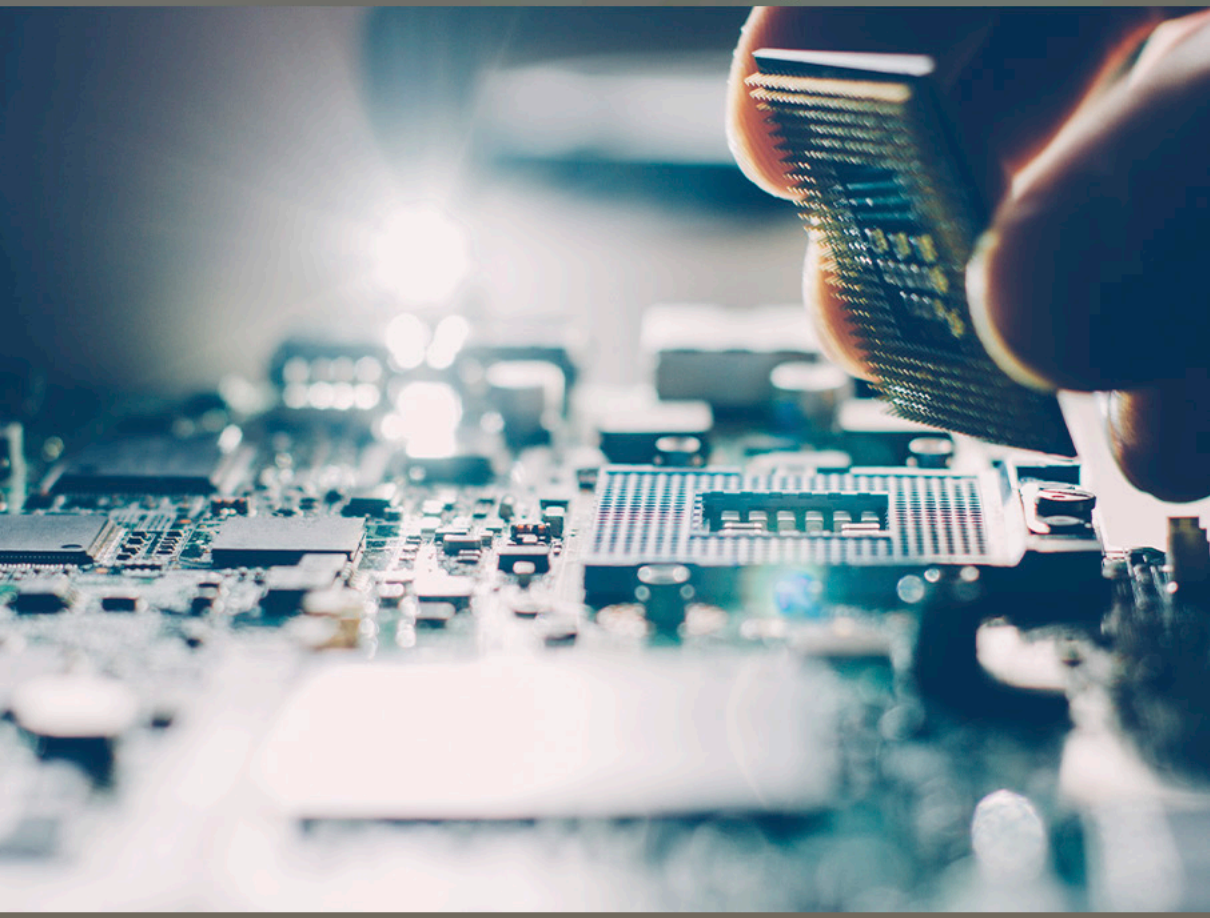
COLEÇÃO

DESAFIOS DAS ENGENHARIAS:

ENGENHARIA DE COMPUTAÇÃO 3



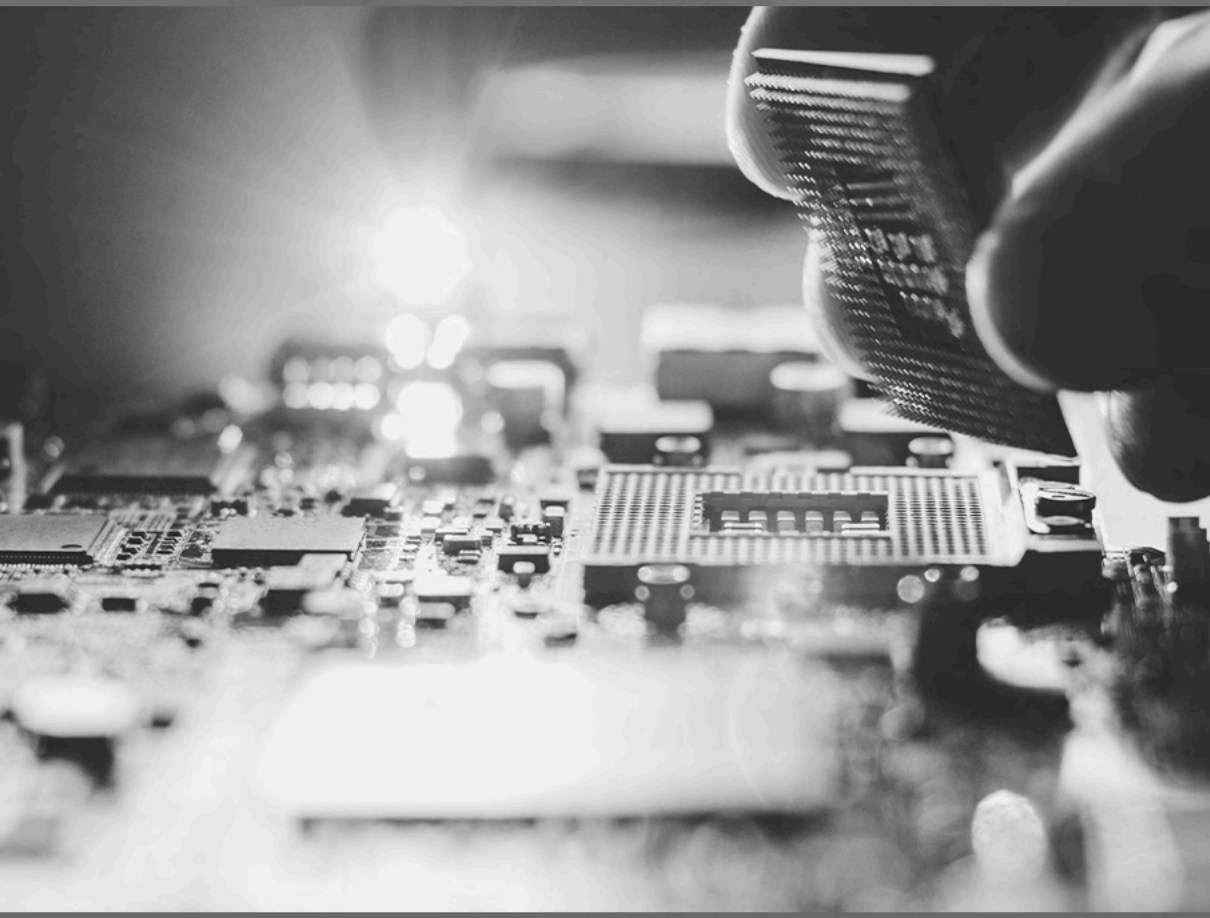
LILIAN COELHO DE FREITAS
(ORGANIZADORA)

