

**UNIVERSIDADE VEIGA DE ALMEIDA – UVA**  
**BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO**

**IMPLEMENTAÇÃO DE ÁRVORES BINÁRIAS DE BUSCA EM**  
**APLICAÇÃO DE APOIO AO ENSINO DAS ESTRUTURAS DE DADOS**

**ANDRES FELIPE GIRALDO MORALES**

**RIO DE JANEIRO**

**2019**

**UNIVERSIDADE VEIGA DE ALMEIDA - UVA**

**ANDRES FELIPE GIRALDO MORALES**

Monografia apresentada ao curso de Engenharia da Computação da Universidade Veiga de Almeida, como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação.

Orientador(a): André Lucio de Oliveira

**IMPLEMENTAÇÃO DE ÁRVORES BINÁRIAS DE BUSCA EM  
APLICAÇÃO DE APOIO AO ENSINO DAS ESTRUTURAS DE DADOS**

**RIO DE JANEIRO**

**2019**

**UNIVERSIDADE VEIGA DE ALMEIDA - UVA**  
**BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO**

**ANDRES FELIPE GIRALDO MORALES**

**IMPLEMENTAÇÃO DE ÁRVORES BINÁRIAS DE BUSCA EM  
APLICAÇÃO DE APOIO AO ENSINO DAS ESTRUTURAS DE DADOS**

Monografia apresentada como  
requisito parcial à conclusão do curso em  
Bacharel em Engenharia da Computação.

APROVADA EM:

CONCEITO: \_\_\_\_\_

BANCA EXAMINADORA:

---

**PROF. MSc ANDRÉ LUCIO DE OLIVEIRA**

---

**PROF. MSc CAMILLA LOBO PAULINO**

---

**PROF. DSc EDGAR AUGUSTO GONÇALVES GURGEL DO AMARAL**

**Coordenação de Engenharia da Computação**

Rio de Janeiro

*Dedico este trabalho ao meu orientador, que  
diante das diversas adversidades foi  
imensamente compreensivo e solidário.*

## **AGRADECIMENTOS**

Gostaria de agradecer à minha mãe, por ter cuidado de mim mesmo diante de problemas de saúde e financeiros muito complicados, por seu empenho incomparável em me educar e por ter me ensinado a importância do amor e gentileza.

Agradeço ao meu pai, por ter me ensinado quase tudo que sei na minha carreira profissional e por ser um grande companheiro cuja presença está entre as coisas que mais prezo na vida.

Agradeço à minha avó que me cuidou e criou durante muitos anos, com seu amor imensurável, e me ensinou tudo que sei sobre respeitar ao próximo.

Agradeço à minha segunda mãe, Maria Fernanda, por ter me amado e me acolhido, mesmo sem ser sua responsabilidade.

Agradeço à minha noiva Rachel e ao meu sogro Luis Alfredo, por terem me brindado companhia em um período difícil da minha vida, e por serem a maior inspiração que me levou a continuar com os meus estudos.

Agradeço ao meu amigo e colega de turma Felipe, por ter ajudado a tornar esta experiência bastante prazerosa.

Agradeço aos meus professores e ao meu coordenador por me guiar com bastante dedicação por esta jornada.

“Não tenha medo de crescer lentamente, tenha medo apenas de ficar parado.”

Provérbio Chinês.

## RESUMO

A desistência nos cursos de graduação por parte dos alunos é muitas vezes dada pela frustração no entendimento por parte das metodologias de ensino utilizadas atualmente. Para diminuir as taxas de evasão por parte dos alunos, se dá a necessidade da utilização de metodologias ativas, onde o aluno é o principal ator no aprendizado, sendo servindo o professor como seu orientador. O aluno é engajado através de desafios e recompensas de forma a aumentar seu interesse e aprendizado pelo conteúdo. Dado que nos cursos de graduação em computação, uma disciplina de alta relevância como algoritmos e estruturas de dados apresentam altas taxas de reprovação, é encontrada a necessidade do emprego de metodologias ativas para o ensino do conteúdo da disciplina de estruturas de dados. O SYM (Structure Your Mind) é uma ferramenta que foi desenvolvida exatamente com estes objetivos, entretanto, ela apenas dá suporte às estruturas como pilhas, filas e listas encadeadas desordenadas e ordenadas. Este trabalho dá seguimento à ferramenta SYM, implementando as estruturas árvore binárias de busca e as binárias de busca com balanceamento de altura (AVL), com o objetivo de aprimorar a ferramenta de apoio ao ensino e que possa atender ao conteúdo das disciplinas de estruturas de dados.

Palavras-Chave: Metodologias ativas, Simulador, Estruturas de dados, Árvores binárias

## **ABSTRACT**

Student dropout in undergraduate courses is often due to frustration in understanding the teaching methods used today. To reduce student's dropout rates, there is the need to use active learning methodologies, which center on the student being the main actor in the learning process, and the teacher being his guide. The student is engaged through challenges and rewards to increase their interest and the quality of the content learned. Given that in computing undergraduate courses, a discipline of high relevance such as algorithms and data structures have high failure rates, there is a need to employ active learning methodologies in the data structure's course syllabus. Structure Your Mind (SYM) is a tool that was developed exactly for these purposes; however, it only supports structures like unordered and ordered stacks, queues, and linked lists. This work follows the one made by the SYM tool, implementing binary search tree structures and height-balanced search binary trees (AVL), in order to improve the learning support tool and so it can meet the active content of data structures courses.

**Keywords:** Active methodologies, Simulator, Data Structures, Binary trees



## LISTA DE ILUSTRAÇÕES

Figura 1: Apresentação do GameLogic, jogo para aprendizado da lógica de programação .....	18
Figura 2: Comandos do GameLogic e formação do algoritmo solução .....	18
Figura 3: Cisco Packet Tracer .....	19
Figura 4: Roteamento de pacotes no Cisco Packet Tracer .....	20
Figura 5: Representação de um vetor na memória .....	22
Figura 6: Representação linear de uma matriz bidimensional na memória.....	22
Figura 7: Estrutura de uma pilha .....	23
Figura 8: Exemplo de operações em uma fila .....	24
Figura 9: Lista simplesmente encadeada.....	25
Figura 10: Lista duplamente encadeada .....	25
Figura 11: Árvore binária.....	26
Figura 12: Árvores distintas para um mesmo conjunto de valores .....	27
Figura 13: Exclusão e substituição de um nó .....	27
Figura 14: Rotação LL .....	28
Figura 15: Rotação LR em árvore AVL.....	29
Figura 16: Tela do SYM em funcionamento.....	30
Figura 17: Área de código do SYM em depuração .....	31
Figura 18: Busca e inserção em árvore binária no SYM.....	33
Figura 19: Cálculo de altura e índice de balanceamento de um nó no SYM .....	34
Figura 20: Rotação LR em árvore AVL no SYM .....	34
Figura 21: Novos scripts adicionados ao SYM.....	35
Figura 22: Principais botões da interface do SYM.....	36
Figura 23: Novos casos de usos implementados no SYM .....	36
Figura 24: Funcionalidade Criar árvore .....	37
Figura 25: Busca pelo nó de valor 12.....	39
Figura 26: Busca e inserção do elemento 12, deslocamento dos nós.....	40
Figura 27: Busca, remoção e substituição de um nó .....	43
Figura 28: Cálculo dos índices de balanceamento .....	47
Figura 29: Balanceamento e Rotação de árvore AVL.....	51
Figura 30: Impressão e Percurso pela árvore .....	53

## LISTA DE TABELAS

Tabela 1: Percentual de desistência por curso.....	13
Tabela 2: Percentual de desistência por ano de curso (matriculados em 2010) .....	13
Tabela 3: Tipos de dados primitivos na linguagem de programação JAVA.....	21
Tabela 4: Algoritmo recursivo para a Busca, Inserção ou Remoção de um Nó.....	37
Tabela 5: Algoritmo de movimentação recursiva dos elementos gráficos da árvore de acordo com a altura.....	40
Tabela 6: Algoritmo de deslocamento dos elementos gráficos de um nó .....	41
Tabela 7: Trecho exemplo do algoritmo da remoção do algoritmo na Tabela 4 de Busca. ....	43
Tabela 8: Algoritmo de remoção de um nó.....	44
Tabela 9: Algoritmo de cálculo da altura da árvore. ....	46
Tabela 10: Algoritmo de balanceamento da árvore.....	46
Tabela 11: Algoritmo de cálculo do índice e balanceamento de um nó.....	47
Tabela 12: Algoritmo de rotação do nó.....	48
Tabela 13: Algoritmo de Percurso e impressão da árvore.....	52
Tabela 14: Algoritmo para realçar o nó através da cor da classe especificada. ....	53
Tabela 15: Função de espera que controla a velocidade do programa. ....	54
Tabela 16: Trecho em HTML do arquivo arvore.html.....	55
Tabela 17: Trecho em CSS do arquivo estilo-arvore.css. ....	55

## **LISTA DE ABREVIATURAS E SIGLAS**

AVL – Árvore binária de busca balanceada

CSS – Cascade Style Sheets

FAETERJ – Faculdade de Educação Tecnológica do Estado do Rio de Janeiro

FIFO – First in, First out

HTML – HyperText Markup Language

INEP – Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira

LIFO – Last in, First out

SYM – Structure Your Mind

TCC – Trabalho de Conclusão de Curso

UFPE – Universidade Federal de Pernambuco

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>13</b>
<b>2</b>	<b>METODOLOGIAS ATIVAS DE ENSINO.....</b>	<b>16</b>
2.1	IMPORTÂNCIA DAS METODOLOGIAS ATIVAS NA COMPUTAÇÃO .....	16
2.2	JOGOS .....	17
2.3	SIMULADORES .....	19
<b>3</b>	<b>CONCEITOS DAS ESTRUTURAS DE DADOS .....</b>	<b>21</b>
3.1	ESTRUTURAS ESTÁTICAS.....	21
3.1.1	<i>Tipos de dados.....</i>	21
3.1.2	<i>Vetores.....</i>	22
3.1.3	<i>Matrizes.....</i>	22
3.2	ESTRUTURAS DINÂMICAS .....	23
3.2.1	<i>Pilhas.....</i>	23
3.2.2	<i>Filas.....</i>	24
3.2.3	<i>Listas encadeadas.....</i>	25
3.2.4	<i>Árvores binárias .....</i>	26
<b>4</b>	<b>O SIMULADOR SYM .....</b>	<b>30</b>
<b>5</b>	<b>IMPLEMENTAÇÃO DE ÁRVORES BINÁRIAS NO SYM .....</b>	<b>33</b>
5.1	FUNCIONALIDADES DOS NOVOS MÓDULOS DO SYM .....	35
5.1.1	<i>Criar árvore.....</i>	36
5.1.2	<i>Pesquisa pelo valor de nó.....</i>	37
5.1.3	<i>Inserção de nó .....</i>	39
5.1.4	<i>Remoção de um nó.....</i>	42
5.1.5	<i>Altura da árvore .....</i>	46
5.1.6	<i>Balanceamento da árvore.....</i>	46
5.1.7	<i>Rotação da árvore .....</i>	48
5.1.8	<i>Impressão da árvore .....</i>	52
5.1.9	<i>Controle da velocidade.....</i>	53
5.2	TECNOLOGIAS UTILIZADAS .....	54
5.2.1	<i>HTML .....</i>	54
5.2.2	<i>CSS .....</i>	55
5.2.3	<i>Javascript .....</i>	56
5.2.4	<i>JQuery .....</i>	56
5.2.5	<i>SVG.....</i>	56
<b>6</b>	<b>CONCLUSÃO.....</b>	<b>58</b>
	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>60</b>

# 1 INTRODUÇÃO

Cada vez é possível observar mais alunos inscritos anualmente em cursos de graduação. Somente de 2007 a 2017 houve um aumento de aproximadamente 52% alunos a mais inscritos no ano, segundo o INEP (2017, p.14). Embora o número de alunos que se matriculam anualmente em cursos de graduação tenha aumentado significativamente, foi observado, segundo VELASCO (2018, p.01), que apenas 44% dos alunos matriculados em 2010 concluíram o curso ou continuavam no mesmo, enquanto o 56% restante abandonaram o curso.

Os estudos feitos pela autora ainda apontam que o percentual de desistências não possui grande oscilação entre os diferentes cursos, nem oscila entre os primeiros três anos do curso, indicando que os motivos para desistência, não esclarecidos pelo censo do INEP, independem do curso e se prolongam através da sua duração.

Tabela 1: Percentual de desistência por curso

Fonte: VELASCO (2018)

Curso	Percentual de desistência (%)
Administração	61,5
Direito	54,2
Pedagogia	49,5
Ciências contábeis	55,8
Enfermagem	51,7
Gestão de pessoal	53,1
Serviço social	57,4
Engenharia civil	54,4
Psicologia	52,8
Educação Física	57,3

Tabela 2: Percentual de desistência por ano de curso (matriculados em 2010)

Fonte: VELASCO (2018)

Ano	Percentual de desistência (%)
1º ano (2010)	10,9
2º ano (2011)	16,7
3º ano (2012)	10,7
4º ano (2013)	7,2
5º ano (2014)	6,6
6º ano (2015)	3,5

É importante reconhecer que embora exista um aumento substancial no número de alunos ingressantes nos cursos de graduação de ensino superior ao longo dos anos, a grande maioria desses alunos acabam por desistir do curso. Um estudo levantado pela Universidade Federal de Pernambuco (UFPE) (2016, p.21), indica os principais motivos que os levaram à evasão do curso de graduação na universidade, separando-os entre fatores internos e externos.

