **anti-horário**, de cada uma das seis faces do cubo, definindo **c** como a face **CIMA**, **d** como a face **DIREITA**, **e** como a face **ESQUERDA**, **t** como a face **TRÁS**, **f** como a face **FRENTE** e **b** como a face **BAIXO**. Cada permutação foi obtida observando-se a nova posição de cada peça após uma rotação de 90° no sentido anti-horário da face correspondente. Os ciclos indicam as posições trocadas nesse movimento. Por exemplo, ao girar a face superior (CIMA), a peça 1 passa para a posição 6, a 6 para 8, a 8 para 3 e a 3 retorna a 1, resultando no ciclo (1,6,8,3). O trecho de código a seguir apresenta essa definição, na qual cada permutação indica como as posições das peças se alteram em função de um giro completo de uma face:

```
gap>c:=(1,6,8,3)(2,4,7,5)(9,17,25,33)(10,18,26,34)(11,19,27,35);
gap>e:=(1,40,41,17)(4,37,44,20)(6,35,46,22)(9,14,16,11)(10,12,15,13);
gap>f:=(6,16,43,25)(7,13,42,28)(8,11,41,30)(17,22,24,19)(18,20,23,21);
gap>d:=(3,19,43,38)(5,21,45,36)(8,24,48,33)(25,30,32,27)(26,28,31,29);
gap>t:=(1,27,48,14)(2,29,47,12)(3,32,46,9)(33,38,40,35)(34,36,39,37);
gap>b:=(14,38,30,22)(15,39,31,23)(16,40,32,24)(41,46,48,43)(42,44,47,45);
```

```
gap>G:=Group( c, e, f, d, t, b );
```

Com essa construção, o grupo G representa o conjunto de todas as configurações possíveis do cubo que podem ser obtidas por meio de sequências de rotações das faces. Cada elemento de G corresponde a uma permutação específica das peças, e as composições entre essas permutações modelam as sequências de movimentos realizadas no cubo físico. Assim, o problema de encontrar uma solução para o Cubo Mágico torna-se equivalente a determinar uma sequência de elementos de G que leve uma configuração de volta ao estado resolvido.

4.2 Análise das órbitas e blocos na ação do grupo

Com o grupo G definido, que modela todas as rotações válidas do Cubo Mágico, torna-se importante investigar como esse grupo atua sobre o conjunto completo das 48 peças numeradas. Para isso, utilizamos as funções `Orbits` e `Blocks` do GAP, que permitem compreender a organização interna da ação do grupo e identificar padrões de simetria.

A função `Orbits` gera todas as órbitas da ação de um grupo sobre um conjunto, ou seja, os subconjuntos de peças que podem ser alcançados mutuamente por meio das permutações de G. No nosso caso, aplicando a função ao conjunto completo de peças:

```
gap> orbitas := Orbits(G, [1..48]);

[ [ 1, 6, 40, 27, 8, 35, 16, 41, 32, 25, 48, 3,
11, 24, 46, 33, 43, 17, 30, 14, 19, 9, 22, 38 ],
  [ 2, 4, 29, 7, 37, 26, 47, 5, 13, 44, 34, 28,
12, 45, 21, 10, 42, 20, 36, 31,15, 18, 23, 39 ] ]
```

Ao calcular as órbitas em G com todas as peças, obtemos uma lista de duas órbitas, onde cada órbita contém peças que podem se mover entre si por rotações do cubo. Se o grupo G fosse transitivo sobre todas as 48 peças, ou seja, cada peça pudesse atingir a posição de qualquer outra peça, teríamos uma única órbita. Entretanto, a existência de múltiplas órbitas evidencia que os subconjuntos de peças de **cantos** (primeira órbita) e **centros/arestas** (segunda órbita) não se misturam entre si, preservando suas funções estruturais no cubo.