Atena Editora
Ano 2021

COLEÇÃO

DESAFIOS DAS ENGENHARIAS:

ENGENHARIA DE COMPUTAÇÃO 3



LILIAN COELHO DE FREITAS
(ORGANIZADORA)

Atena Editora
Ano 2021

Editora chefe
Profª Drª Antonella Carvalho de Oliveira
Editora executiva
Natalia Oliveira
Assistente editorial
Flávia Roberta Barão
Bibliotecária
Janaina Ramos
Projeto gráfico
Camila Alves de Cremo
Daphynny Pamplona
Gabriel Motomu Teshima
Luiza Alves Batista
Natália Sandrini de Azevedo
Imagens da capa
iStock
Edição de arte
Luiza Alves Batista

Atena Editora
Ano 2021

Atena Editora
Ano 2021

Coleção desafios das engenharias: engenharia de computação 3

Diagramação: Daphynny Pamplona
Correção: Gabriel Motomu Teshima
Indexação: Amanda Kelly da Costa Veiga
Revisão: Os autores
Organizadora: Lilian Coelho de Freitas

Dados Internacionais de Catalogação na Publicação (CIP)

C691 Coleção desafios das engenharias: engenharia de computação 3 / Organizadora Lilian Coelho de Freitas. – Ponta Grossa - PR: Atena, 2021.

Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
Inclui bibliografia
ISBN 978-65-5983-619-2
DOI: https://doi.org/10.22533/at.ed.192212911

1. Engenharia de computação. I. Freitas, Lilian Coelho de (Organizadora). II. Título.

CDD 621.39

Elaborado por Bibliotecária Janaina Ramos – CRB-8/9166

Atena Editora
Ponta Grossa – Paraná – Brasil
Telefone: +55 (42) 3323-5493
www.atenaeditora.com.br
contato@atenaeditora.com.br

Atena Editora
Ano 2021

Ciências Exatas e da Terra e Engenharias

DECLARAÇÃO DOS AUTORES

Os autores desta obra: 1. Atestam não possuir qualquer interesse comercial que constitua um conflito de interesses em relação ao artigo científico publicado; 2. Declaram que participaram ativamente da construção dos respectivos manuscritos, preferencialmente na: a) Concepção do estudo, e/ou aquisição de dados, e/ou análise e interpretação de dados; b) Elaboração do artigo ou revisão com vistas a tornar o material intelectualmente relevante; c) Aprovação final do manuscrito para submissão.; 3. Certificam que os artigos científicos publicados estão completamente isentos de dados e/ou resultados fraudulentos; 4. Confirmam a citação e a referência correta de todos os dados e de interpretações de dados de outras pesquisas; 5. Reconhecem terem informado todas as fontes de financiamento recebidas para a consecução da pesquisa; 6. Autorizam a edição da obra, que incluem os registros de ficha catalográfica, ISBN, DOI e demais indexadores, projeto visual e criação de capa, diagramação de miolo, assim como lançamento e divulgação da mesma conforme critérios da Atena Editora.

Atena Editora
Ano 2021

DECLARAÇÃO DA EDITORA

A Atena Editora declara, para os devidos fins de direito, que: 1. A presente publicação constitui apenas transferência temporária dos direitos autorais, direito sobre a publicação, inclusive não constitui responsabilidade solidária na criação dos manuscritos publicados, nos termos previstos na Lei sobre direitos autorais (Lei 9610/98), no art. 184 do Código Penal e no art. 927 do Código Civil; 2. Autoriza e incentiva os autores a assinarem contratos com repositórios institucionais, com fins exclusivos de divulgação da obra, desde que com o devido reconhecimento de autoria e edição e sem qualquer finalidade comercial; 3. Todos os e-book são *open access, desta forma* não os comercializa em seu site, sites parceiros, plataformas de *e-commerce,* ou qualquer outro meio virtual ou físico, portanto, está isenta de repasses de direitos autorais aos autores; 4. Todos os membros do conselho editorial são doutores e vinculados a instituições de ensino superior públicas, conforme recomendação da CAPES para obtenção do Qualis livro; 5. Não cede, comercializa ou autoriza a utilização dos nomes e e-mails dos autores, bem como nenhum outro dado dos mesmos, para qualquer finalidade que não o escopo da divulgação desta obra.

Atena Editora
Ano 2021

APRESENTAÇÃO

A Atena Editora tem a honra de presentear o público em geral com a série de *e-books* intitulada "*Coleção desafios das engenharias: Engenharia de computação*". Em seu terceiro volume, esta obra tem o objetivo de divulgar aplicações tecnológicas da Engenharia de Computação na resolução de problemas atuais, com o intuito de facilitar a difusão do conhecimento científico produzido em várias instituições de ensino e pesquisa do país.

Organizado em 20 capítulos, este volume apresenta temas como utilização de aprendizagem de máquina na avaliação de riscos de infeção por COVID-19; dispositivos automatizados para administração de remédios; comunicação científica apoiada por realidade aumentada; métodos de elementos finitos aplicados na análise de materiais para indústria aeronáutica; aplicações de processamento digital de imagens e de algoritmos genéticos; entre diversas outras aplicações da automação e do desenvolvimento de *software,* combinados para melhorar as atividades do nosso dia-a-dia.

Dessa forma, esta obra contribuirá para aprimoramento do conhecimento de seus leitores e servirá de base referencial para futuras investigações.

Os organizadores da Atena Editora, agradecem especialmente os autores dos diversos capítulos apresentados, parabenizam a dedicação e esforço de cada um, os quais viabilizaram a construção deste trabalho.

Boa leitura.

Lilian Coelho de Freitas

SUMÁRIO

CAPÍTULO 11

ESTUDO SOBRE AUTOMATIZAÇÃO DE EQUIVALÊNCIA DE FUNÇÕES

Data de aceite: 01/11/2021

Lucas Fernando Frighetto
Departamento de Ciência da Computação – Universidade Estadual do Centro-Oeste
Guarapuava– PR– Brasil
http://lattes.cnpq.br/1095181674675390

Fábio Hernandes
Departamento de Ciência da Computação – Universidade Estadual do Centro-Oeste
Guarapuava– PR– Brasil
http://lattes.cnpq.br/2110808766116337

RESUMO: Este projeto tem por objetivo propor um método exato que, prova uma equivalência entre funções. As transformações em estudo são comprovadas por meio da indução matemática. Foi proposto um algoritmo que comprova a equivalência entre um conjunto de funções utilizando manipulação de strings. Para encontrar a igualdade entre funções são realizadas operações matemáticas sobre o números naturais, tais como: distributiva, associativa, comutativa e mínimo múltiplo comum. Como resultado, foram comprovadas as equivalências entre um conjunto de funções constante do banco de dados.