Os fatores externos se classificam como aqueles onde a universidade não contém jurisdição, como problemas de saúde, familiares, financeiros, dificuldade de locomoção ou no mercado de trabalho. Já os fatores internos se referem àqueles onde a universidade precisa dar mais atenção, pois é de sua responsabilidade. O principal fator interno relacionado à evasão foi o de falta de formação pedagógica, ou desestímulo dos docentes, quando 71,43% dos alunos entrevistados apontaram esta causa como a principal para a desistência do curso na universidade. O estudo trata apenas dos alunos da universidade, o que não é suficiente para ter compreender em totalidade os motivos que levam mais de 50% dos alunos no Brasil a não concluir o curso de graduação, entretanto, podemos observar que a dificuldade do aluno de assimilar o conteúdo pode levar à sua desistência pelo curso.

Entende-se de significada importância o uso de metodologias que possam auxiliar no ensino, para evitar o desestímulo tanto por parte dos professores quanto dos alunos. Atualmente existem metodologias ativas de apoio ao ensino que serão apresentadas posteriormente, no capítulo 2.

Na graduação em áreas da computação, um tema abordado que traz dificuldade é o de estrutura de dados, conforme levantamento pelo estudo publicado por LIMA JUNIOR, VIEIRA e VIEIRA (2015, p.11), para o curso de Sistemas de Informação, pela Faculdade de Educação Tecnológica do Estado do Rio de Janeiro (FAETERJ), 54% dos alunos foram reprovados na aprendizagem de uma disciplina inicial de algoritmos e programação. Segundo LIMA JUNIOR, VIEIRA e VIEIRA (2015, p.12), o conteúdo que mais apresentou dificuldade para os alunos foi o de estruturas de dados homogêneas (vetores e matrizes) quando 57% deles reportaram ter apresentado dificuldade nesse conteúdo.

É possível verificar que mesmo nas disciplinas iniciais as estruturas de dados mais simples, como vetores e matrizes já apresentaram maior dificuldade de aprendizado para os alunos. Estruturas de dados mais complexas como listas, pilhas, filas e árvores tendem a evidenciar mais esta dificuldade.

Este trabalho tem como objetivo trazer um apoio ao ensino no aprendizado de algoritmos e estruturas de dados para os cursos relacionados à computação, como Ciência da

Computação, Sistemas de Informação, Engenharia da Computação, entre outros, de forma a empregar o uso de metodologias ativas nas disciplinas, aumentando o interesse do aluno pelo conteúdo e facilitando a sua aprendizagem, de forma a evitar que a dificuldade de compreensão, frustração e reprovação possam ser eventuais causas para a evasão do curso por parte dos alunos.

Este trabalho será dividido em seis capítulos. No capítulo 2, serão apresentadas as tecnologias atuais e metodologias ativas na educação na área da computação.

No capítulo 3, serão explicados os conceitos e o funcionamento das estruturas de dados, como as pilhas, listas encadeadas e as árvores binárias.

No capítulo 4, será apresentado o sistema Structure Your Mind (SYM), desenvolvido por SOUZA (2019), o qual apresenta o apoio ao ensino das estruturas de dados como pilhas, listas encadeadas e filas, sistema no qual este trabalho foi baseado para dar continuidade na implementação das estruturas de dados de árvores binárias simples e árvores binárias de busca balanceada (AVL). De forma a deixar o sistema mais completo para que possa ser utilizado no auxílio de disciplinas cuja ementa apresente estas estruturas de dados.

No capítulo 5 serão apresentadas as mudanças e funcionalidades no sistema que este trabalho implementou para o funcionamento das árvores binárias, serão apresentados os principais algoritmos com ilustrações mostrando como se comportam no sistema.

No capítulo 6 as conclusões acerca deste trabalho serão apresentadas, e serão discutidas as propostas para trabalhos futuros.

## **2 METODOLOGIAS ATIVAS DE ENSINO**

MORÁN (2016, p.15), define as mudanças na educação como suaves ou profundas, onde a primeira mantém o modelo curricular, mas envolve de forma maior a participação do aluno em sala de aula, através do ensino híbrido, que mistura o aprendizado presencial e a complementação virtual, o aluno sozinho, ou a sala invertida, onde o aluno é responsável por aprender os conteúdos em casa e posteriormente se encontrará na sala de aula apenas para realizar atividades e exercícios onde será possível ao aluno tirar suas dúvidas, e concomitantemente auxiliar a fixar o conteúdo.

As mudanças profundas definidas pelo autor definem modelos de ensino sem disciplinas, onde os ensinamentos são feitos por professores orientadores que baseiam o ensino em atividades e desafios, onde é possível ao aluno, que participará de grupos e projetos, aprender a solucionar esses problemas e superar obstáculos enquanto desenvolve as habilidades necessárias para o domínio do conteúdo.

### **2.1 IMPORTÂNCIA DAS METODOLOGIAS ATIVAS NA COMPUTAÇÃO**

Para SILVA e FONSECA (2017, p.25) a utilização da tecnologia no processo de ensino é de imensa importância para a expansão de oportunidades na difusão de conhecimentos, pois o acesso aos conteúdos feito pelo educador observa inúmeras opções para a abordagem do conteúdo e de acesso à informação.

A área da computação avança constantemente, e é frequente o surgimento de novas tecnologias, metodologias de trabalho ou frameworks. Um framework é uma abstração de múltiplos códigos de diversos projetos de software, que permite a sua reutilização prática, aumentando a produtividade dos desenvolvedores através do aumento de nível de abstração. É importante que os profissionais de computação estejam sempre atualizados não só para poder participar da competência do mercado de trabalho, mas também para garantir que seu trabalho possui a maior qualidade e produtividade que a tecnologia permite no momento.

Para MORÁN (2016, p.17), as metodologias ativas mais profundas abolem as classificações dos conteúdos em disciplinas, por entender que o conhecimento é, essencialmente, interdisciplinar. CUNHA, et. al. (2008, p.02), estudam a relevância da interdisciplinaridade no ensino de Engenharia de Software, onde apontam como um projeto de Engenharia requer do conhecimento de diversas disciplinas, como Banco de Dados, Qualidade, Confiabilidade e Segurança, Sistemas Embarcados de Tempo Real, Tecnologias da Informação, Teste de Software e Engenharia de Software.



O entendimento das regras do negócio é um fator chave no desenvolvimento de um software, entendem DALLAVALLE e CAZARINI (2000, p.01), cabendo ao desenvolvedor entender os componentes e requisitos do sistema de informação organizacional para que o sistema possa desempenhar as tarefas objetivas. Estas regras de negócio representam a forma da interdisciplinaridade em um projeto de software, indo além das disciplinas encontradas na computação. Para cada área de aplicabilidade do software, é necessário que os engenheiros de software ou desenvolvedores entendam, de forma clara e precisa as regras de negócio que serão aplicadas. Como exemplo, em softwares financeiros e contábeis, é importante que os mesmos entendam as regras básicas de economia e contabilidade, em um sistema de vendas de seguros é importante que os profissionais de computação entendam o motor de regras de aceitação ou recusa automática de propostas.

Portanto é imprescindível que o profissional da computação possa ser familiarizado com as metodologias ativas e de ensino individual, para que seja capaz, sempre que for necessário, da atualização e aprendizado de novos conhecimentos.

## **2.2 JOGOS**

Os jogos durante as aulas são enfatizados por MORÁN (2016, p.18) por cada vez mais estarem presentes no âmbito escolar, devido a que com o passar das gerações, a grande maioria de novos alunos estão acostumados aos jogos e a superação de desafios que eles brindam. O autor define esta metodologia como bastante atraente para os novos alunos, pois incentiva através da competição, da cooperação e das recompensas à realização das diversas atividades propostas nas aulas.

Na Figura 1, é apresentado o jogo proposto por NETTO et. al. (2017, p.01), o propósito deste é incentivar o aprendizado da lógica da programação através de um jogo disponível para dispositivos móveis. KAPP (2012, p.09) define a gamificação (do inglês gamification) como a formação da educação através de estratégias baseadas em jogos, cujo objetivo principal é do aprendizado como consequência do engajamento e interesse do aluno no jogo. Histórias e narrativas são utilizadas para guiar os alunos a se comportar conforme o esperado dentro de um contexto. Desafios e níveis, que permitem que o progresso do aluno seja feito de forma gradativa, permitindo que o aluno só avance na complexidade do conteúdo quando conseguir atingir os objetivos propostos pelo nível anterior, de forma a evitar que a dificuldade seja demasiado elevada e dessa forma, o aluno não seria passível de frustração que poderia leva-lo à evasão. Os pontos e rankings servem como propósito principal incentivar a competição entre os colegas de aprendizado, aumentando o interesse pela progressão no jogo.



Figura 1: Apresentação do GameLogic, jogo para aprendizado da lógica de programação  
 Fonte: NETTO et. al. (2017)

O jogo apresentado por NETTO et. al. (2017, p.07) introduz os comandos principais da lógica programação e estrutura de algoritmos, como os sequenciais, condição de decisão e laços de repetição, é possível verificar na Figura 2 que estes comandos se encontram na parte inferior da tela, enquanto na parte lateral direita se encontra a formação da solução que o usuário dará através da combinação dos outros comandos, formando o algoritmo que solucionará o problema.



Figura 2: Comandos do GameLogic e formação do algoritmo solução  
 Fonte: NETTO et. al. (2017)

Os resultados encontrados por NETTO et. al. (2017, p.08) identificaram que com uma amostra de 10 alunos, 100% se sentiram motivadas a continuar jogando, ainda com um deles

afirmando que seria tão viciante quanto os outros jogos que está acostumado a jogar no seu dispositivo móvel, um resultado positivo que atinge o objetivo de engajamento proposto pela escolha da metodologia ativa.

## 2.3 SIMULADORES

Os simuladores também são excelentes ferramentas das metodologias ativas de ensino, possibilitando a criação de ambientes e eventos similares aos reais, onde o aluno se depara com situações do cotidiano da profissão, permitindo que o aluno adquira conhecimento de forma ativa, dos conceitos e habilidades necessárias para resolução de um problema, segundo COSTA et. al. (2015, p.61). Dessa forma, pode-se utilizar três dos principais métodos ativos de aprendizagem, o Aprendizado Baseado em Problemas, o Aprendizado Baseado em Projetos e o Estudo de Caso, GEIB (2017, p.37).

De acordo com o estudo realizado por SANTOS (2016, p.50), um dos simuladores mais populares com fins acadêmicos foi desenvolvido pela própria Cisco Systems Inc., gigante das soluções de redes de computadores. A figura 3 apresenta o Cisco Packet Tracer, uma ferramenta que permite o aprendizado através da simulação de conexão dos mais diversos dispositivos através da rede como: switches e roteadores, equipamentos sem fio, variados tipos de servidores, dispositivos da Internet of Things (interfaces inteligentes em infraestrutura global onde os dispositivos físicos e virtuais adquirem identidades e personalidades virtuais, interconectados através da internet, (CERP, 2009)).

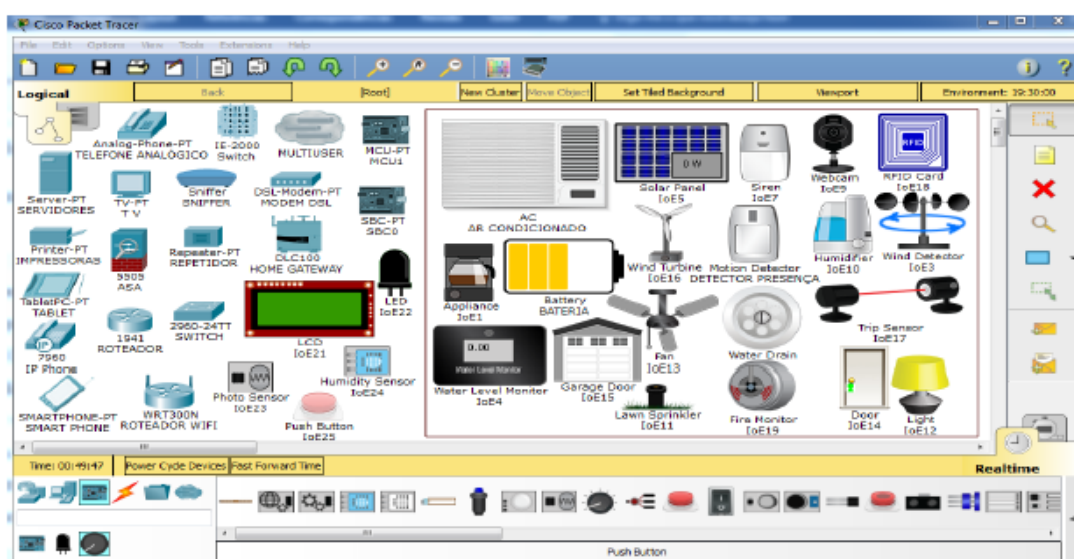


Figura 3: Cisco Packet Tracer

Fonte: SANTOS (2016)

O Cisco Packet Tracer permite ainda, a simulação de envio e roteamento de pacotes através de diferentes protocolos, como é possível verificar na figura 4, os protocolos são escolhidos pelo aluno, em redes locais (LAN) ou de longa distância (WAN) através de dois hospedeiros diferentes. A ferramenta permite que o aluno desenvolva os projetos de simulação de forma individuais ou com outros usuários, que permitem o teste de múltiplos projetos em tempo real.