Para aprofundar a análise da estrutura interna, avaliamos os blocos de cada órbita separadamente. Um bloco é um subconjunto de elementos que, sob a ação de qualquer permutação do grupo, é enviado para ele mesmo ou para outro bloco disjunto. Em termos do cubo mágico, isso permite detectar grupos de peças que se comportam de maneira

estruturalmente equivalente sob as rotações. Isso pode ser implementado pelo algoritmo em *GAP* Listing 4.1.

```
BlocosOrbitas:=function(orbitas, G)
    local orb, blocos;
    for orb in orbitas do
        blocos := Blocks(G, orb);;
        Print("Blocos na orbita ", orb, ": ", blocos, "\n");
    od;
end;
```

Listing 4.1 – Algoritmo que calcula os blocos nas órbitas passadas por parâmetro.

A análise combinada das órbitas e dos blocos fornece uma visão detalhada das restrições impostas pelo grupo G, ou seja, a forma que o grupo G preserva o tipo das peças como as de canto e de arestas, permitindo compreender tanto a mobilidade dessas peças quanto os padrões que definem suas posições em relação a estrutura do cubo. Ao executar o algoritmo acima temos o seguinte resultado:

```
gap> BlocosOrbitas(orbitas, G);

Blocos na orbita [ 1, 6, 40, 27, 8, 35, 16, 41, 32, 25,
48, 3, 11, 24, 46, 33, 43, 17, 30, 14, 19, 9, 22, 38 ]:
[ [ 1, 9, 35 ], [ 3, 27, 33 ], [ 6, 11, 17 ], [ 8, 19, 25 ],
  [ 14, 40, 46 ], [ 16, 22, 41 ], [ 24, 30, 43 ], [ 32, 38, 48 ] ]

Blocos na orbita [ 2, 4, 29, 7, 37, 26, 47, 5, 13, 44,
34, 28, 12, 45, 21, 10, 42, 20, 36, 31, 15, 18, 23, 39 ]:
[ [ 2, 34 ], [ 4, 10 ], [ 5, 26 ], [ 7, 18 ],
  [ 12, 37 ], [ 13, 20 ], [ 15, 44 ], [ 21, 28 ],
  [ 23, 42 ], [ 29, 36 ], [ 31, 45 ], [ 39, 47 ] ]
```

Esses resultados são de grande importância para a modelagem do problema. As órbitas indicam quais peças podem realmente trocar de posição entre si, enquanto os blocos revelam como o grupo organiza as relações de simetria entre as partes do cubo.

Dessa forma, esses resultados mostram que é possível identificar corretamente a numeração das peças mesmo que inicialmente elas não estejam numeradas, já que a posição de cada peça em suas respectivas órbitas e blocos permite determinar a qual número ela corresponde, garantindo a coerência da representação algébrica do Cubo Mágico.

Com a análise das órbitas e blocos concluída, podemos agora associar a numeração das peças às suas cores correspondentes em cada face do Cubo Mágico mesmo sem ter o

cubo numerado previamente. Considerando a convenção adotada no diagrama da Figura [1], temos as seguintes associações de cores para cada face: **CIMA** é amarela, **TRÁS** é laranja, **ESQUERDA** é azul, **FRENTE** é vermelha, **DIREITA** é verde e **BAIXO** é branca.

Por fim, com base nas análises e definições das cores, temos como resultado as seguintes tabelas de numeração de acordo com as cores de canto [1] e aresta [2]:

Canto	**Cores do Canto**	**Peças**
1	Amarela / Azul / Laranja	1, 9, 35
2	Amarela / Verde / Laranja	3, 27, 33
3	Amarela / Azul / Vermelha	6, 11, 17
4	Amarela / Vermelha / Verde	8, 19, 25
5	Azul / Laranja / Branca	14, 40, 46
6	Azul / Vermelha / Branca	16, 22, 41
7	Vermelha / Verde / Branca	24, 30, 43
8	Verde / Laranja / Branca	32, 38, 48

Tabela 1 – Numeração respectiva das peças de acordo com as cores de canto do Cubo Mágico

Aresta	**Cores da Aresta**	**Peças**
1	Amarela / Laranja	2, 34
2	Amarela / Azul	4, 10
3	Amarela / Verde	5, 26
4	Amarela / Vermelha	7, 18
5	Azul / Laranja	12, 37
6	Azul / Vermelha	13, 20
7	Azul / Branca	15, 44
8	Vermalha / Verde	21, 28
9	Vermelha / Branca	23, 42
10	Verde / Laranja	29, 36
11	Verde / Branca	31, 45
12	Laranja / Branca	39, 47