PALAVRAS-CHAVE: Indução matemática, funções, equivalência

A STUDY ABOUT EQUIVALENCE AUTOMATIZATION OF FUNCTIONS

ABSTRACT: This project aims to propose an exact method that, referring to a database, returns an equivalent function. The study transformations are provided through mathematical induction. It was proposed an algorithm that using strings manipulation, proves an equivalence between a set of functions. To find the equality between functions are used mathematical operations on the natural numbers, such as: distributive, associative, commutative and minimum common multiple. As a result, they were proved the equivalences between a set of functions in the database.

KEYWORDS: Mathematical induction, functions, equivalence.

1 I INTRODUÇÃO

Um dos principais problemas da Ciência da Computação é a otimização automática de códigos, os compiladores atuais possuem diversas técnicas para otimizá-los, porém, são limitadas [Aho et al. 2008]. A principal dificuldade nesta otimização está em descobrir como transformá-los preservando seu significado e reduzindo o tempo computacional. Algumas transformações são triviais, como código morto e recalculo de funções, porém outras são mais difíceis, como, por exemplo, transformar um *bubblesort* em *quicksort*. Com isso, foi realizado neste trabalho uma pesquisa com a meta de obter conclusões sobre automatização de otimização de funções.

Existem várias formas pelas quais um

compilador pode efetuar melhorias num programa preservando a sua função, os exemplos mais comuns de melhorias que preservam o significado do código podem ser encontradas em [Aho et al. 2008].

Entre outros trabalhos encontrados na literatura relacionados à otimização de código, destacam-se a otimização de laços [Allen 1969], que foi um dos estudos pioneiros sobre otimização de código, este é sobre otimização em estruturas de repetição. Outros trabalhos citados aqui são a otimização de código em *Power Architectures* [IBM 2016] e a aproximação estática para o problema de compactação de código horizontal para a classe paralela, sincronizada em arquiteturas não homogêneas [De Gloria 1991]. Além destes, ressalta-se também o trabalho sobre ferramentas para otimização de código e sistema de avaliação de processamento de imagem para o sistema *PAPRICA-3* [Bertozzi 1999].

Seguindo a temática de otimização de códigos, este projeto tem por objetivo o desenvolvimento de uma técnica capaz de comprovar a equivalência entre funções envolvendo números naturais, técnica esta que poderá ser aplicada à otimização de código, pois se provada a equivalência entre duas funções, o compilador poderá trocar uma função recorrente de alto custo por outra função fechada de custo computacional menor. O desenvolvimento dessa técnica consiste em implementar operações matemáticas (distributiva, associativa, comutativa, mínimo múltiplo comum), por meio de manipulação de *strings,* a fim de provar a equivalência entre funções por meio da indução matemática.

Este trabalho está organizado da seguinte forma: na Seção 2 estão os materiais e métodos, a Seção 3 aborda o algoritmo proposto, na Seção 4 estão os resultados computacionais, enquanto que na Seção 5 são destacadas as conclusões e trabalhos futuros.

2 | MATERIAIS E MÉTODOS

Nesta seção são abordados alguns conceitos e definições utilizados no trabalho. Inicialmente é introduzido o problema da preservação do significado do código e as técnicas de transformação, em seguida é abordado o método da indução matemática, sendo esta a base para confirmação das equivalências.

2.1 Indução matemática

Dentre todos os números já considerados, os naturais (N) foram os pioneiros, inicialmente com o intuito de contar. Apesar desses números serem os mais simples, não quer dizer que sejam totalmente entendidos, havendo ainda muitos mistérios que os cercam [Iezzi e Murakami 2004, Hazzan 2004, Menezes 2005].

Provar operações matemáticas entre números naturais não é tão simples, sendo o princípio de indução matemática o mais utilizado [Iezzi 2005]. Porém, além da prova entre operações matemáticas, a indução matemática [Antonio 2017] é uma importante ferramenta utilizada para verificar conjecturas sobre padrões de termos em sequências.

Neste estudo, a indução matemática é utilizada para verificar a equivalência entre funções a fim de compará-las em seus respectivos custos computacionais e aplicar estes resultados à otimização de código.

A fim de visualizar a ideia da indução matemática, imagine uma coleção de dominós colocados numa sequência (formação) de tal forma que a queda do primeiro dominó força a queda do segundo, que força a queda do terceiro, e assim sucessivamente, até todos os dominós caírem. Em outras palavras, se a equivalência entre funções for encontrada para um número qualquer e para o próximo número, então consequentemente será válida para todos os números subsequentes.

A igualdade entre as funções pode ser encontrada aplicando transformações de propriedades algébricas. O problema é descobrir a sequência de transformações que levam as funções à um resultado em comum.

2.1.1 Gauss e a Indução Matemática

Conta-se a seguinte história [Hefez 2009] sobre o matemático alemão Carl Friedrich Gauss (1777-1855), quando ainda garoto. Na escola, o professor, para aquietar a turma de Gauss, mandou os alunos calcularem a soma de todos os números naturais de 1 a 100. Qual não foi a surpresa quando, pouco tempo depois, o menino deu a resposta: 5050. Indagado como tinha descoberto tão rapidamente o resultado, Gauss, então com nove anos de idade, descreveu o método a seguir.

$S_n = 1 + 2 + \cdots + n$, o objetivo é encontrar uma fórmula fechada para S_n. Somando a igualdade de S_n, membro a membro, com ela mesma, porém com as parcelas do segundo membro em ordem invertida, temos que:

$$\begin{array}{rcccccc} S_n &=& 1 &+& 2 &+\cdots+& n \\ S_n &=& n &+& (n-1) &+\cdots+& 1 \\ \hline 2S_n &=& (n+1) &+& (n+1) &+\cdots+& (n+1) \end{array}.$$

$$2S_n = n(n+1) \text{ e, portanto, } S_n = \frac{n(n+1)}{2}.$$

2.1.2 Princípio de Indução Matemática

Dado um subconjunto *S* do conjunto dos números naturais N, tal que e sempre que um número $n \in S$, o número *n+1* também pertence a *S*, tem-se que S = N.

Esta simples propriedade fornece uma poderosa técnica de demonstração matemática, a *demonstração por indução*, que consiste em [Antonio 2017]:

> "*Suponha que seja dada uma sentença matemática P(n) que dependa de uma variável natural n, a qual se torna verdadeira ou falsa quando substituímos n por um número natural dado qualquer. Tais sentenças são ditas sentenças abertas definidas sobre o conjunto dos naturais*".

Logo, a demonstração por indução matemática segue o teorema a seguir:

Teorema. Seja *P(n)* uma sentença aberta sobre N. Suponha que:

(i) *P(1)* é verdadeira; e

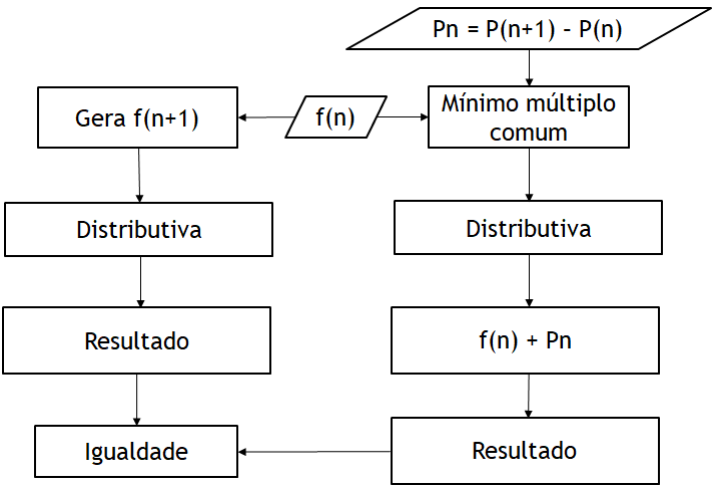
(ii) qualquer que seja $n \in S$, sempre que *P(n)* é verdadeira, segue que *P(n + 1)* é verdadeira.