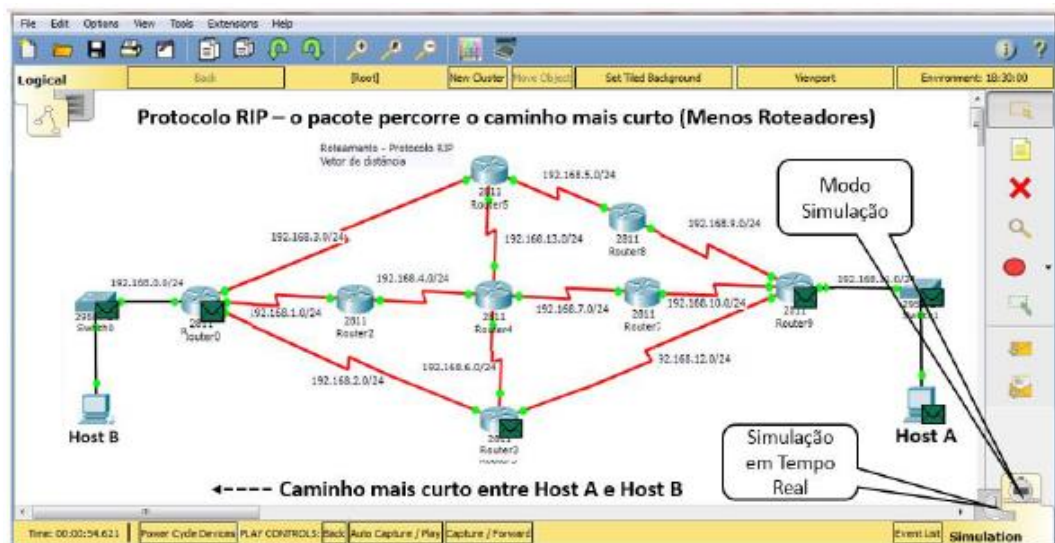


Figura 4: Roteamento de pacotes no Cisco Packet Tracer

Fonte: SANTOS (2016)

SANTOS (2016, p.51), descreve as diferentes formas como o aluno pode se aproveitar do aprendizado através da simulação, configurando e implementando topologias de redes complexas, a transmissão e acompanhamento de pacotes criados por ele, ou sobre o comando do professor, a ferramenta serve como auxílio a visualização do conteúdo e de exemplos complexos desenhados por ele.

Através do estudo apresentado por VOSS, et. al. (2012, p.09), foi possível concluir, em meio à análise de laboratórios virtuais para o ensino da disciplina de Redes de Computadores que a ferramenta Cisco Packet Tracer era bastante indicada para servir de auxílio à fixação do conteúdo ministrado na disciplina em nível de graduação.

### 3 CONCEITOS DAS ESTRUTURAS DE DADOS

#### 3.1 ESTRUTURAS ESTÁTICAS

CELES et. al. (2016, p.02) definem as estruturas de dados estáticas como estruturas que não se encontram programadas para que possam ser feitas inserções ou remoções de forma dinâmica e adequadamente. Para sua formação são utilizadas as formas primitivas e estruturas providas pela linguagem, como os vetores, tipos de dados.

##### 3.1.1 Tipos de dados

Os tipos de dados primitivos são aqueles com os quais serão manipuladas estruturas a fim de criar algoritmos e estruturas de dados mais complexas, representadas por esses tipos de dados. ASCENCIO e CAMPOS (2002, p. 08), apresentam os tipos de dados primitivos: os dados numéricos que se dividem entre inteiros e reais, a diferença entre os dois é dada através da posse de parte fracionária. Os dados lógicos ou booleanos que representam os valores de verdadeiro ou falso. E os dados literais ou caracteres, representados por um conjunto de caracteres que podem ser letras, números sem finalidade de cálculo ou caracteres especiais.

Os tipos de dados são associados às variáveis ou constantes, que por sua vez são os recursos para armazenamento de dados para um determinado algoritmo. As constantes são dados que não sofrem nenhuma alteração até o fim do algoritmo, enquanto as variáveis podem as variáveis podem sofrer alterações em inúmeros instantes durante a execução do algoritmo (FORBELLONE e EBESPÄCHER, 2005, p.16). Algumas linguagens de programação classificam os tipos de dados primitivos de acordo a otimizar o espaço de memória que será necessário para armazenar os valores de uma variável daquele tipo de dados. Como exemplo, a linguagem JAVA possui divisões para os dados primitivos, conforme a Tabela 3.

Tabela 3: Tipos de dados primitivos na linguagem de programação JAVA

Fonte: ASCENCIO e CAMPOS (2002)

Tipo (JAVA)	Tipo de dado primitivo	Faixa de valores	Tamanho (aproximado)
byte	Inteiro	-128 a 127	8 bits
char	Caractere	0 a 65.535	16 bits
short	Inteiro	-32.768 a 32.767	16 bits
int	Inteiro	-2.147.483.648 a 2.147.483.647	32 bits
long	Inteiro	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	64 bits

float	Real	$-3.4 \times 10^{-38}$ a $3.4 \times 10^{38}$	32 bits
double	Real	$-1.7 \times 10^{-308}$ a $1.7 \times 10^{308}$	64 bits
boolean	booleano	true ou false	Indefinido

### 3.1.2 Vetores

Os vetores são, segundo LAUREANO (2008, p.02), uma estrutura de dados linear que pode ser endereçado a partir de um único índice. Os valores que são armazenados em lista são do mesmo tipo, por ser uma estrutura estática, é definido como tendo um número fixo de células. Sua estrutura é homogênea por conter dados de apenas um tipo. A alocação na memória é sequencial, conforme a Figura 5, de forma que não podem ser inseridos novos elementos além do fixo definido, e se um elemento é excluído, a posição da memória não é liberada.

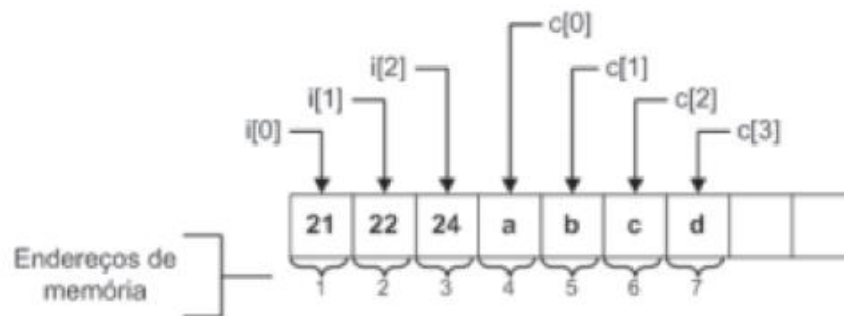


Figura 5: Representação de um vetor na memória

Fonte: LAUREANO (2008)

### 3.1.3 Matrizes

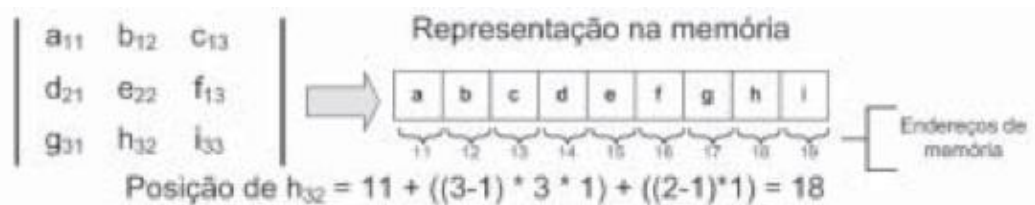


Figura 6: Representação linear de uma matriz bidimensional na memória

Fonte: LAUREANO (2008)

As matrizes são descritas por LAUREANO (2008, p.05) como arranjos bidimensionais ou multidimensionais cuja alocação em memória, assim como nos vetores, é estática e sequencial. Para poder fazer referência a um dado dentro da matriz é necessário de um número  $n$  de índices igual ao número de dimensões. Assim como o vetor, são estruturas de

dados homogêneas, portanto todos os elementos são do mesmo tipo. Os elementos são alocados de forma contígua na memória, por exemplo, uma matriz com 3 linhas e 3 colunas possui 9 elementos com armazenamento linear, conforme ilustrado na Figura 6.

### 3.2 ESTRUTURAS DINÂMICAS

As estruturas dinâmicas são definidas por CELES et. al. (2016, p.199) de maneira oposta como estruturas que foram construídas para oferecer as funcionalidades de inserção e remoção de elementos de forma dinâmica, seu número de elementos não está limitado de forma inicial e não é necessário ocupar uma grande porção da memória sem a estar utilizando.

Segundo LAUREANO (2008, p.22), diversas linguagens de programação permitem que a memória seja manipulada dinamicamente, alocando quando necessário e na remoção, liberando a área de memória. Porém, de acordo com o nível de abstração, algumas linguagens como Java possibilitam que este processo aconteça sem maior interferência do desenvolvedor, enquanto em linguagens como o C exige que o desenvolvedor efetue previamente a alocação e liberação para utilização da memória.

#### 3.2.1 Pilhas

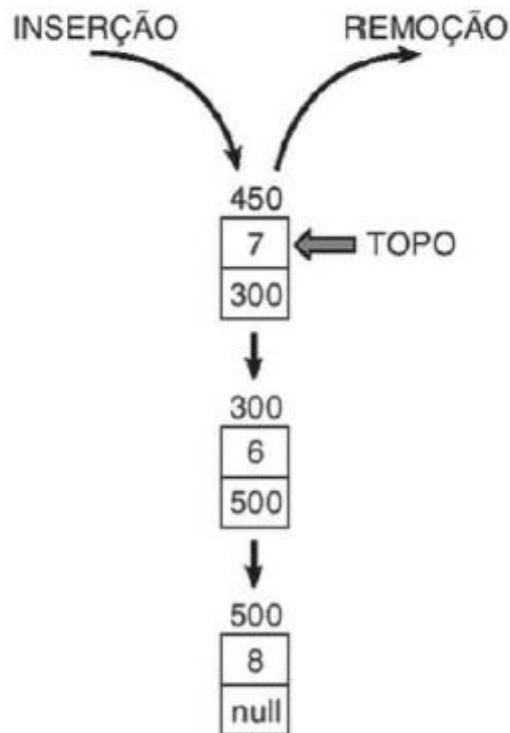


Figura 7: Estrutura de uma pilha

Fonte: ASCENCIO e ARAÚJO (2010)

As pilhas são definidas por LAUREANO (2008, p.41) como listas lineares ordenadas de itens, podendo ocorrer a inserção de novos elementos, que serão inseridos na extremidade chamada como topo da pilha, como pode ser verificado na Figura 7. Permite também a remoção através dessa extremidade, por isso apresenta a característica definida como LIFO (Last In, First Out), onde o primeiro elemento é o último a sair, pois fica na base da pilha, e analogamente o último a entrar, é o primeiro a sair, pois fica no topo da mesma. Por serem estruturas dinâmicas, é possível alocar o espaço de memória apenas no instante anterior à inserção do elemento, e a liberação de memória após a remoção de um elemento.

### 3.2.2 Filas

LAUREANO (2008, p.48) define as filas como listas lineares ordenadas de itens, assim como as pilhas, sendo o princípio de organização o grande diferencial entre essas duas estruturas. Enquanto a pilha é uma estrutura com princípio de LIFO, a fila possui o princípio de FIFO (First in, first out), onde o primeiro elemento é o primeiro a sair, analogamente o último a entrar é o último a sair, desta forma, agora existem duas extremidades acessíveis às funções de manipulação da fila, onde a inserção insere pela extremidade de fim da fila, e a remoção retira pela extremidade de início da mesma. É necessário que existam dois índices de apontamentos para estas extremidades.