Tabela 2 – Numeração respectiva das peças de acordo com as cores de aresta do Cubo Mágico

4.3 Resolução do Cubo Mágico

Com a análise algébrica do Cubo Mágico concluída, passamos agora para a resolução do cubo. É necessário criar uma ferramenta que permita manipular movimentos do cubo como símbolos abstratos, antes de traduzi-los para as permutações específicas das peças. Para isso, construímos um Grupo Livre, denotado Fr, cujos geradores representam cada rotação possível das faces do cubo. O uso do grupo livre é de extrema importância

para traduzir a resolução do cubo, que, em vez de ser dada por um produto de ciclos, passa a ser expressa em função dos geradores de Fr, os quais são simplesmente letras representando cada uma das faces do cubo.

```
gap> Fr := FreeGroup("C","E","F","D","T","B");
```

O grupo livre Fr é então relacionado ao grupo G por meio de um homomorfismo criado pela função `GroupHomomorphismByImages`:

```
gap> hom := GroupHomomorphismByImages(Fr, G, GeneratorsOfGroup(Fr),
GeneratorsOfGroup(G));
```

Com o homomorfismo criado, é possível determinar os passos necessários para resolver o cubo a partir de qualquer configuração atual por meio da execução da função `PreImagesRepresentative`. No entanto, nos falta traduzir para o GAP como está a situação atual do cubo em mãos, ou seja, escrever via código a configuração atual do cubo.

Para traduzirmos uma configuração do cubo para o GAP podemos utilizar da função `PermList` que tem como objetivo transformar um vetor que representa a posição atual das peças em uma permutação expressa como produto de ciclos, o que facilita a manipulação algébrica da configuração do cubo.

Por exemplo, seja a configuração atual do cubo representada pela Figura 2 que é o resultado de girar a face CIMA do cubo no sentido horário.

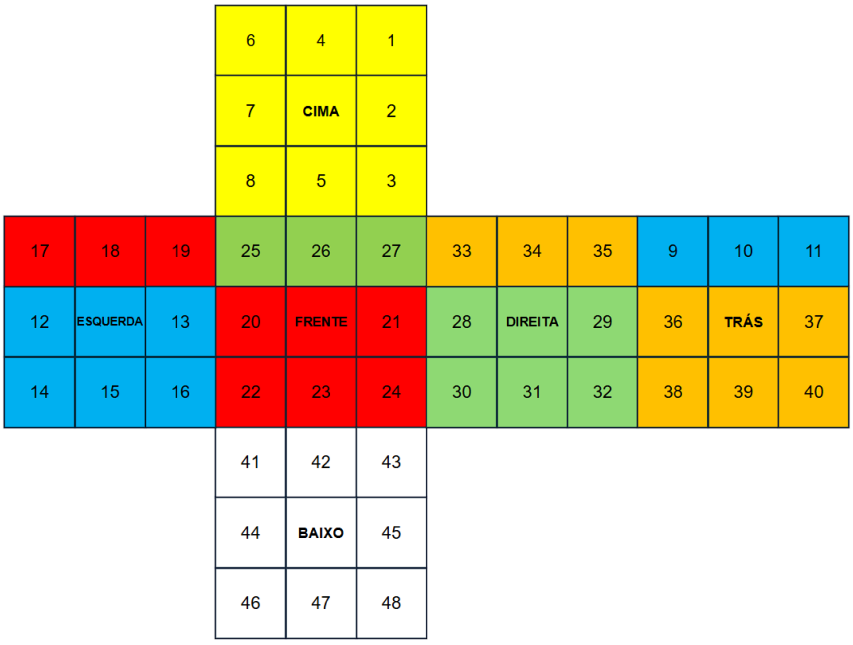


Figura 2 – Configuração atual do Cubo Mágico ao rotacionar a face CIMA no sentido horário. (Autoria própria)