Então, *P(n)* é verdadeira para todo $n \in \mathbb{N}$.

3 | ALGORITMO PROPOSTO

O algoritmo proposto procura a igualdade entre uma *P(n),* neste trabalho associado à funções recorrentes e uma função *f(n),* neste trabalho associado à funções fechadas. O método tem como entrada uma função *f(n)* e o termo necessário da *P(n)* que é *P(n+1) - P(n)*. Existe um número muito grande de possibilidades de transformações e portanto não há um método exato e ótimo para encontrar a igualdade entre funções. Para resolver este problema, foram desenvolvidos dois caminhos (possibilidades de encontrar a equivalência), ambos compostos de transformações válidas, porém com resultados finais diferentes, ou seja, se entre as funções da base de dados não for encontrada a equivalência utilizando o caminho 1, esta será encontrada por meio do caminho 2 (2ª possibilidade).

Inicialmente é gerada uma função *f(n+1)*, a partir da função *f(n)*, ao substituir variável *n* por *n+1*, então, aplica-se a operação distributiva para a função e, após, é realizada a soma da função *f(n)* inicial com o termo da *P(n)*. Para isso, primeiramente é executada uma operação de mínimo múltiplo comum, então aplica-se a distributiva e, para finalizar, somam-se os termos, se o resultado for igual ao obtido por meio da substituição de *n* por *n+1* seguido da distributiva, então conclui-se que a equivalência é verdadeira. Esta parte da implementação está ilustrada no fluxograma a seguir.



Algoritmo 1. Passos de parte da implementação.

O que difere o Caminho 2 (segunda possibilidade) do primeiro na implementação é que para algumas funções é necessário utilizar uma sequência de transformações não triviais como, por exemplo, quando se busca verificar se o divisor de uma função está contido no divisor da outra, o que minimiza o cálculo do mínimo múltiplo comum, porém utilizam-se os mesmos métodos do Caminho 1. Outra diferença é a necessidade de multiplicar a *f(n+1)* por *a/a* tal que *f(n+1) = a/b*.

Exemplo. Caminho 2.

Dada a seguinte soma:

$$\frac{n}{(n+1)} + \frac{(n+1)}{(n+1)(n+2)}.$$

Utilizando o Caminho 1 (Figura 2) tem-se o seguinte resultado:

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

Já considerando o Caminho 2, o resultado será:

$$\frac{n}{(n+1)} + \frac{(n+1)}{(n+1)(n+2)} = \frac{n(n+2) + (n+1)}{(n+1)(n+2)},$$

isso ocorre porque o denominador da primeira fração está contido no denominador da segunda.

Os métodos implementados podem ser encontrados no Apêndice.

4 I RESULTADOS COMPUTACIONAIS

A linguagem de programação utilizada foi JAVA, pois possui uma biblioteca de *strings* que oferece muitos recursos para facilitar a implementação de operações de propriedades algébricas.

As principais operações matemáticas implementadas foram: distributiva, associativa, comutativa, mínimo múltiplo comum, soma e reconhecedores de expressões.

Para a validação do resultados, foi proposta uma base de dados constituída de cinco funções, que estão descritas e provadas matematicamente por meio da indução matemática. Estas funções foram escolhidas pois já são utilizadas para exemplos em outros trabalhos da literatura sobre indução matemática [Antonio 2017], [Hefez 2009].

4.2 Base de dados e resultados obtidos

1. Soma dos n ímpares - P (n) $1 + 3 + ... + (2n - 1) = n^2$

Prova por indução matemática

I. Observe que *P(1)* é verdadeira, já que a fórmula é trivialmente válida para *n = 1*.

II. Suponha que, para algum *n* natural, *P(n)* seja verdadeira; ou seja, que:

$$1 + 3 + ... + (2n - 1) = n^2$$

Provando que *P(n+1)* é verdadeira. Somando *2n+1*, que é o próximo número ímpar, após *2n-1*, a ambos os lados da igualdade de *P(n)*, obtêm-se que a igualdade também verdadeira, pois:

$$1 + 3 + ... + (2n - 1) + (2n + 1) = n^2 + (2n + 1) = (n + 1) = n^2 + 2n + 1.$$

O Quadro 1 contém a saída da implementação para a função anterior.

```
f(n) = ((1n)(1n))/(1)
f(n+1) = (((1)(1n+1))((1)(1n+1)))/(1)
pn = (((2)(1n+1)-1)/(1)
Após o mínimo múltiplo comum:
pn = (((1)(2)(1n+1)-1)/(1)
f(n) = ((1)(1n)(1n))/(1)
f(n) + p(n+1) = f(n+1):
((1)(1n)(1n))/(1)+(((1)(2)(1n+1)-1)/(1) = (((1)(1n+1))((1)(1n+1)))/(1)
distributiva: (((1)(1n+1))((1)(1n+1)))/(1)
((1n+1)(1n+1))/(1)
(1nn+2n+1)/(1)
f(n+1) = (1nn+2n+1)/(1)

distributiva: ((1)(1n)(1n))/(1)
((1n)(1n))/(1)
(1nn)/(1)
distributiva: (((1)(2)(1n+1)-1)/(1)
(((2)(1n+1)-1)/(1)
((2n+2-1)/(1)
f(n) = (1nn)/(1)
pn = ((2n+2-1)/(1)
f(n) + pn = (1nn+2n+1)/(1)
```

Quadro 1. Resultados da soma dos n ímpares P(n)

2. Soma dos n números naturais - $P(n): 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

Prova por indução matemática

I. Observe que P(1) é verdadeira, pois:

$$P(1): 1 = \frac{1(1+1)}{2}.$$

II. Suponha que, para algum n natural, P(n) seja verdadeira, ou seja, que:

$$P(n): 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

Provando que *P(n+1)* é verdadeira. Somando *n+1* a ambos os lados da igualdade de *P(n)*, obtêm-se:

$$P(n+1): 1 + 2 + 3 + \cdots + n + (n+1) = \frac{n(n+1)}{2} + (n+1)$$

desenvolvendo a segunda parte da igualdade tem-se:

$$\frac{n(n+1)}{2} + (n+1) = \frac{n(n+1) + 2(n+1)}{2} = \frac{(n+1)(n+2)}{2},$$

que mostra que *P(n+1)* também verdadeira.