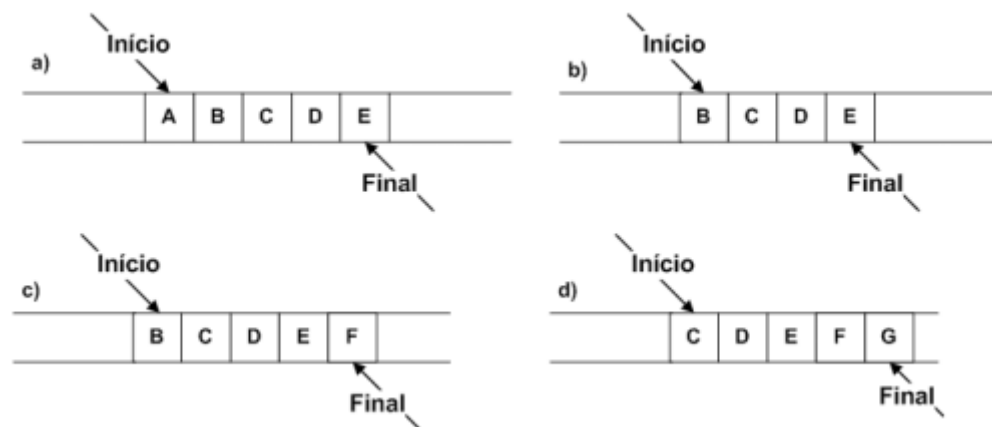


Figura 8: Exemplo de operações em uma fila  
Fonte: LAUREANO (2008)

É possível perceber na Figura 8 que a operação entre o quadro a e b foi uma remoção, retirando o elemento “A” do início, a operação entre o quadro b,c e d foi de inserção dos elementos “F” e “G”, inseridos no final da fila.



### 3.2.3 Listas encadeadas

São estruturas de dados similares às pilhas ou filas, com alocação dinâmica de memória, segundo LAUREANO (2008, p.79) cada elemento da fila está ligado ao próximo elemento da lista, permitindo que elementos possam ser inseridos ou removidos em qualquer posição da lista, como exemplo, para remover um item na posição  $n$  bastaria apenas fazer com que o elemento na posição  $n-1$  aponte para a posição  $n+1$  e posteriormente remover o item da posição  $n$ . Desta forma a lista continuaria ordenada e as possibilidades de manipulação de dados são muito maiores que nas duas estruturas anteriores.

De acordo com ASCENSIO e ARAÚJO (2010, p. 106), uma lista encadeada onde apenas existe uma ligação entre os elementos, para o próximo elemento é chamada de lista simplesmente encadeada.



Figura 9: Lista simplesmente encadeada

Fonte: LAUREANO (2008)

ASCENSIO e ARAÚJO (2010, p.130), definem a lista duplamente encadeada como aquela onde seus elementos possuem a ligação para o próximo elemento e também para o elemento anterior. Esta estrutura permite que a navegação entre os elementos da lista seja feita de forma otimizada, dispensando a preocupação com elementos auxiliares para as posições anteriores da fila durante os laços de interações nos algoritmos.

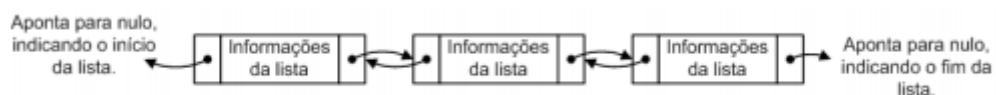


Figura 10: Lista duplamente encadeada

Fonte: LAUREANO (2008)

Como é possível observar nas Figuras 9 e 10, nas extremidades da lista, quando o apontamento é feito para um elemento inexistente ou vazio, as figuras indicam que o elemento aponta para nulo. Implicando que na implantação dos algoritmos para percorrer a lista,

quando o algoritmo atingir o elemento que estiver apontando para o nulo, o algoritmo saberá que chegou à extremidade da lista (LAUREANO, 2008, p.85).

### 3.2.4 Árvores binárias

Segundo LAUREANO (2008, p.126), uma árvore binária ou se encontra vazia ou possui três subconjuntos, a raiz da árvore, a subárvore da esquerda e a subárvore da direita, estas subárvores quando isoladas compõem outras árvores. Na Figura 11 é possível observar um exemplo de árvore binária.

Todo elemento de uma árvore pode ser chamado de nó, e possui uma relação de hierarquia. Os nós que sucedem um nó anterior são chamados de filhos, e este de pai. Se o nó não possui nenhum filho recebe a declaração de folha (FORBELLONE e EBERSPACHER, 2005, p.169).

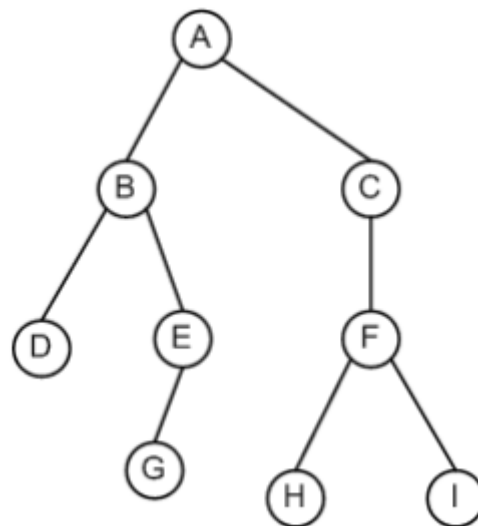


Figura 11: Árvore binária  
Fonte: LAUREANO (2008)

Conforme LAUREANO (2008, p.129), uma árvore de busca binária, é aquela onde os valores da subárvore esquerda são menores que o valor chave do nó, e os valores da subárvore direita são maiores. São organizados com este objetivo para otimizar consideravelmente a busca de elementos. Não existe uma única forma de organizar o conjunto de informações, portanto, a partir de um conjunto de valores, podemos apresentar árvores distintas, como na Figura 12.

LAUREANO (2008, p.130), define a pesquisa por um valor específico começando pela raiz, sendo o valor igual à raiz, o valor existe, sendo menor o processo será repetido

recursivamente na subárvore esquerda, e sendo o valor maior, na subárvore direita. O autor define a inserção em uma árvore binária com a busca do valor a ser inserido pela árvore, o valor somente pode ser inserido se a folha for alcançada e o elemento não existir. O nó é inserido na subárvore direita se for maior à folha e na subárvore esquerda se for menor.

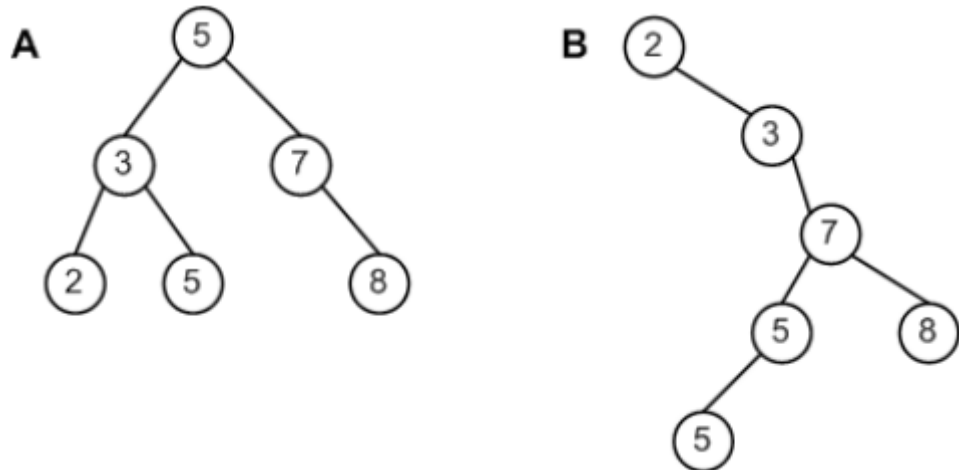


Figura 12: Árvores distintas para um mesmo conjunto de valores  
Fonte: LAUREANO (2008)

LAUREANO (2008, p.133) define a exclusão como um caso entre três distintos: Caso uma folha venha a ser excluída, basta remover o nó da árvore. Caso o nó a ser removido possua um único filho, o pai deste nó herda a subárvore filha do nó a ser removido, e o nó é removido. Caso um nó com dois filhos venha a ser excluído, o substituto é encontrado através do filho esquerdo do nó, e posteriormente navegar até a folha mais à direita, caso o filho esquerdo do nó não possua filhos à direita, este será o nó que substitui o nó a ser removido, conforme a Figura 13.

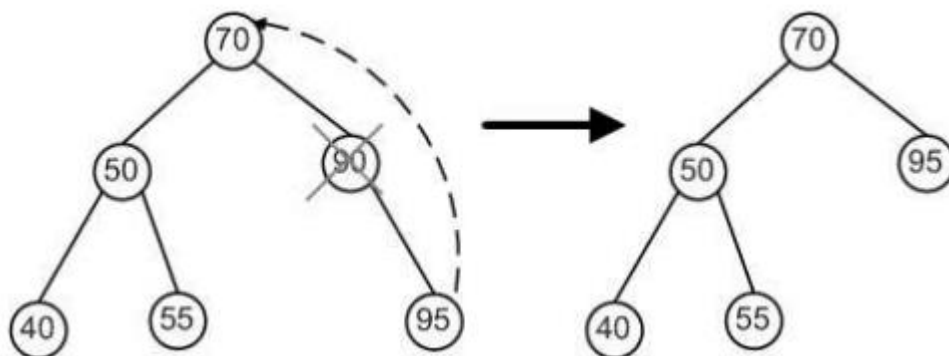


Figura 13: Exclusão e substituição de um nó  
Fonte: LAUREANO (2008)

TSAKALIDIS (1985, p.174), define uma árvore AVL como uma árvore binária de busca se as alturas de ambas as subárvores esquerda e direita diferem no máximo por uma unidade, definindo a altura como a maior distância entre um nó e a folha em sua subárvore. O índice de balanceamento é definido como esta diferença entre a altura de ambos os filhos do nó, e não pode ser diferente de +1, 0 ou -1. TSAKALIDIS (1985, p.175) aponta o nó crítico como o último nó desbalanceado no caminho de busca. Portanto, na existência de um nó crítico, a árvore precisa ser balanceada.

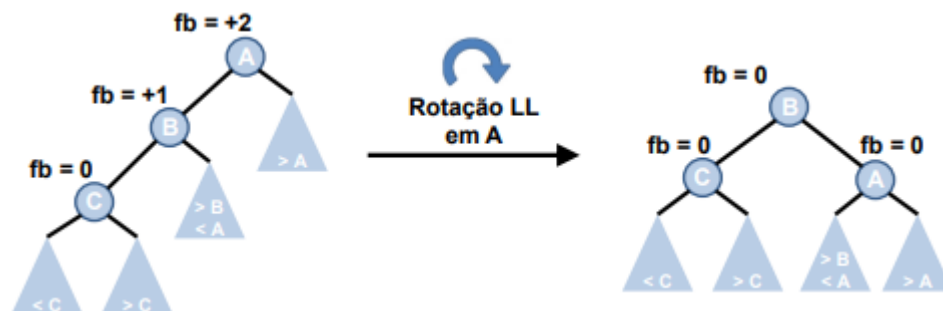


Figura 14: Rotação LL  
Fonte: BACKES (2019)

As rotações LL e RR são simétricas, assim como LR e RL também. Estas rotações são dadas pelas fórmulas abaixo, considerando um nó N como o último desbalanceado da árvore, e três variáveis auxiliares nó A, nó B e nó C. E o Nó A recebe o nó N.

Rotação LL, conforme a Figura 14:

- Nó B recebe a subárvore esquerda de A.
- Nó C recebe a subárvore esquerda de B.
- O filho esquerdo de A recebe o filho direito de B.
- O filho esquerdo de B recebe o Nó C.
- O Filho direito de B recebe o Nó A.
- O nó N recebe o nó B, sobrescrevendo a árvore e completando a rotação.

Rotação RR:

- Nó B recebe a subárvore direita de A.
- Nó C recebe a subárvore direita de B.
- O filho direito do nó A recebe o filho esquerdo do nó B.
- O filho direito do nó B recebe o nó C.
- O filho esquerdo do nó B recebe o nó A.

- O nó N recebe o nó B, sobrescrevendo a árvore e completando a rotação.

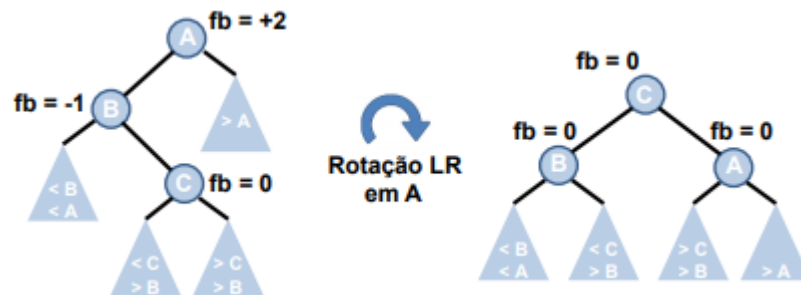


Figura 15: Rotação LR em árvore AVL

Fonte: BACKES (2019)

#### Rotação LR:

- Nó B recebe a subárvore esquerda de A.
- Nó C recebe a subárvore direita de B.
- O filho esquerdo de A recebe o filho direito de C.
- O filho direito de B recebe o filho esquerdo de C.
- O filho esquerdo de C recebe o nó B.
- O filho direito de C recebe o nó A.
- O nó N recebe o nó C, sobrescrevendo a árvore e completando a rotação.