Será criado uma lista que contém 48 elementos, cada um indicando qual peça ocupa determinada posição no cubo: o índice do vetor corresponde à posição física no cubo, enquanto o valor armazenado indica qual peça se encontra naquela posição. Nesse exemplo, a peça 6 está na posição 1, a peça 4 está na posição 2, a peça 1 está na posição 3, e assim por diante. Logo, a lista será descrito da seguinte forma:

```
gap> posicaoAtual := [6, 4, 1, 7, 2, 8, 5, 3,
                      17, 18, 19, 12, 13, 14, 15, 16,
                      25, 26, 27, 20, 21, 22, 23, 24,
                      33, 34, 35, 28, 29, 30, 31, 32,
                      9, 10, 11, 36, 37, 38, 39, 40,
                      41, 42, 43, 44, 45, 46, 47, 48];
```

Ao passar essa lista para a função `PermList`, o *GAP* interpreta a relação entre índices e valores como uma permutação, identificando ciclos de elementos que se movem entre si. O resultado é uma representação compacta e estruturada da configuração do cubo, expressa em produtos de ciclos, que será usada no passo final da resolução do problema.

```
gap> configAtual := PermList(posicaoAtual);

(1,6,8,3)(2,4,7,5)(9,17,25,33)(10,18,26,34)(11,19,27,35)
```

Por fim, supondo que `configAtual` seja a configuração do cubo em mãos, podemos utilizar a função:

```
gap> solucao := PreImagesRepresentative(hom, configAtual);
C
```

Essa função retorna uma sequência de movimentos no grupo livre, que, quando aplicada, leva o cubo à configuração identidade, em outras palavras, resolve o cubo.

A solução será uma sequência de letras que referenciam a face a ser rotacionada no sentido anti-horário: C para a face **CIMA**, D para a face **DIREITA**, E para a face **ESQUERDA**, T para a face **TRÁS**, F como a face **FRENTE** e B para a face **BAIXO**. Letras seguidas de ^-1 significam que devem ser rotacionadas no sentido **horário** e letras seguidas de ^2 devem ser rotacionadas duas vezes no sentido horário se o expoente for negativo e sentido anti-horário caso contrário. Nesse exemplo, para resolver o cubo, basta rotacionar a face CIMA no sentido anti-horário uma vez.

Assim, o trecho de código Listing 4.2 apresenta o algoritmo final `ResolveCubo` constituído de todos os procedimentos anteriores.

```
ResolveCubo:=function(posicaoAtual)
    local c, e, f, d, t, b, G, hom, Fr, configAtual, solucao;

    c:=(1,6,8,3)(2,4,7,5)(9,17,25,33)(10,18,26,34)(11,19,27,35);
    e:=(1,40,41,17)(4,37,44,20)(6,35,46,22)(9,14,16,11)
       (10,12,15,13);
    f:=(6,16,43,25)(7,13,42,28)(8,11,41,30)(17,22,24,19)
       (18,20,23,21);
    d:=(3,19,43,38)(5,21,45,36)(8,24,48,33)(25,30,32,27)
       (26,28,31,29);
    t:=(1,27,48,14)(2,29,47,12)(3,32,46,9)(33,38,40,35)
       (34,36,39,37);
    b:=(14,38,30,22)(15,39,31,23)(16,40,32,24)(41,46,48,43)
       (42,44,47,45);

    G := Group( c, e, f, d, t, b );

    configAtual := PermList(posicaoAtual);
    Fr := FreeGroup("C","E","F","D","T","B");
    hom := GroupHomomorphismByImages(Fr, G,
    GeneratorsOfGroup(Fr),GeneratorsOfGroup(G));

    solucao:=PreImagesRepresentative(hom, configAtual);

    return solucao;
end;
```

Listing 4.2 – Procedimento em GAP que resolve o Cubo Mágico.

4.4 Teste do Algoritmo

Como teste, vamos resolver alguns cubos mágicos embaralhados após várias rotações aleatórias que resultaram nas configurações das Figuras 3, 4 e 5:

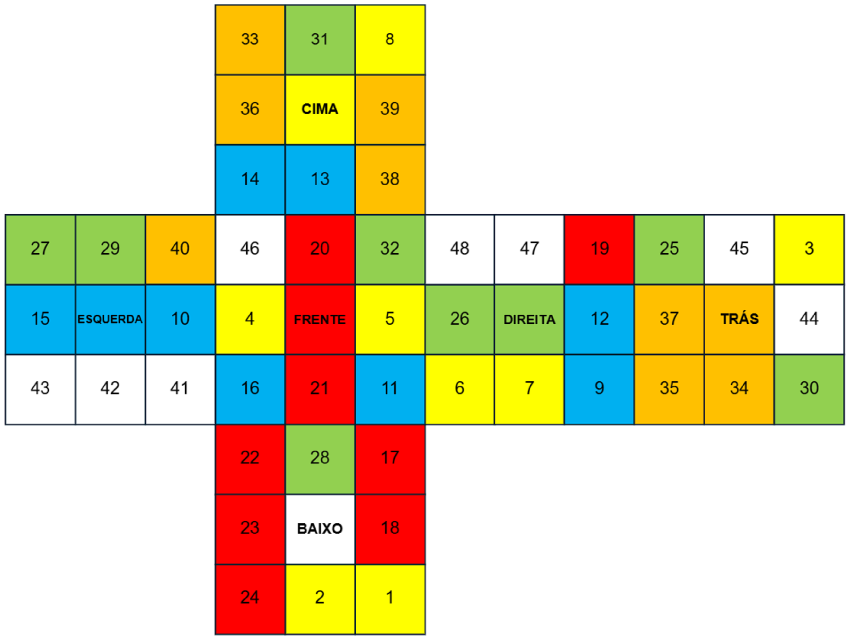


Figura 3 – Primeira configuração de teste do Cubo Mágico. (Autoria própria)

Primeiramente vamos colocar essa configuração em uma lista:

```
gap> posicao1:=[33,31,8,36,39,14,13,38,
                27,29,40,15,10,43,42,41,
                46,20,32,4,5,16,21,11,
                48,47,19,26,12,6,7,9,
                25,45,3,37,44,35,34,30,
                22,28,17,23,18,24,2,1];
```

E por fim, colocar `posicao1` como parâmetro no algoritmo `ResolveCubo`.

```
gap> ResolveCubo(posicao1);
E*T*D^-1*T^-1*F*T^-2
```

Dessa forma, devemos fazer a seguinte sequência de sete rotações para resolver o cubo: rotacionar face ESQUERDA no sentido anti-horário, TRÁS no anti-horário, DIREITA no horário, TRÁS no horário, FRENTE no anti-horário e TRÁS duas vezes no horário.

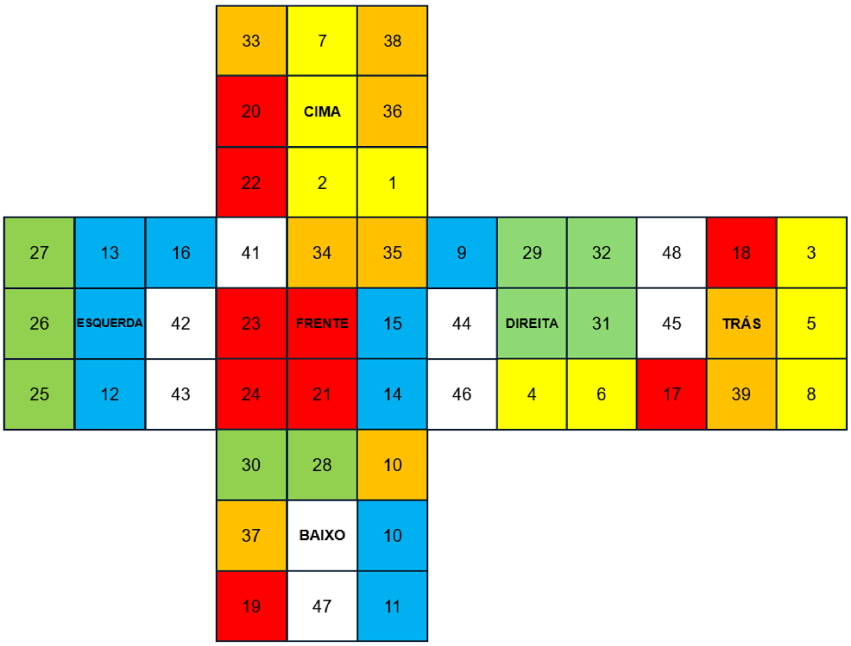


Figura 4 – Segunda configuração de teste do Cubo Mágico. (Autoria própria)