Vale destacar que:

$$\frac{(n+1)(n+2)}{2} = \frac{n^2+3n+2}{2}.$$

No Quadro 2 está contida a saída da implementação para a função anterior.

```
f(n) = ((1n)(1n+1))/(2)
f(n+1) = (((1)(1n+1))((1)(1n+1)+1))/(2)
pn = p(n+1) – p(n) = (1n+1)/(1)
Após o mínimo múltiplo comum:
pn = (2)(1n+1)/(2)
f(n) = ((1)(1n)(1n+1))/(2)
f(n) + p(n+1) = f(n+1):
((1)(1n)(1n+1))/(2)+(2)(1n+1)/(2) = (((1)(1n+1))((1)(1n+1)+1))/(2)
distributiva: (((1)(1n+1))((1)(1n+1)+1))/(2)
((1n+1)(1n+1+1))/(2)
(1nn+3n+2)/(2)
f(n+1) = (1nn+3n+2)/(2)

distributiva: ((1)(1n)(1n+1))/(2)
((1n)(1n+1))/(2)
(1nn+1n)/(2)
distributiva: (2)(1n+1)/(2)
(2n+2)/(2)
f(n) = (1nn+1n)/(2)
pn = (2n+2)/(2)
f(n) + pn= (1nn+3n+2)/(2)
```

Quadro 2. Resultados da soma dos n números - P (n): 1 + 2 + 3 + ... + n

3. Soma dos quadrados - $P(n)\colon 1^2+2^2+3^2+\cdots+n^2=\frac{n(n+1)(2n+1)}{6}$

Prova por indução matemática:

I. Observe que *P(1)* é verdadeira, pois:

$$P(1)\colon 1=\frac{1(1+1)(2(1)+1)}{6}.$$

II. Suponha que, para algum *n* natural, *P(n)* seja verdadeira, ou seja, que:

$$P(n)\colon 1^2+2^2+3^2+\cdots+n^2=\frac{n(n+1)(2n+1)}{6},$$

Prova de que *P(n+1)* é verdadeira. Somando *n+1* a ambos os lados da igualdade de *P(n)*, obtêm-se:

$$P(n+1)\colon 1^2+2^2+3^2+\cdots+n^2+(n+1)^2=\frac{n(n+1)(2n+1)}{6}+(n+1)^2,$$

desenvolvendo a segunda parte da igualdade tem-se:

$$\frac{n(n+1)(n+2)}{6}+(n+1)=\frac{n(n+1)(n+2)+6(n+1)}{6}=\frac{(n+1)[n(n+2)+6]}{6}$$
$$=\frac{(n+1)[(n+1)+1][2(n+1)+1]}{6}=\frac{(n+1)(n+2)(2n+3)}{6}.$$

que mostra que *P(n+1)* também verdadeira.

Vale destacar que:

$$\frac{(n+1)(n+2)(2n+3)}{6}=\frac{2n^3+9n^2+13n+6}{6}.$$

O Quadro 3 contém a saída da implementação para a função anterior.

f(n) = ((1n)(1n+1)(2n+1))/(6)
f(n+1) = (((1)(1n+1))((1)(1n+1)+1)((2)(1n+1)+1))/(6)
pn = ((1n+1)(1n+1))/(1)
Após o mínimo múltiplo comum:
pn = ((6)(1n+1)(1n+1))/(6)
f(n) = ((1)(1n)(1n+1)(2n+1))/(6)
f(n) + p(n+1) = f(n+1):
((1)(1n)(1n+1)(2n+1))/(6)+((6)(1n+1)(1n+1))/(6) = (((1)(1n+1))((1)(1n+1)+1)((2)(1n+1)+1))/(6)
distributiva: (((1)(1n+1))((1)(1n+1)+1)((2)(1n+1)+1))/(6)
((1n+1)(1n+1+1)((2)(1n+1)+1))/(6)
((1nn+3n+2)((2)(1n+1)+1))/(6)
((1nn+3n+2)(2n+2+1))/(6)
(2nnn+13n+9nn+6)/(6)
f(n+1) = (2nnn+13n+9nn+6)/(6)

distributiva: ((1)(1n)(1n+1)(2n+1))/(6)
((1n)(1n+1)(2n+1))/(6)
((1nn+1n)(2n+1))/(6)
(2nnn+1n+3nn)/(6)
distributiva: ((6)(1n+1)(1n+1))/(6)
((6n+6)(1n+1))/(6)
(6nn+12n+6)/(6)
f(n) = (2nnn+1n+3nn)/(6)
pn = (6nn+12n+6)/(6)
f(n) + pn = (2nnn+13n+9nn+6)/(6)

Quadro 3. Resultados da soma dos quadrados $-\ \boldsymbol{P(n)}\colon \mathbf{1^2+2^2+3^2+\cdots+n^2}$

4. Soma das n frações $P(n)\colon \frac{1}{1(2)}+\frac{1}{2(3)}+\frac{1}{3(4)}+\cdots+\frac{1}{n(n+1)}=\frac{n}{(n+1)}$

Prova por indução matemática

I. Observe que *P(1)* é verdadeira, pois:

$$P(1)\colon \frac{1}{1(2)}=\frac{1}{(1+1)}.$$

II. Suponha que, para algum *n* natural, *P(n)* seja verdadeira, ou seja, que:

$$P(n)\colon \frac{1}{1(2)}+\frac{1}{2(3)}+\frac{1}{3(4)}+\cdots+\frac{1}{n(n+1)}=\frac{n}{(n+1)}$$

Provando que *P(n+1)* é verdadeira. Somando a ambos os lados da igualdade de *P(n)*, obtêm-se:

$$P(n+1): \frac{1}{1(2)} + \frac{1}{2(3)} + \frac{1}{3(4)} + \cdots + \frac{1}{n(n+1)} + \frac{1}{(n+1)(n+2)} = \frac{n}{(n+1)} + \frac{1}{(n+1)(n+2)}$$

desenvolvendo a segunda parte da igualdade tem-se:

$$\frac{n}{(n+1)} + \frac{1}{(n+1)(n+2)} = \frac{n(n+2)}{(n+1)(n+2)} + \frac{1}{(n+1)(n+2)} = \frac{(n+1)^2}{(n+1)(n+2)} = \frac{n+1}{n+2}$$

que mostra que *P(n+1)* também verdadeira.

O Quadro 4 contém a saída da implementação para a função anterior.

```
f(n) = (1n)/(1n+1)
f(n+1) = ((1)(1n+1))/((1)(1n+1)+1)
pn = (1)/((1n+1)(1n+2))
distributiva: ((1)(1n+1))
(1n+1)
distributiva: ((1)(1n+1)+1)
(1n+1+1)
f(n+1) = f(n+1) * (1n+1)(1n+1)
distributiva: (1n+1)(1n+1)
(1nn+2n+1)
distributiva: (1n+1+1)(1n+1)
(1nn+3n+2)
f(n+1) = (1nn+2n+1)/(1nn+3n+2)

f(n) = (1n)/(1n+1)
pn = (1)/((1n+1)(1n+2))
distributiva: ((1n)(1n+2))
(1nn+2n)
distributiva: ((1n+1)(1n+2))
(1nn+3n+2)
distributiva: (1)
distributiva: ((1n+1)(1n+2))
(1nn+3n+2)
f(n) = (1nn+2n)/(1nn+3n+2)
pn = (1)/(1nn+3n+2)
f(n) + pn = (1nn+2n+1)/(1nn+3n+2)
```

Quadro 4. Resultados da soma das n fracções $P(n): \frac{1}{1(2)} + \frac{1}{2(3)} + \frac{1}{3(4)} + \cdots + \frac{1}{n(n+1)}$

Analisando o resultado obtido (Figura 7) com o resultado da prova por indução tem-se que:

$$\frac{n+1}{n+2} = \frac{(n+1)^2}{(n+1)(n+2)} = \frac{n^2+2n+1}{n^2+3n+2},$$

mostrando que o resultado da Figura 7 é equivalente.