#### Rotação RL:

- Nó B recebe a subárvore direita de A.
- Nó C recebe a subárvore esquerda de B.
- O filho direito de A recebe o filho esquerdo de C.
- O filho esquerdo de B recebe o filho direito de C.
- O filho esquerdo de C recebe o nó A.
- O filho direito de C recebe o nó B.
- O nó N recebe o nó C, sobrescrevendo a árvore e completando a rotação.

## 4 O SIMULADOR SYM

Este trabalho tem como objetivo implementar as estruturas de dados de árvores binárias de busca e de altura balanceada no simulador SYM, uma ferramenta de metodologia ativa de ensino que permite o auxílio ao ensino das estruturas de dados, dos tipos listas encadeadas, pilhas e filas, conteúdos da disciplina de Estrutura de Dados na graduação. O Programa ainda permite que seja visualizado uma simulação da depuração do código em C, o que facilita a compreensão (SOUZA, 2019, p.35).

É uma ferramenta web, que não necessita de instalação de programas adicionais, pode ser hospedada ou utilizada de forma off-line, por todas suas funcionalidades trabalharem no cliente-side (lado do cliente).

O SYM possui três divisões de tela, apresentadas na Figura 16. Na tela lateral esquerda é possível verificar a depuração do código em C, esta é apenas uma simulação, já que a ferramenta foi desenvolvida com javascript, porém, foi escolhida a linguagem C para a depuração por ser a linguagem de comum utilização para o ensino da disciplina. Na tela do meio são posicionados os elementos gráficos, estes elementos se movimentam de forma a dar um melhor entendimento ao usuário sobre como funcionam as operações de inserção, busca ou remoção. Na lateral direita, é possível verificar uma simulação da área de memória, e em quais blocos são alocados dinamicamente para armazenar os ponteiros e valores dos elementos.

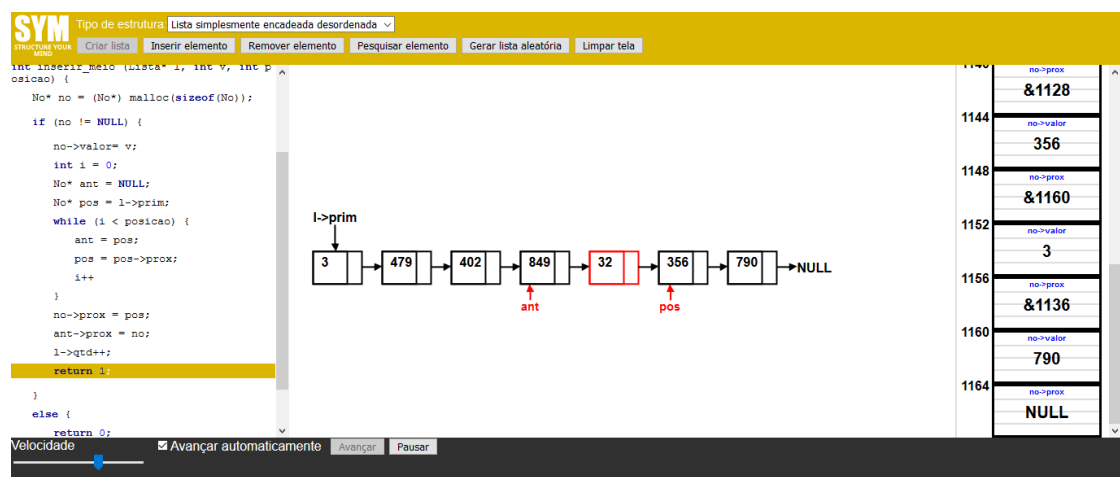


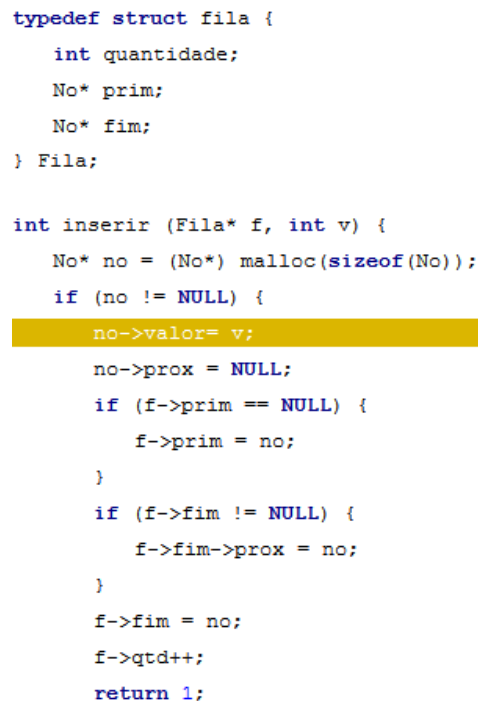
Figura 16: Tela do SYM em funcionamento

Fonte: SOUZA (2019)

Na parte superior da tela estão localizados os comandos de manipulação das estruturas de dados, podendo escolher o tipo de estrutura, criar uma nova lista, inserir, remover ou pesquisar um elemento, gerar uma lista com valores aleatórios ou limpar a tela.

Na barra inferior da tela são apresentados botões para controle da velocidade com as quais as animações são executadas, bem como controlar o tempo dos processos para a execução das operações. Por exemplo, na inserção de um elemento, é possível avançar no passo a passo desde que o novo ponteiro para o elemento é criado, durante o momento onde a busca está sendo realizada dentro da lista ordenada, até o momento da inserção no lugar correto da lista.

SOUZA (2019, p.35) cita ainda que a motivação para o trabalho foi devida à pesquisa de múltiplos simuladores com o mesmo objetivo, e embora tenham sido encontrados bons projetos como o VISUALGO desenvolvido pelo professor Steven Halim da Universidade da Singapura, e o DATA STRUCTURE VISUALIZATIONS, desenvolvido pelo professor David Galles, da Universidade de San Francisco, nenhum destes simuladores encontrados apresentava a simulação de depuração de código em C, ou que não fosse pseudocódigo (SOUZA, 2019, p.30). Na Figura 17, observa-se que esta funcionalidade foi implementada no SYM.



```
typedef struct fila {
    int quantidade;
    No* prim;
    No* fim;
} Fila;

int inserir (Fila* f, int v) {
    No* no = (No*) malloc(sizeof(No));
    if (no != NULL) {
        no->valor= v;
        no->prox = NULL;
        if (f->prim == NULL) {
            f->prim = no;
        }
        if (f->fim != NULL) {
            f->fim->prox = no;
        }
        f->fim = no;
        f->qtd++;
        return 1;
    }
}
```

Figura 17: Área de código do SYM em depuração

Fonte: SOUZA (2019)

A ferramenta foi apresentada para os alunos das disciplinas de Estruturas de Dados II, na turma do orientador do projeto, foi realizada uma pesquisa de satisfação e a partir dela foi possível identificar que com a metodologia padrão de ensino para a disciplina, 84% dos alunos apresentavam um nível alto ou médio de dificuldade no aprendizado do conteúdo (SOUZA, 2019, p.48). Outra pergunta realizada se referia à opinião sobre a disponibilização de uma ferramenta de apoio ao ensino para o conteúdo da disciplina e sua relevância, aos quais 44% dos alunos se manifestaram a favor, porém desde que fossem orientados pelo professor, e 56% dos alunos se manifestaram também a favor, com o interesse de utilizar a ferramenta de forma individual.

Devido à importância da existência de uma ferramenta de metodologia ativa de apoio ao ensino, discutidos nos capítulos 1 e 2, à ótima funcionalidade apresentada pela ferramenta e o diferencial sobre outras semelhantes, e ao conteúdo de árvores binárias da disciplina ser complexo, e muitos alunos apresentarem dificuldades com os algoritmos recursivos, para este trabalho optou-se por implementar as funcionalidades das estruturas de dados árvores binárias de busca e AVL dentro da ferramenta SYM.



## 5 IMPLEMENTAÇÃO DE ÁRVORES BINÁRIAS NO SYM

Conforme descrito no capítulo 4, o SYM é uma ferramenta de apoio ao ensino da disciplina de Estruturas de Dados nos cursos de graduação, representando o uso de metodologias ativas no objetivo de facilitar o aprendizado e engajar o interesse do aluno pelo conteúdo, aumentando a fixação do conteúdo e diminuindo a evasão da disciplina pelos alunos, SOUZA (2019, p.50).

O SYM implementa em seu funcionamento as pilhas, listas e filas encadeadas, tanto simples quanto duplas, sugerindo para trabalhos futuros a implementação no próprio sistema as funcionalidades já existentes para as estruturas de dados de árvores binárias, que também fazem parte do conteúdo de ensino da disciplina de Estrutura de Dados na graduação, SOUZA(2019, p.53).

Este trabalho tem como objetivo a implementação seguindo o padrão da ferramenta SYM para as estruturas de dados de árvores binárias e árvores AVL. Implementando a busca, inserção, impressão, remoção cálculo de altura e geração de árvore aleatória para ambas as estruturas. Para a árvore AVL, ainda, será necessário a implementação adicional das funcionalidades: cálculo de índice de balanceamento e rotações LL (Left-Left), LR (Left-Right), RL (Right-Left) e RR (Right-Right).

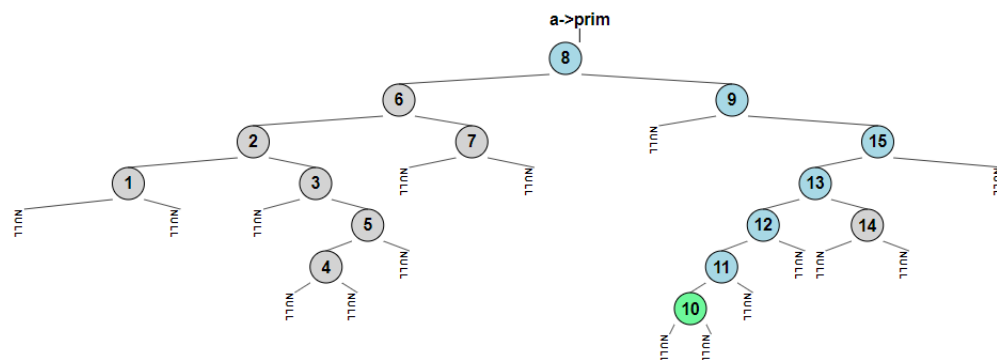


Figura 18: Busca e inserção em árvore binária no SYM

Fonte: Autor

Os elementos das árvores (nós) possuem representações visuais na coluna central da tela, representados como círculos com seu valor circunscrito, apresentando as ligações para o filho esquerdo e o filho direito. De forma a aumentar a inteligibilidade do sistema, os nós serão coloridos de acordo com o tipo de operação que está sendo realizado nele.

O Azul representado na Figura 18 significa que o nó está sendo percorrido pelo algoritmo recursivo de busca. Cinza significa que o nó foi excluído da busca pelo algoritmo recursivo da busca, por exemplo, se o valor a ser encontrado for o número 10 e a iteração do algoritmo recursivo se encontrar buscando no nó de valor 20, a busca seguirá à esquerda de 20, realçando de azul o nó filho esquerdo de 20 (caso não seja nulo) e toda a subárvore direita de 20 será pintada de cinza.

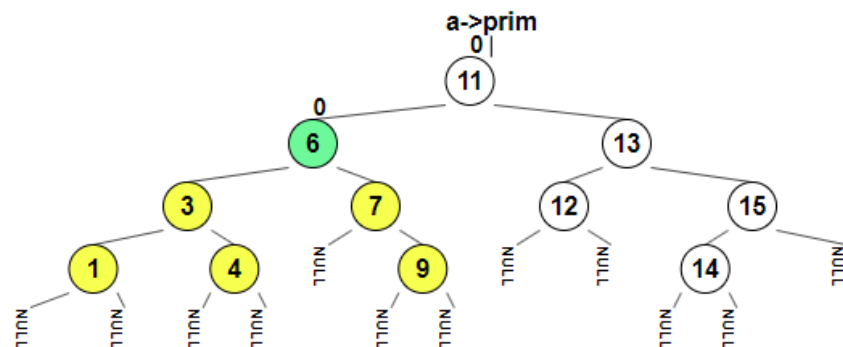


Figura 19: Cálculo de altura e índice de balanceamento de um nó no SYM

Fonte: Autor

Na Figura 19, amarelo significa que o nó está sendo percorrido pelo algoritmo recursivo de cálculo de altura, o qual é utilizando de forma independente, ou neste caso, também durante o cálculo do índice de balanceamento para cada nó para as árvores AVL. Já o verde, significa que o nó foi encontrado e/ou é alvo de uma ação, por exemplo, na inserção o nó inserido seria verde, na busca seria o nó encontrado, no cálculo de altura e balanceamento, o nó cuja altura está sendo calculada e na remoção o nó a ser removido.