Primeiramente vamos colocar essa configuração em uma lista:

```
gap> posicao2:=[33,7,38,20,36,22,2,1,
                27,13,16,26,42,25,12,43,
                41,34,35,23,15,24,21,14,
                9,29,32,44,31,46,4,6,
                48,18,3,45,5,17,39,8,
                30,28,40,37,10,19,47,11];
```

E por fim, colocar `posicao2` como parâmetro no algoritmo `ResolveCubo`.

```
gap> ResolveCubo(posicao2);
D^-1*C^-1*F*C*E^-1*C^2
```

Dessa forma, devemos fazer a seguinte sequência de sete rotações para resolver o cubo: rotacionar face DIREITA no sentido horário, CIMA no horário, FRENTE no anti-horário, CIMA no anti-horário, ESQUERDA no horário e CIMA duas vezes no anti-horário.

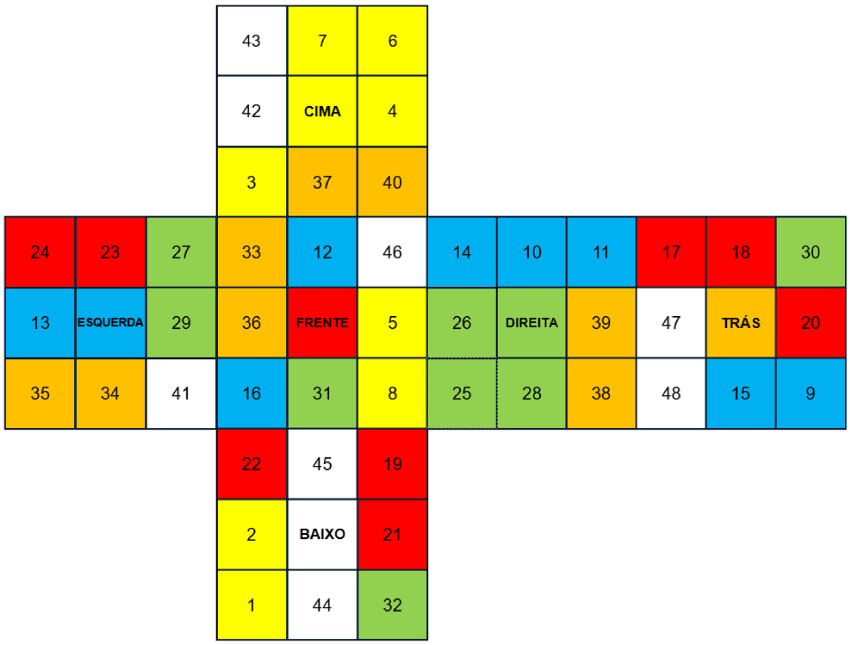


Figura 5 – Terceira configuração de teste do Cubo Mágico. (Autoria própria)

Primeiramente vamos colocar essa configuração em uma lista:

```
gap> posicao3:=[43,7,6,42,4,3,37,40,
                24,23,27,13,29,35,34,41,
                33,12,46,36,5,16,31,8,
                14,10,11,26,39,25,28,38,
                17,18,30,47,20,48,15,9,
                22,45,19,2,21,1,44,32];
```

E por fim, colocar `posicao3` como parâmetro no algoritmo `ResolveCubo`.

```
gap> ResolveCubo(posicao3);
E*C^-1*E^-3*C^-1*D^-1*B^-1
```

Dessa forma, devemos fazer a seguinte sequência de nove rotações para resolver o cubo: rotacionar face ESQUERDA no sentido anti-horário, CIMA no horário, ESQUERDA no horário três vezes, CIMA no horário, DIREITA no horário e BAIXO horário.

5 Conclusão e Trabalhos Futuros

O presente trabalho teve como objetivo explorar a aplicação da Álgebra Computacional no estudo de estruturas algébricas, com ênfase particular nos grupos simétricos, utilizando o sistema computacional *GAP* (*Groups, Algorithms, Programming*). Ao longo do desenvolvimento, foi possível compreender de forma aprofundada como conceitos teóricos da álgebra abstrata podem ser implementados e manipulados computacionalmente, promovendo uma ponte concreta entre a matemática pura e a computação simbólica.