5. Soma dos ímpares quadrados $P(n): 1^2 + 3^2 + 5^2 + \cdots + (2n-1)^2 = \frac{1}{3}n(2n-1)(2n+1)$

Prova por indução matemática:

a. Observe que *P(1)* é verdadeira, pois

$$P(1): 1^2 = \frac{1}{3}1(2(1)-1)(2(1)+1).$$

b. Suponha que, para algum *n* natural, *P(n)* seja verdadeira, ou seja, que:

$$P(n): 1^2 + 3^2 + 5^2 + \cdots + (2n-1)^2 = \frac{1}{3}n(2n-1)(2n+1),$$

Provando que *P(n+1)* é verdadeira. Somando $(2n + 1)^2$ a ambos os lados da igualdade de *P(n)*, obtêm-se:

$$P(n+1): 1^2 + 3^2 + 5^2 + \cdots + (2n-1)^2 + (2n+1)^2 = \\ = \frac{1}{3}n(2n-1)(2n+1) + +(2n+1)^2$$

desenvolvendo a segunda parte da igualdade tem-se:

$$\frac{1}{3}n(2n-1)(2n+1) + (2n+1)^2 = \frac{1}{3}(2n+1)[n(2n-1) + 3(2n+1)] =$$

$$= \frac{1}{3}(2n+1)[n(2n-1) + 6n + 3] = \frac{1}{3}(2n+1)[n(2n-1) + 4n + 2n + 3] =$$

$$= \frac{1}{3}(2n+1)[2n^2 - n + 4n + 2n + 3] = \frac{1}{3}(2n+1)[2n^2 + 3n + 2n + 3] =$$

$$= \frac{1}{3}(2n+1)[n(2n+3) + (2n+3)] = \frac{1}{3}(2n+1)[(2n+3)(n+1)] =$$

$$= \frac{1}{3}(n+1)(2n+1)(2n+3)$$

que mostra que *P(n+1)* também verdadeira.

Vale destacar que:

$$\frac{1}{3}(n+1)(2n+1)(2n+3) = \frac{4n^3 + 12n^2 + 11n + 3}{3}.$$

O Quadro 5 contém a saída da implementação para a função anterior.

f(n) = ((1n)(2n-1)(2n+1))/(3)
f(n+1) = (((1)(1n+1))((2)(1n+1)-1)((2)(1n+1)+1))/(3)
pn = (2n+1)(2n+1)/(1)
Após o mínimo múltiplo comum:
pn = (3)(2n+1)(2n+1)/(3)
f(n) = ((1)(1n)(2n-1)(2n+1))/(3)
f(n) + p(n+1) = f(n+1):
((1)(1n)(2n-1)(2n+1))/(3)+(3)(2n+1)(2n+1)/(3) = (((1)(1n+1))((2)(1n+1)-1)((2)(1n+1)+1))/(3)
distributiva: (((1)(1n+1))((2)(1n+1)-1)((2)(1n+1)+1))/(3)
((1n+1)((2)(1n+1)-1)((2)(1n+1)+1))/(3)
((1n+1)(2n+2-1)(2n+2+1))/(3)
((2nn+3n+1)(2n+2+1))/(3)
(4nnn+11n+12nn+3)/(3)
f(n+1) = (4nnn+11n+12nn+3)/(3)

distributiva: ((1)(1n)(2n-1)(2n+1))/(3)
((1n)(2n-1)(2n+1))/(3)
((2nn-1n)(2n+1))/(3)
(4nnn-1n+0nn)/(3)
distributiva: (3)(2n+1)(2n+1)/(3)
(6n+3)(2n+1)/(3)
(12nn+12n+3)/(3)
f(n) = (4nnn-1n+0nn)/(3)
pn = (12nn+12n+3)/(3)
f(n) + pn = (4nnn+11n+12nn+3)/(3)

Quadro 5. Resultados da soma dos ímpares quadrados $- P(n)$: $1^2 + 3^2 + \cdots + (2n-1)^2$

4.3 Comentários gerais

Como verificou-se, por meio dos resultados obtidos, o algoritmo proposto conseguiu obter a equivalência entre algumas funções que trabalham com números naturais, cumprindo assim o objetivo deste projeto. Porém, ressalta-se que este esteve limitado a funções de baixa complexidade, ou seja, funções com sequências simples.

Com relação aos resultados vale ressaltar que apenas os obtidos na soma das n frações foram destoantes dos demais, porém matematicamente a equivalência foi comprovada.

5 | CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi estudado como automatizar a equivalência entre funções com números naturais, a fim de explicar processos de preservação do significado do código na etapa de otimização. Como contribuição, foi proposto um algoritmo com a possibilidade otimizar funções recorrentes, fazendo uso das operações básicas da matemática elementar (associativa, distributiva, comutativa, soma, dentre outras), já as equivalências entre funções foram provada por meio da indução matemática.

Considerando que para algumas funções não é trivial encontrar a sequência de operações matemáticas, que resultem numa igualdade com outra função, os testes executados ficaram restritos às funções contidas neste artigo.

A grande dificuldade deste estudo foi a análise e manipulação de expoentes e somatórios complexos, isto devido à complexidade da analise léxica e semântica, por isso foram implementados métodos limitados à funções relativamente simples.

Para trabalhos futuros pode ser estudado como implementar a técnica de Gauss na fase de otimização de código, no processo de compilação, a fim de transformar funções recorrentes em funções fechadas. Funções recorrentes possuem complexidade em função de n, enquanto que funções fechadas possuem complexidade constante, que considera-se um ganho em desempenho computacional significativo.

REFERÊNCIAS

Aho, A. V. et al. Compiladores – Princípios, Técnicas e Ferramentas. Massachusets: Pearson Addison – Wesley, 2ª edição, 2008.

Allen, F. O tratado fundamental sobre técnicas para otimização de loops – 1969. Disponível em: <http://www.francesallen.com/. Acesso: junho/2017>. Data de acesso: 07/07/2017.

Antonio, A.F. - Sequências Indução Matemática – UFMG. Disponível em: <http://homepages.dcc.ufmg.br/~loureiro/md/md_4SequenciaEInducaoMatematica.pdf>. Data de acesso: 07/07/2017.

Bertozzi, M. Broggi, A. Elsevier - Tools for code optimization and system evaluation of the image processing system PAPRICA-3 - UNIPR. 1999. Disponível em: <http://www.ce.unipr.it/people/broggi/publications/jsa.pdf>. Data de acesso: 07/07/2017.

De Gloria, A. Faraboschi, P. Elsevier: A boltzmann machine approach to code optimization 9ª edição, 1991. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167819105800420?via%3Dihub>. Data de acesso: 07/07/2017.

Hazzan, S. Fundamentos de matemática elementar: combinatória e probabilidade. São Paulo: Atual, 7ª edição, 2004.