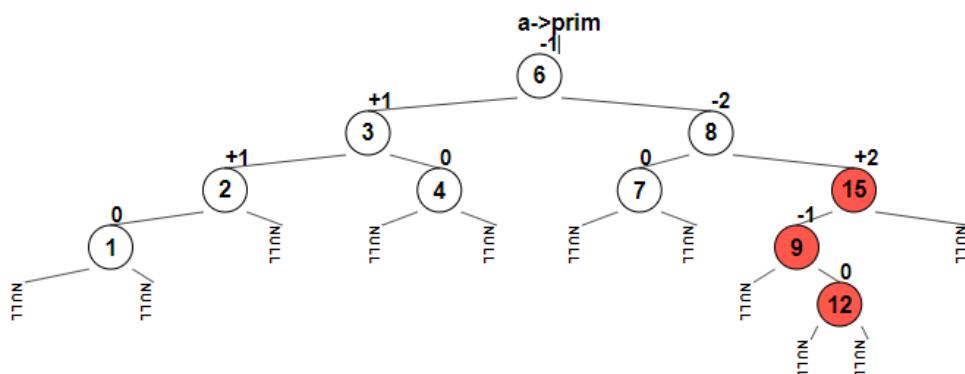


Figura 20: Rotação LR em árvore AVL no SYM

Fonte: Autor

Na Figura 20, o vermelho significa que existe um desbalanceamento na árvore e são realçados os três nós a partir dos quais será efetuada a rotação.

A estrutura em javascript das funcionalidades originais do SYM desenvolvidas por SOUZA (2019, p.34), eram implementadas em um único arquivo, o script.js. Tendo como objetivo as boas práticas de programação e de reutilização de código orientada a modelos (LUCREDIO, 2009, p.37), optou-se por separar as principais funcionalidades em diferentes arquivos, conforme a Figura 21. Por este motivo não foi possível atualizar o arquivo original, script.js, embora o mesmo ainda funcione para as funcionalidades originais do SYM (Pilhas, Filas e Listas encadeadas).

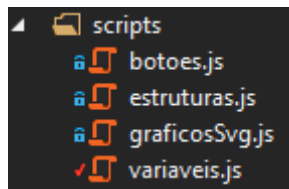


Figura 21: Novos scripts adicionados ao SYM

Fonte: Autor

Estruturas.js é o arquivo onde são implementadas todas as funcionalidades referentes à árvore, buscas, inserção e remoção de elementos, balanceamento e rotações, este é o arquivo responsável pelo funcionamento da árvore. GraficosSvg.js é o arquivo responsável por todas as funções referentes aos gráficos, desenhos dos elementos, colorização, rotações e locomoções de elementos. Botoes.js é o script responsável por tratar os eventos acionados através do click dos botões da tela, principais responsáveis pela interação do usuário com a ferramenta. Variaveis.js é o arquivo responsável pela parametrização de configurações referentes ao sistema, como o tempo de execução entre as ações, espaçamento entre os nós, largura dos nós, entre outros.

## 5.1 FUNCIONALIDADES DOS NOVOS MÓDULOS DO SYM

As principais funcionalidades do SYM, como controle de velocidade, inserir, remover, pesquisar elementos e gerar lista aleatória foram implementadas para as estruturas de árvores binárias de busca e de altura balanceada, conforme a Figura 22. Novos botões foram adicionados para a implementação de funções específicas da árvore, como o cálculo de altura e impressão dos elementos da árvore.

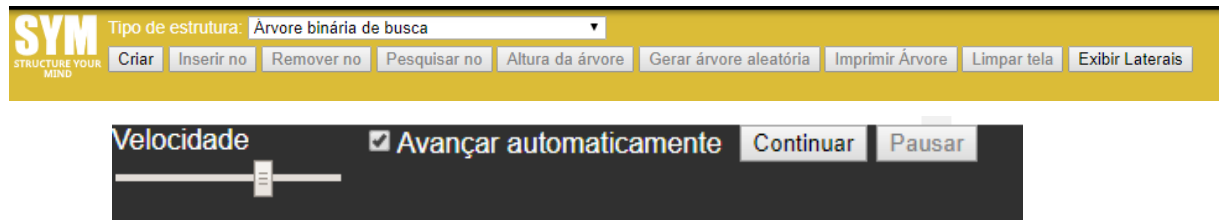


Figura 22: Principais botões da interface do SYM

Fonte: Autor

Na Figura 23 são apresentados os casos de uso do sistema para as novas estruturas de árvore binária de busca e de altura balanceada.

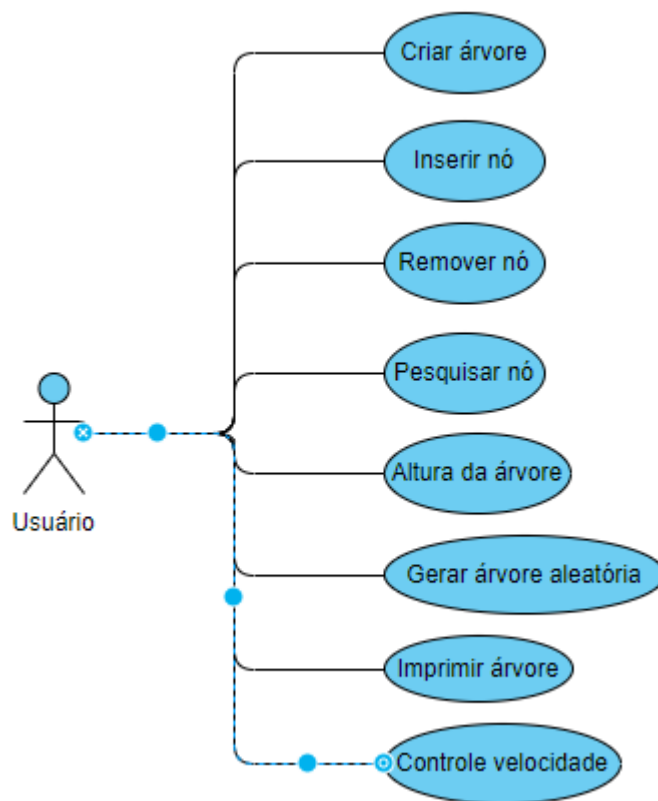


Figura 23: Novos casos de usos implementados no SYM

Fonte: Autor

### 5.1.1 Criar árvore

Cria a instância da árvore, se alguma árvore estiver criada os seus elementos são removidos e o ponteiro inicial da árvore aponta para nulo. É importante ressaltar que os elementos SVG gráficos também são removidos. Os botões são desbloqueados exceto o de impressão e de remoção (pelo fato de a árvore estar vazia). É desenhado na tela a representação do ponteiro inicial apontando para o nulo. Esta ação é demonstrada na Figura 24.

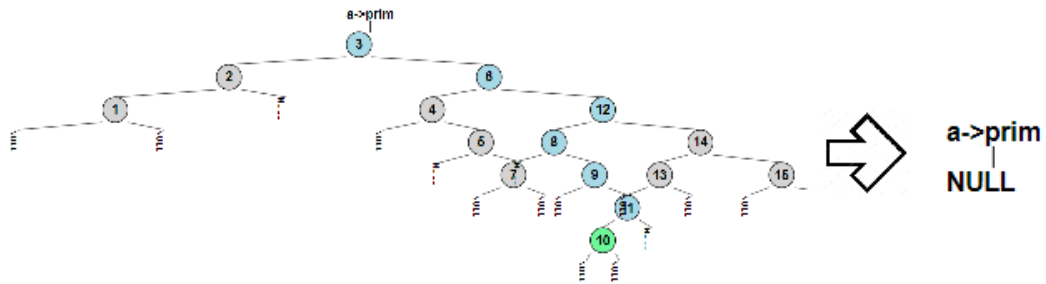


Figura 24: Funcionalidade Criar árvore

Fonte: Autor

### 5.1.2 Pesquisa pelo valor de nó

Pede ao usuário o valor a ser pesquisado na árvore, procede a fazer a pesquisa recursiva do elemento, caso ele não exista ou a árvore esteja vazia, informa ao usuário que o elemento não foi encontrado, ou no caso contrário afirma que foi encontrado com sucesso.

A função principal de busca foi implementada no arquivo estruturas.js, é uma função recursiva que aceita quatro parâmetros (linha 1 da Tabela 4), o objeto “No” é passado por referência, representa o nó que está sendo analisado na interação. A variável “valorBusca” é o valor inserido pelo usuário para efetuar a busca. A variável “operacao” que representa se a intenção da busca é apenas pesquisar por um elemento, onde a função seria chamada com o valor “P”, ou o valor “I” para efetuar uma inserção, ou “R” para efetuar uma remoção. O objeto “doDelete” para a pesquisa recebe o valor “{value:true}”, o mesmo é passado por referência, e tem a função de indicar se a remoção já foi efetuada, sua utilidade será explicada na seção 5.1.4.

Tabela 4: Algoritmo recursivo para a Busca, Inserção ou Remoção de um Nó.

Fonte: Autor

```

1 . async function e_BuscaNo(No, valorBusca, operacao, doDelete) {
2 .     await svg_paint_no(No, 'noPath', false);
3 .     var result;
4 .     if (No.valor === valorBusca) {
5 .         await svg_paint_no(No, 'noSuccess', false);
6 .         return true;
7 .     }
8 .     else if (No.valor > valorBusca) {
9 .         await svg_paint_no(No.no_direita, 'noDisabled', true);
10 .         if (No.no_esquerda === null) {
11 .             if (operacao === 'I') {
12 .                 No.no_esquerda = e_No(valorBusca);
13 .                 e_graficosNo(No.no_esquerda, true);
14 .                 await svg_paint_no(No.no_esquerda, 'noSuccess', false);
15 .             }
16 .             return false;
17 .         }
18 .         else {
19 .             result = await e_BuscaNo(No.no_esquerda, valorBusca, operacao,
doDelete);

```

```

20.         if (doDelete.value === true && result === true && operacao ===
'R') {
21.             await svg_paint_no(No.no_esquerda, 'noSuccess', false);
22.             await e_RemoveNo(No, 'E');
23.             doDelete.value = false;
24.         }
25.         return result;
26.     }
27. }
28. else if (No.valor < valorBusca) {
29.     await svg_paint_no(No.no_esquerda, 'noDisabled', true);
30.     if (No.no_direita === null) {
31.         if (operacao === 'I') {
32.             No.no_direita = e_No(valorBusca);
33.             e_graficosNo(No.no_direita, true);
34.             await svg_paint_no(No.no_direita, 'noSuccess', false);
35.         }
36.         return false;
37.     }
38.     else {
39.         result = await e_BuscaNo(No.no_direita, valorBusca, operacao,
doDelete);
40.         if (doDelete.value === true && result === true && operacao ===
'R') {
41.             await svg_paint_no(No.no_direita, 'noSuccess', false);
42.             await e_RemoveNo(No, 'D');
43.             doDelete.value = false;
44.         }
45.         return result;
46.     }
47. }
48. }

```

Conforme a explicação dos algoritmos de LAUREANO (2009, p.130), as três funções de pesquisa, remoção e inserção dependem de efetuar uma busca recursiva pela árvore até encontrar o nó que será utilizado. Dessa forma, de forma a melhorar a reutilização de código, evitando que a lógica de três algoritmos de busca diferentes ficasse espalhada pelo sistema, os algoritmos originais foram adaptados para que um único pudesse implementar as três funcionalidades. Se a variável “operacao” for definida como “P” (Pesquisa) e o valor do nó na iteração recursiva for igual ao valor de “valorBusca”, a função retorna positivo por ter encontrado o valor (linha 6 da Tabela 4), caso o valor do nó for nulo, a busca retorna negativo, pois o nó não existe (linhas 16 e 36 da Tabela 4). Seguindo a explicação de LAUREANO (2009, p.131), se o valor do nó for menor que o valor de “valorBusca”, a pesquisa continua recursivamente através do elemento à direita (linha 39 da Tabela 4), caso o valor do nó seja maior, a pesquisa continua através do elemento à esquerda (linha 19 da Tabela 4).

Para o realçamento dos nós através da cor, visualizados na Figura 25, a função `svg_paint_no` é chamada e recebe os parâmetros: Objeto “No”, o nó da árvore é passado como referência para ser pintado. A classe que indica a cor do nó, neste caso “noPath” seria a coloração azul, que indica que a iteração do nó está passando por esse caminho. “noDisabled”

seria a coloração cinza, indicando que esse nó será descartado por não fazer parte do caminho de busca. “noSuccess” indica que o nó que estava sendo procurado foi encontrado e receberá a coloração verde. O terceiro parâmetro da função `svg_paint_no` recebe um booleano para a variável “cascade”, este parâmetro indica se todos os filhos do nó também serão pintados da mesma cor, recebe o valor “true” quando um caminho descartado receberá a cor cinza para todos seus filhos (linhas 9 e 29 da Tabela 4).