A fundamentação teórica apresentada forneceu a base necessária para o entendimento das estruturas de grupos, subgrupos, homomorfismos e ações de grupos. A partir dessa base, a utilização do sistema *GAP* permitiu não apenas a aplicação prática desses conceitos, mas também a análise de algoritmos fundamentais, que desempenham papel central na investigação da estrutura interna das ações de grupos, e no contexto deste trabalho, no cubo mágico.

A modelagem do Cubo Mágico no ambiente do *GAP* demonstrou de forma concreta como a teoria dos grupos simétricos pode ser aplicada à resolução de problemas reais e complexos.

Conclui-se que a integração entre teoria e prática, mediada por sistemas como o *GAP*, amplia significativamente as possibilidades de pesquisa. Além disso, a abordagem adotada neste trabalho reforça a importância de compreender os fundamentos matemáticos por trás das ferramentas computacionais, permitindo o desenvolvimento de algoritmos mais eficientes e a interpretação correta de seus resultados, contribuindo assim para o fortalecimento da conexão entre a matemática e a computação.

É importante ressaltar que a função `PreImagesRepresentative` retorna um representante da classe lateral dada pelo núcleo do homomorfismo `hom` (4.3), como observado na Observação 2.3.1. Propomos como trabalho futuro a investigação de um algoritmo para que o retorno da função `PreImagesRepresentative` seja uma palavra de tamanho mínimo no grupo livre, garantindo a menor quantidade de movimentos para a resolução do cubo.

Referências

BUTLER, G. Fundamental algorithms for permutation groups. **Lecture Notes in Computer Science, 559, Spring-Verlag, Berlim**, 1991. Citado na página 37.

HERSTEIN, I. **Topics in Algebra**. 2nd edition. ed. [S.l.]: (Wiley International Editions)-John Wiley and Sons (WIE), 1975. <https://archive.org/details/i-n-herstein-topics-in-algebra-2nd-edition-1975-wiley-international-editions-joh>. Citado 2 vezes nas páginas 10 e 13.

HULPKE, A. **Imprimitive Permutations in Primitive Groups**. 2016. <https://arxiv.org/abs/1611.06450>. Citado na página 10.

_____. **Calculating Subgroups with GAP**. 2020. <https://arxiv.org/abs/2012.01595>. Citado na página 29.

HUNGERFORD, T. W. **Graduate Texts in Mathematics v. 73**. [S.l.]: Algebra Springer, 1980. Citado 2 vezes nas páginas 10 e 28.

IEZZI, H. H. D. G. **Álgebra Moderna**. 4ª edição. ed. São Paulo: Saraiva, 2003. Citado na página 10.

LYRA, N. G. de. **Algoritmo para Resolver o Cubo Mágico**. 2025. <https://github.com/NatanGLyra/algoritmo-resolvecubo>. Citado na página 39.

ROMERO, A. T. S. **Computação em grupos de permutação finitos com GAP**. 2018. <https://repositorio.bc.ufg.br/tede/items/dc9aed99-95e0-445a-aad3-dc4829789601/full>. Citado 3 vezes nas páginas 8, 29 e 37.

SCHNEIDER C., LIMA I. **Álgebra Computacional com GAP**. 2024. <https://schcs.github.io/MiniCursoGAP/cubo.html>. [Online; último acesso em 20-Outubro-2025]. Citado na página 39.

SERESS Ákos. An introduction to computational group theory. **Notices Amer.Math. Soc**, 1997. Citado 2 vezes nas páginas 8 e 37.

_____. **Permutation group algorithms**. [S.l.]: Cambridge University Press, 2003. Citado 2 vezes nas páginas 8 e 10.

The GAP Group. **GAP – Groups, Algorithms, Programming, Version 4.13.1**. [S.l.], 2024. Disponível em: <https://docs.gap-system.org/doc/ref/chap0_mj.html>. Citado 2 vezes nas páginas 29 e 36.

_____. **GAP - Groups, Algorithms, Programming, Version 4.13.1**. 2024. <https://www.gap-system.org/>. [Online; accessed 14-Outubro-2024]. Citado 2 vezes nas páginas 8 e 28.