Hefez, A. UFF. OBMEP: Indução Matemática. 2009. Disponível em: <http://www.obmep.org.br/docs/apostila4.pdf>. Data de acesso: 07/07/2017.

IBM. Code optimization with the IBM XL compilers on Power architectures. 2016. Disponível em: <http://www-01.ibm.com/support/docview.wss?uid=swg27005174&aid=1>. Data de acesso: 07/07/2017.

Iezzi, G. Fundamentos de matemática elementar: complexos, polinômios e equações. São Paulo: Atual, 7ª edição, 2005.

Iezzi, G. e Murakami, C. Fundamentos de matemática elementar: conjuntos e funções. São Paulo: Atual, 8ª edição, 2004.

Menezes, P. B. Matemática discreta: para computação e informática. Porto Alegre: Sagra Luzzatto, 2ª edição, 2005.

```
private static String addTerm(String f, int n, String nstr) {
    if (nstr.length() == 0) {
        if (!Character.isDigit(f.charAt(f.length() - 2))) {
            if (n >= 0) {
                return f.substring(0, f.length() - 1) + "+" + n + ")";
            }
            return f.substring(0, f.length() - 1) + n + ")";
        }
        int i;
        for (i = f.length() - 2; Character.isDigit(f.charAt(i)); i--) { }
        if (f.charAt(i) == '-') {
            n = n + Integer.parseInt(f.substring(i, f.length()));
            return f.substring(0, i) + n + ")";
        } else {
            n = n + Integer.parseInt(f.substring(i + 1, f.length() - 1));
            return f.substring(0, i) + "+" + n + ")";
        }
    }
    int instr = -1;
    for (int i = 0; i < f.length(); i++) {
        if (f.charAt(i) == 'n') {
            int x = nstr.length();
            instr = i;
            for (i = i; f.charAt(i) == 'n'; i++) {
                x--;
            }
            if (x == 0) {
                break;
            } else {
                instr = -1;
            }
        }
    }
    if (instr == -1) {
```

```
        int cont;
        for (int i = 0; i < f.length(); i++) {
            if (f.charAt(i) == 'n') {
                for (cont = 0; f.charAt(i) == 'n'; i++) {
                    cont++;
                }
                if (cont < nstr.length()) {
                    for (i = i; Character.isDigit(f.charAt(i)) || f.charAt(i) == 'n'; i--) {}
                }
                if (n < 0) {
                    return f.substring(0, i) + n + nstr + f.substring(i, f.length());
                } else {
                    return f.substring(0, i) + "+" + n + nstr + f.substring(i, f.length());
                }
            }
        }
        return "(" + n + nstr + f.substring(1);
    } else {
        int i;
        for (i = instr - 1; i >= 0 && Character.isDigit(f.charAt(i)); i--) {}
        if (f.charAt(i) == '-') {
            n = n + Integer.parseInt(f.substring(i, instr));
            if (n >= 0) {
                f = f.substring(0, i) + "+" + n + f.substring(instr, f.length());
            } else {
                f = f.substring(0, i) + n + f.substring(instr, f.length());
            }
        } else {
            n = n + Integer.parseInt(f.substring(i + 1, instr));
            f = f.substring(0, i) + "+" + n + f.substring(instr, f.length());
        }
    }
    return f;
}
```

```
private static String distr(String f) {
```

```
        f = f.replace(")(", "),(");
        String parts[] = f.split(",");
        f = "()";
        for (int i = 0; i < parts[0].length(); i++) {
          for (i = i; i < parts[0].length() && !Character.isDigit(parts[0].charAt(i));
i++) {}
          int j;
          for (j = i; j < parts[0].length() && Character.isDigit(parts[0].charAt(j)); j++)
{ }
          if (j > i) {
            int n = Integer.parseInt(parts[0].substring(i, j));
            String nstr = "";
            for (int temp = j; parts[0].charAt(temp) == 'n'; temp++) {
              nstr = nstr + "n";
            }
            for (int k = 0; k < parts[1].length(); k++) {
              for (k = k; k < parts[1].length() &&
!Character.isDigit(parts[1].charAt(k)); k++) {}
              int l;
              for (l = k; l < parts[1].length() &&
Character.isDigit(parts[1].charAt(l)); l++) {}
              String nstr1 = "";
              for (int temp = l; temp < parts[1].length() && parts[1].charAt(temp) ==
'n'; temp++) { nstr1 = nstr1 + "n"; }
              if (l > k) {
                int n1 = Integer.parseInt(parts[1].substring(k, l)) * n;
                if ((parts[0].charAt(i - 1) == '-' && (parts[1].charAt(k - 1) == '+' ||
parts[1].charAt(k - 1) == '(')) || ((parts[0].charAt(i - 1) == '+' || parts[0].charAt(i - 1)
== '(') && parts[1].charAt(k - 1) == '-')) { n1 *= -1; }
                nstr1 = nstr + nstr1; f = addTerm(f, n1, nstr1);
              }
              k = l;
            }
            i = j;
          }
        }
        if (f.charAt(1) == '+') {
```

```
        f = "(" + f.substring(2);
      }
      return f;
   }
```

```
private static int middleIndex(String f) {
      String aux = f;
      int middle = f.indexOf(")(");
      while (middle != -1) {
         if (f.charAt(middle - 1) == ')' || f.charAt(middle + 2) == '(') {
            aux = aux.substring(0, middle) + "$" + aux.substring(middle + 1,
aux.length());
            middle = aux.indexOf(")(");
         } else {
            break;
         }
      }
      return middle;
   }
```

```
public static String distributive(String f) {
      System.out.println("distributiva: "+f);
      int middle = middleIndex(f);
      int start = 0, end = 0;
      f = checkparents(f);
      while (middle != -1) {
         for (int i = middle; i >= 0; i--) {
            if (f.charAt(i) == '(') {
               start = i;
               break;
            }
         }
         for (int i = middle + 1; i < f.length(); i++) {
            if (f.charAt(i) == ')') {  end = i;   break;  }
         }
         String aux = f.substring(start, middle + 1) + f.substring(middle + 1, end +
1);
```

```
        f = "(" + f.substring(2);
      }
      return f;
   }
```

```
private static int middleIndex(String f) {
      String aux = f;
      int middle = f.indexOf(")(");
      while (middle != -1) {
         if (f.charAt(middle - 1) == ')' || f.charAt(middle + 2) == '(') {
            aux = aux.substring(0, middle) + "$" + aux.substring(middle + 1,
aux.length());
            middle = aux.indexOf(")(");
         } else {
            break;
         }
      }
      return middle;
   }
```

```
public static String distributive(String f) {
      System.out.println("distributiva: "+f);
      int middle = middleIndex(f);
      int start = 0, end = 0;
      f = checkparents(f);
      while (middle != -1) {
         for (int i = middle; i >= 0; i--) {
            if (f.charAt(i) == '(') {
               start = i;
               break;
            }
         }