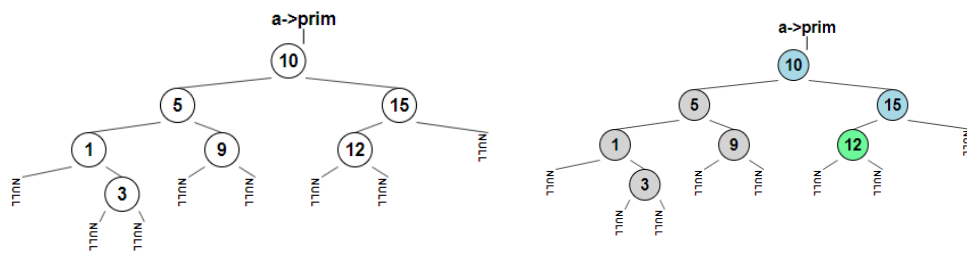


Figura 25: Busca pelo nó de valor 12

Fonte: Autor

### 5.1.3 Inserção de nó

É utilizado o algoritmo da Tabela 4, entretanto, a diferença entre a função de pesquisa nos valores de parâmetros para a chamada da função, é na variável “operacao” onde é atribuída o valor “I”. A lógica de busca através dos nós permanece igual, assim como o realçamento de cores.

Na iteração do algoritmo, caso o valor do nó seja nulo significa que o nó não foi encontrado, e desta forma procede a inserir um novo nó com o valor da variável “valorBusca” nesse espaço (linhas 12 e 32 da Tabela 4). Como o objeto “No” é passado como referência, a alteração do seu valor dentro da função implica na mudança da árvore global.

A função `e_graficosNo` é chamada com os parâmetros “No” passado por referência do nó que foi inserido e a variável “animate” recebe o valor “true” (linhas 13 e 33 da Tabela 4). Esta função tem a responsabilidade de criar os novos elementos SVG gráficos para o nó e posteriormente mover a árvore. Entre os elementos SVG criados para o controle dos gráficos se encontram: O círculo, o número do nó, a linha que aponta para o próximo elemento à esquerda e a linha que aponta para o próximo elemento à direita, caso o elemento para o nulo, possui o elemento texto que dirá “NULL”, o elemento texto do índice de balanceamento e objeto de direções “x1,y1,x2l,x2r,y2”. “x1,y1” contém a posição de onde o elemento inicia a ser desenhado, “x2l,x2r,y2” possuem as posições de onde os elementos filhos à esquerda e à direita serão desenhados (linhas 9 a 13 da Tabela 2). A partir destas direções do nó pai, os nós

filhos começam a ser desenhados. Na Figura 26 é possível verificar o exemplo de busca e inserção para um valor:

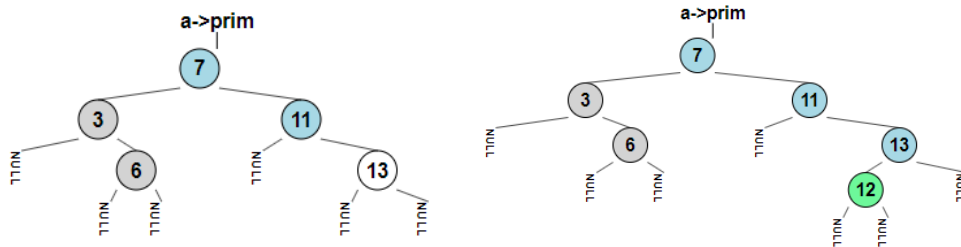


Figura 26: Busca e inserção do elemento 12, deslocamento dos nós

Fonte: Autor

Para mover a árvore é calculada a altura da árvore, isto permite saber o espaçamento que todos os nós precisam possuir, pois, entre mais alta a árvore, mais elementos internos possui, maior deve ser a abertura da árvore para as laterais. Após o cálculo da altura da árvore, é chamada a função `svg_mover_arvore`, com os parâmetros “h” que é a altura da árvore e “animate” que é o mesmo valor passado com a qual a função `e_graficosNo` foi passada. A variável “animate” permite que as mudanças dos elementos sejam feitas de acordo com a velocidade definida pelo usuário, quando seu valor é “true”, quando seu valor é falso, os elementos são movidos instantaneamente.

Tabela 5: Algoritmo de movimentação recursiva dos elementos gráficos da árvore de acordo com a altura.

Fonte: Autor

```

1 . function svg_mover_no(No, h, x1, y1, fronteira, animate) {
2 .     if (No !== null) {
3 .         h--;
4 .         var distfronteira = h * espacamento;
5 .         var distnormal = h * 0.5 * espacamento;
6 .         var x2l = x1 + (larguraNo * (fronteira === 'E' ? distfronteira :
distnormal) * -1);
7 .         var x2r = x1 + (larguraNo * (fronteira === 'D' ? distfronteira :
distnormal));
8 .         var y2 = y1 + larguraNo * 1.3;
9 .         No.svg_nopont.directions.x1 = x1;
10 .        No.svg_nopont.directions.y1 = y1;
11 .        No.svg_nopont.directions.y2 = y2;
12 .        No.svg_nopont.directions.x2l = x2l;
13 .        No.svg_nopont.directions.x2r = x2r;
14 .        move(No, animate);
15 .        svg_mover_no(No.no_esquerda, h, x2l, y2, fronteira === 'E' ? 'E' : 'N',
animate);
16 .        svg_mover_no(No.no_direita, h, x2r, y2, fronteira === 'D' ? 'D' : 'N',
animate);
17 .    }

```



O algoritmo trabalha de forma recursiva, e possui um parâmetro chamado “fronteira”, o qual lhe permite saber se o nó que está sendo movimentado se encontra nas extremidades da árvore. Este fator é útil para o espaçamento, pois, conforme a árvore vai aumentando de altura, mais elementos serão inseridos em potências de 2, logo, no oitavo nível, teremos até 256 nós (BAUER, 2018, pág. 25). A partir deste conceito, os nós das extremidades esquerda e direita serão afastados o máximo possível (representados pela variável “distfronteira”, enquanto que os nós internos serão afastados o um valor consideravelmente menor, de forma a que evitar a sobreposição de elementos (representados pela variável “distnormal”) (linhas 4 e 5 da Tabela 5). O algoritmo da Tabela 5 calcula o objeto de direções do elemento gráfico SVG e chama recursivamente os filhos esquerdo e direito para que possam também ser movidos (linhas 15 e 16 da Tabela 5), o nó é efetivamente movido a partir da função “move(No, animate)”, o objeto “No” é o nó a ser movido e a variável “animate” determina se o deslocamento será visível ao usuário (linha 14 da Tabela 5).

Tabela 6: Algoritmo de deslocamento dos elementos gráficos de um nó

Fonte: Autor

```

1 . function move(No, animate) {
2 .     var speed = animate === true ? velocidade : 1;
3 .     var directions = No.svg_nopont.directions;
4 .     No.svg_nopont.circulo.animate(speed).move(directions.x1, directions.y1);
5 .     if (No.svg_nopont.balance !== null) {
6 .         remove(No.svg_nopont.balance);
7 .         No.svg_nopont.balance = null;
8 .     }
9 .     var numeroDimensions =
document.getElementById(No.svg_nopont.numero.node.id).getBoundingClientRect();
10 .    var heightNumero = numeroDimensions.height;
11 .    var widthNumero = numeroDimensions.width;
12 .    var tx = directions.x1 + ((larguraNo - widthNumero) / 2);
13 .    var ty = directions.y1 + ((larguraNo - heightNumero) / 2);
14 .    No.svg_nopont.numero.animate(speed).move(tx, ty);
15 .    remove(No.svg_nopont.linha_dir);
16 .    remove(No.svg_nopont.linha_esq);
17 .    var lx2l = directions.x2l + larguraNo / 2;
18 .    var lx2r = directions.x2r + larguraNo / 2;
19 .    var ly2 = directions.y2;
20 .    var linhaesq = svg.line(directions.x1, directions.y1 + larguraNo, lx2l,
ly2);
21 .    linhaesq.stroke('#000000');
22 .    fadeIn(linhaesq, speed);
23 .    No.svg_nopont.linha_esq = linhaesq;
24 .    var linhadir = svg.line(directions.x1 + larguraNo, directions.y1 +
larguraNo, lx2r, ly2);
25 .    linhadir.stroke('#000000');
26 .    fadeIn(linhadir, speed);
27 .    No.svg_nopont.linha_dir = linhadir;
28 .    remove(No.svg_nopont.null_esq);
29 .    No.svg_nopont.null_esq = null;
30 .    if (No.no_esquerda === null) {
31 .        No.svg_nopont.null_esq = svg.plain("NULL");
32 .        No.svg_nopont.null_esq.node.style.fontSize = fontsize * 0.65;
33 .        No.svg_nopont.null_esq.move(lx2l, ly2, animate);

```

```

34.         No.svg_nopont.null_esq.rotate(90, lx2l, ly2);
35.         fadeIn(No.svg_nopont.null_esq, speed * 1.75);
36.     }
37.     remove(No.svg_nopont.null_dir);
38.     No.svg_nopont.null_dir = null;
39.
40.     if (No.no_direita === null) {
41.         No.svg_nopont.null_dir = svg.plain("NULL");
42.         No.svg_nopont.null_dir.node.style.fontSize = fontsize * 0.65;
43.         No.svg_nopont.null_dir.move(lx2r, ly2, animate);
44.         No.svg_nopont.null_dir.rotate(90, lx2r, ly2);
45.         fadeIn(No.svg_nopont.null_dir, speed * 1.75);
46.     }
47. }

```

A velocidade com a qual os elementos são deslocados através da função `move` do SVG é parametrizada pelo próprio usuário, ou recebe o valor de 1 milissegundo na eventualidade da variável “`animate`” receber o valor “`false`” (linha 2 da Tabela 6). O elemento nó já possui um objeto com as direções (“`No.svg_nopont.directions = {x1,y1,x2r,x2l,y2}`”) (linha 3 da Tabela 6), a partir das quais o elemento gráfico será desenhado.

Alguns elementos gráficos são removidos, como o texto do título de balanceamento (“`No.svg_nopont.balance`”) (linha 5 a 8 da Tabela 6), pois este será recalculado após o deslocamento da árvore. O texto para onde o elemento aponta se o próximo elemento filho for nulo “`No.svg_nopont.null_esq`” ou “`No.svg_nopont.null_dir`” não é inserido se o próximo elemento possuir valor (linhas 28 a 46 da Tabela 6). As linhas são removidas e sua posição é calculada novamente, os elementos de linha SVG não disponibilizavam um método prático para efetuar o deslocamento, ao contrário do círculo, desta forma foi optado por removê-las e desenhá-las novamente a cada iteração do algoritmo (linhas 15 a 27 da Tabela 6).

As direções com as quais cada elemento gráfico é posicionado são diferentes, entretanto são todas calculadas pelo próprio algoritmo a partir das direções iniciais do círculo.

#### 5.1.4 Remoção de um nó

É utilizado o algoritmo de busca da Tabela 4 novamente, para encontrar o nó a ser excluído. São modificados os parâmetros “`operacao`” para “`R`” (Remoção) e o objeto “`doDelete`” é colocado para “`{value:true}`”. De acordo com FLANAGAN (2013, p.166), para fazer a chamada de uma função em Javascript e que esse parâmetro seja modificado fora do escopo da função, é necessário que o parâmetro seja um objeto. Por este motivo, o “`doDelete`”, que só possui um valor, booleano, é contruído e passado como um objeto. Entretanto, se a propriedade de um objeto também for um objeto, por exemplo, “`NoPai.NoFilho`”, e `NoFilho` for enviado para uma função para ser removido, por exemplo, “`remove(NoFilho) {NoFilho = null;}`”, `NoPai` não será modificado, e “`NoPai.NoFilho`” continua possuindo valor.

Tabela 7: Trecho exemplo do algoritmo da remoção do algoritmo na Tabela 4 de Busca.

Fonte: Autor

```

1. result = await e_BuscaNo(No.no_direita, valorBusca, operacao, doDelete);
2.     if (doDelete.value === true && result === true && operacao === 'R')
3.     {
4.         await svg_paint_no(No.no_direita, 'noSuccess', false);
5.         await e_RemoveNo(No, 'D');
6.         doDelete.value = false;

```

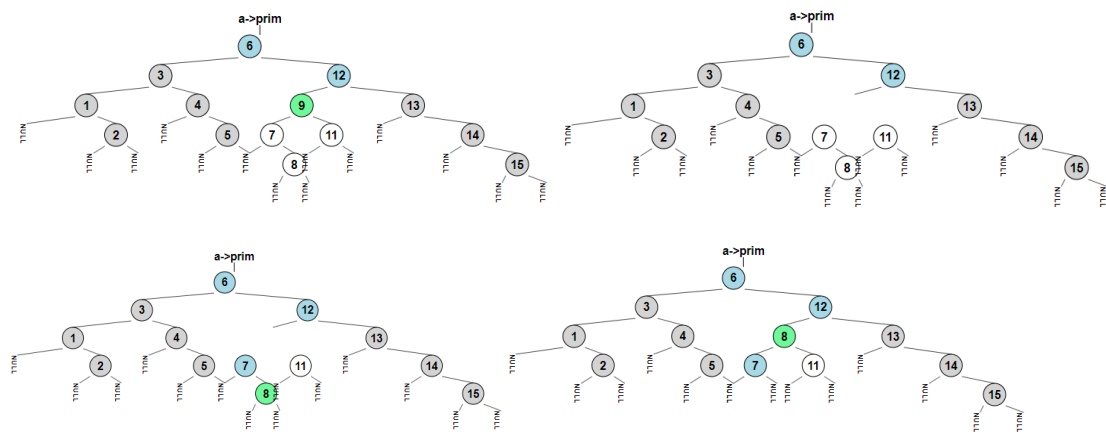


Figura 27: Busca, remoção e substituição de um nó

Fonte: Autor

O próximo algoritmo de remoção, a função “e\_RemoveNo” (linha 4, Tabela 7), precisou ter sua lógica duplicada para a remoção de um nó no lado esquerdo, ou um nó no lado direito. Isto se deve a peculiaridade do Javascript comentada na seção 4.1.3, na passagem de objetos como referência. No mesmo exemplo citado na seção 4.1.3, para que um nó filho objeto “NoPai.NoFilho” seja removido com sucesso, não seria possível enviar o “NoFilho” para a função e\_RemoveNo, pois, após a exclusão, estas mudanças não seriam refletidas fora do escopo da mesma. Portanto, é necessário a passagem do “NoPai” à função, indicando se o nó a ser removido será o esquerdo ou o direito, passando respectivamente os valores “E” ou “D” para o parâmetro “lado” da função e\_RemoveNo (linha 1 da Tabela 8).