         for (int i = middle + 1; i < f.length(); i++) {
            if (f.charAt(i) == ')') {  end = i;   break;  }
         }
         String aux = f.substring(start, middle + 1) + f.substring(middle + 1, end +
1);
```

```
        f = f.replace(aux, distr(aux));
        f = checkparents(f);
        System.out.println(f);
        middle = middleIndex(f);
      }
      System.out.println("");
      return f;
   }
```

```
private static String checkparents(String f) {
      for (int i = 1; i < f.length(); i++) {
         if (f.charAt(i) == '(' && f.charAt(i - 1) == '(') {
            for (int j = i + 1; j < f.length() - 1; j++) {
               if (f.charAt(j) == ')' && f.charAt(j + 1) == ')') {
                  f = f.substring(0, i) + f.substring(i + 1, j) + f.substring(j + 1,
f.length());
                  i = j;
                  break;
               } else if (f.charAt(j) == ')' && f.charAt(j + 1) != ')') {
                  break;
               }
            }
         }
      }
      for (int i = 1; i < f.length(); i++) {
         if (f.charAt(i) == '(') {
            for (int j = i + 1; j < f.length() - 1; j++) {
               if (f.charAt(j) == ')') {
                  if (f.charAt(i - 1) == '(' && (f.charAt(j + 1) == '+' || f.charAt(j + 1)
== '-')) {
                     f = f.substring(0, i) + f.substring(i + 1, j) + f.substring(j + 1,
f.length());
                  } else {
                     break;
                  }
               }
            }
```

```
            int j;
            for (j = i - 1; Character.isDigit(j); j--) {
            }
            int k;
            for (k = i + 1; Character.isDigit(k); k++) {
            }
            int newn = Integer.parseInt(f.substring(j, i)) -
Integer.parseInt(f.substring(i + 1, k + 1));

            f = f.substring(0, j) + newn + f.substring(k + 1, f.length());
        }
    }

    return f;
  }
```

```
public static String mmc(String f, String f1) {
    int a0 = -1;
    int a1 = -1;
    int b0 = -1;
    int b1 = -1;
    for (int i = 0; i < f.length() - 1; i++) {
      if (f.charAt(i) == '/') {
        for (i = i; i < f.length(); i++) {
          if (f.charAt(i) == '(') { a0 = i; }
          if (f.charAt(i) == ')') {
            a1 = i + 1;
            break;
          }
        }
        break;
      }
    }

    for (int i = 0; i < f1.length() - 1; i++) {
      if (f1.charAt(i) == '/') {
```

```
            for (i = i; i < f1.length(); i++) {
                if (f1.charAt(i) == '(') {
                    b0 = i;
                }
                if (f1.charAt(i) == ')') {
                    b1 = i + 1;
                    break;
                }
            }
            break;
        }
    }
    String a = f.substring(a0, a1);
    String b = f1.substring(b0, b1);
    String result = distr(a + b);
    f = f.substring(0, a0) + result + f.substring(a1, f.length());
    int i;
    for (i = 0; f.charAt(i) == '('; i++) { }
    f = f.substring(0, i - 1) + b + f.substring(i - 1, f.length());
    return f;
}
```

```
private static String add2(String f, String f1) {
    String temp = f.substring(0, f.indexOf("/"));
    for (int i = 1; i < f1.indexOf("/"); i++) {
        int j;
        for (j = i; Character.isDigit(f1.charAt(j)); j++) {          }
        int n;
        if (j > i) {
            if (f1.charAt(i - 1) == '-') {
                n = Integer.parseInt(f1.substring(i - 1, j));
            } else {
                n = Integer.parseInt(f1.substring(i, j));
            }
            String nstr = "";
            int l;
```

```
            for (l = j; f1.charAt(l) == 'n'; l++) {
                nstr += "n";
            }
            temp = addTerm(temp, n, nstr);
        }
        i = j;
    }
    return temp + f.substring(f.indexOf("/"));
}
```

```
public static String npn1(String f) {
    for (int i = 0; i < f.length(); i++) {
        if (f.charAt(i) == 'n') {
            int aux = 0;
            for (int j = i - 1; j >= 0; j--) {
                if (!Character.isDigit(j)) {
                    f = f.substring(aux, j) + "(" + f.substring(j, i) + ")(1n+1)" +
f.substring(i + 1, f.length());
                    aux = i + 1;
                    i += 6;
                    break;
                }
            }
        }
    }
    return f;
}
```

```
public static void inducao2(String f, String pn) {
    String passoIndutivo = npn1(f);
    System.out.println("f(n+1) = "+passoIndutivo);
    System.out.println("f(n) = "+f);
     System.out.println("p(n+1) = " + pn);
    String divIndutive[] = passoIndutivo.split("/");
    divIndutive[0] = distributive(divIndutive[0]);
    divIndutive[1] = distributive(divIndutive[1]);
    String aux = divIndutive[0];
    divIndutive[0] = distributive(divIndutive[0] + aux);
    divIndutive[1] = distributive(divIndutive[1] + aux);
    System.out.println("f(n+1) = " + divIndutive[0] + "/" + divIndutive[1]);
    System.out.println("");
    String divf[] = f.split("/");
    String divfn[] = pn.split("/");
    System.out.println("f(n) = " + divf[0] + "/" + divf[1]);
    System.out.println("p(n+1) = " + divfn[0] + "/" + divfn[1]);
    System.out.println("");
    if (!divf[1].equals(divfn[1])) {
       String temp = checkparents(divfn[1].replace(divf[1], ""));
       if (temp.length() != divfn.length) {
          divf[1] = "(" + divf[1] + temp + ")";
          divf[0] = "(" + divf[0] + temp + ")";
       }
    }
    divf[0] = distributive(divf[0]);
    divf[1] = distributive(divf[1]);
    divfn[0] = distributive(divfn[0]);
    divfn[1] = distributive(divfn[1]);
    System.out.println("f(n) = " + divf[0] + "/" + divf[1]);
    System.out.println("p(n+1) = " + divfn[0] + "/" + divfn[1]);
    System.out.print("f(n) + p(n+1) = ");

    System.out.println(add2(divf[0] + "/" + divf[1], divfn[0] + "/" + divfn[1]));
  }
```

```
System.out.println("0");
FunctionOperations.inducao("((1n)(1n))/(1)", "(((2)(1n+1)-1)/(1)");
System.out.println("1");
FunctionOperations.inducao("((1n)(1n+1)(2n+1))/(6)", "((1n+1)(1n+1))/(1)");
System.out.println("2");
FunctionOperations.inducao2("(1n)/(1n+1)", "(1)/((1n+1)(1n+2))");
System.out.println("4");
FunctionOperations.inducao("((1n)(2n-1)(2n+1))/(3)", "(2n+1)(2n+1)/(1)");
System.out.println("6");
FunctionOperations.inducao("((1n)(1n+1))/(2)", "(1n+1)/(1)");
```

ÍNDICE REMISSIVO

P

R

S

T

U

V

W

COLEÇÃO

DESAFIOS DAS ENGENHARIAS:

ENGENHARIA DE COMPUTAÇÃO 3

www.atenaeditora.com.br
contato@atenaeditora.com.br
@atenaeditora
www.facebook.com/atenaeditora.com.br

Atena Editora
Ano 2021

COLEÇÃO

DESAFIOS DAS ENGENHARIAS:

ENGENHARIA DE COMPUTAÇÃO 3

www.atenaeditora.com.br
contato@atenaeditora.com.br
@atenaeditora
www.facebook.com/atenaeditora.com.br

Atena Editora
Ano 2021