Se o nó a ser removido é o da esquerda (linha 3 da Tabela 8), é executado o bloco de código das linhas 4 a 40 do algoritmo da Tabela 8, caso contrário, é executado o bloco de código das linhas 20 a 79 do algoritmo da Tabela 8, a linha 81 do algoritmo da Tabela 8 faz o deslocamento da árvore de acordo com a nova altura a ser obtida após a remoção do nó. Os passos de funcionamento deste algoritmo podem ser visualizados na Figura 27.

A remoção do nó é simples, são removidos os elementos SVG gráficos (linha 4 e 43 da Tabela 8), a complexidade do algoritmo se encontra em encontrar o nó correto para fazer a substituição do nó antigo de forma a manter a propriedade de busca da árvore binária.

Tabela 8: Algoritmo de remoção de um nó.

Fonte: Autor

```

1 . async function e_RemoveNo(No, lado) {
2 .     var noIt = null, novoNo = null;
3 .     if (lado === 'E') {
4 .         svg_removeSvgs(No.no_esquerda, false);
5 .         if (No.no_esquerda.no_esquerda === null) {
6 .             if (No.no_esquerda.no_direita === null) {
7 .                 No.no_esquerda = null;
8 .             }
9 .             else {
10.                await svg_paint_no(No.no_esquerda.no_direita, 'noSuccess',
false);
11.                No.no_esquerda = No.no_esquerda.no_direita;
12.            }
13.        }
14.        else {
15.            if (No.no_esquerda.no_direita === null) {
16.                await svg_paint_no(No.no_esquerda.no_esquerda, 'noSuccess',
false);
17.                No.no_esquerda = No.no_esquerda.no_esquerda;
18.            }
19.            else {
20.                noIt = No.no_esquerda.no_esquerda;
21.                await svg_paint_no(noIt, 'noPath', false);
22.                if (noIt.no_direita === null) {
23.                    noIt.no_direita = No.no_esquerda.no_direita;
24.                    No.no_esquerda = noIt;
25.                    await svg_paint_no(noIt, 'noSuccess', false);
26.                }
27.                else {
28.                    while (noIt.no_direita.no_direita !== null) {
29.                        noIt = noIt.no_direita;
30.                        await svg_paint_no(noIt, 'noPath', false);
31.                    }
32.                    novoNo = noIt.no_direita;
33.                    noIt.no_direita = novoNo.no_esquerda;
34.                    novoNo.no_esquerda = No.no_esquerda.no_esquerda;
35.                    novoNo.no_direita = No.no_esquerda.no_direita;
36.                    No.no_esquerda = novoNo;
37.                    await svg_paint_no(novoNo, 'noSuccess', false);
38.                }
39.            }
40.        }
41.    }
42.    else {
43.        svg_removeSvgs(No.no_direita, false);
44.        if (No.no_direita.no_esquerda === null) {
45.            if (No.no_direita.no_direita === null) {
46.                No.no_direita = null;
47.            }
48.            else {
49.                await svg_paint_no(No.no_direita.no_direita, 'noSuccess',
false);
50.                No.no_direita = No.no_direita.no_direita;
51.            }
52.        }
53.        else {
54.            if (No.no_direita.no_direita === null) {
55.                await svg_paint_no(No.no_direita.no_esquerda, 'noSuccess',
false);
56.                No.no_direita = No.no_direita.no_esquerda;
57.            }
58.            else {
59.                noIt = No.no_direita.no_esquerda;
60.                await svg_paint_no(noIt, 'noPath', false);
61.                if (noIt.no_direita === null) {

```

```

62.         noIt.no_direita = No.no_direita.no_direita;
63.         No.no_direita = noIt;
64.         await svg_paint_no(noIt, 'noSuccess', false);
65.     }
66.     else {
67.         while (noIt.no_direita.no_direita !== null) {
68.             noIt = noIt.no_direita;
69.             await svg_paint_no(noIt, 'noPath', false);
70.         }
71.         novoNo = noIt.no_direita;
72.         noIt.no_direita = novoNo.no_esquerda;
73.         novoNo.no_esquerda = No.no_direita.no_esquerda;
74.         novoNo.no_direita = No.no_direita.no_direita;
75.         No.no_direita = novoNo;
76.         await svg_paint_no(novoNo, 'noSuccess', false);
77.     }
78. }
79. }
80. }
81. await e_moverArvore(true);
82. }

```

Caso ambos os filhos do nó a ser removido sejam nulos, o nó é substituído por um valor nulo (linhas 5 a 8 ou 44 a 47 da Tabela 8). Caso o filho da esquerda seja nulo, mas o filho da direita não, o nó é substituído pelo filho da direita (linhas 9 a 12 ou 48 a 51 da Tabela 8).

Caso o filho da esquerda não seja nulo, o nó é realçado de cor azul e recebe o nome auxiliar de “noIt” (nó da iteração) (linhas 19 a 21 ou 58 a 60 da Tabela 8). Caso o filho da direita de “noIt” seja nulo, “noIt” substitui o nó que está sendo removido (linhas 22 a 25 ou 61 a 64 da Tabela 8). Caso o filho da direita de “noIt” possua valor, é iniciada uma busca através de um laço de repetição pelo filho mais à direita de “noIt”, o resultado dessa busca é nomeado como “novoNo” e “noIt” passa a ser o pai de “novoNo” (linhas 27 a 32 ou 66 a 71 da Tabela 8). Como “novoNo” vai substituir o nó a ser removido, o filho à esquerda do “novoNo” passa a ser o filho à direita do “noIt”, o filho à direita de “novoNo” é vazio, pois ele é o mais à direita (linha 33 ou 72 da Tabela 8). “novoNo” recebe os filhos à esquerda e à direita do nó a ser removido (que constitui a substituição) e o “NoPai” da iteração principal, que é o pai do nó a ser excluído, recebe como filho o “novoNo” (linhas 34 a 38 ou 73 a 77 da Tabela 8).

Conforme a busca pelo nó que substituirá o nó removido é feita, os nós são realçados conforme as iterações de busca, para no caso de sucesso, realçar o nó a substituir de verde (linhas 10,15,21,25,30,37 ou 49,54,60,64,69,76 do código).

### 5.1.5 Altura da árvore

Para o cálculo da altura da árvore, são realçados os nós que estão sendo iterados em amarelo, quando o parâmetro de entrada “paint” é indicado como verdadeiro (Linhas 4 a 5 da Tabela 9). É uma função recursiva que retorna o maior valor de altura para as subárvores à esquerda e à direita do nó.

Tabela 9: Algoritmo de cálculo da altura da árvore.

Fonte: Autor

```

1 . async function e_Altura_Arvore(No, h, paint = false) {
2 .     if (No !== null) {
3 .         $('#acao-principal').text("Calculando a altura dos nos...");
4 .         if (paint === true) {
5 .             await svg_paint_no(No, 'noHeight', false);
6 .         }
7 .         h++;
8 .         var he = await e_Altura_Arvore(No.no_esquerda, h, paint);
9 .         var hd = await e_Altura_Arvore(No.no_direita, h, paint);
10 .        h = he > h ? he : h;
11 .        h = hd > h ? hd : h;
12 .        return h;
13 .    }
14 .    else return 0;
15 . }
```

A função é chamada pela primeira vez com o parâmetro “h” igual a 0, se o elemento acessado for diferente de nulo, um novo nível de altura foi alcançado, e chama a função recursivamente para que o valor da altura seja calculada para as subárvores do elemento (linhas 7 a 9 da Tabela 9). Posteriormente é feita a comparação da maior altura entre a subárvore esquerda e a direita, e a maior determinará a altura real daquele nó. A função retorna para propagação da altura do cálculo recursivo (Linhas 10 a 12 da Tabela 9).

### 5.1.6 Balanceamento da árvore

Conforme estudado no capítulo 3, o cálculo do índice de balanceamento da árvore AVL precisa ser efetuado após a inserção ou remoção de um elemento, e no caso de um dos índices não pertencer a  $\{-1, 0, 1\}$ , a árvore será balanceada.

Tabela 10: Algoritmo de balanceamento da árvore.

Fonte: Autor

```

1 . async function e_BalancearArvore() {
2 .     if (tipo_arvore === 'AVL') {
3 .         // O algoritmo só é efetuado se a árvore escolhida for AVL;
4 .         var desbalanceado = { valor: null };
5 .         await e_Calcular_Balanceamento(arvore.raiz, desbalanceado, true);
6 .         if (desbalanceado.valor !== null) {
7 .             await e_RotacaoArvore(desbalanceado);
8 .             await e_BalancearArvore();
9 .         }
10 .    }
```

```

9 .      }
10.    }
11. }

```

As rotações de balanceamento só serão efetuadas se algum índice de balanceamento for diferente de  $\{-1,0,1\}$ , e as rotações serão feitas a partir do nó desbalanceado mais abaixo. Por motivo de otimização, no algoritmo recursivo, o último nó a ficar desbalanceado é o mais perto da folha, portanto, é passado um objeto por referência à função “e\_Calcular\_Balanceamento” (linha 4 a 5 da Tabela 10), que será atualizado sempre que houver um desbalanceamento, portanto seu valor será o valor do nó onde serão iniciadas as rotações. Caso o valor de “desbalanceado” seja nulo, significa que a árvore está balanceada. No caso contrário, a árvore será rotacionada e posteriormente será chamado de forma recursiva novamente o algoritmo de balanceamento, no caso de outro balanceamento ser necessário (Linhas 7 a 8 da Tabela 10).

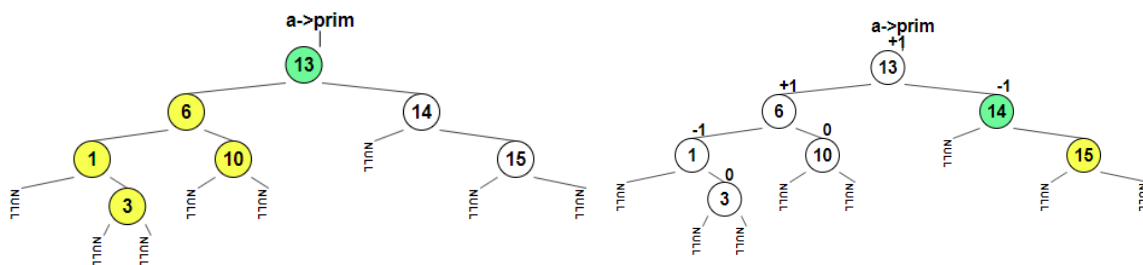


Figura 28: Cálculo dos índices de balanceamento

Fonte: Autor

O cálculo do índice de balanceamento, verificado na Figura 28, é efetuado através da diferença de alturas da subárvore esquerda e a subárvore direita e guardado na propriedade “balance” do nó (Linhas 5 a 7 da Tabela 11). O nó é realçado de verde enquanto o seu índice está sendo calculado (Linha 4 da Tabela 11), e dentro de suas subárvores conforme cada nó é iterado é realçado de amarelo conforme o algoritmo da Tabela 9 de cálculo da altura do nó.

Tabela 11: Algoritmo de cálculo do índice e balanceamento de um nó.

Fonte: Autor

```

1 . async function e_Calcular_Balanceamento(No,desbalanceado,cascade = true) {
2 .     if (No !== null) {
3 .         await svg_limparCores();
4 .         await svg_paint_no(No, 'noSuccess', false, false);
5 .         var he = await e_Altura_Arvore(No.no_esquerda, 0,true);
6 .         var hd = await e_Altura_Arvore(No.no_direita, 0, true);
7 .         No.balance = he - hd;
8 .         if (Math.abs(No.balance) > 1) {
9 .             desbalanceado.valor = No.valor;
10.        }
11.        await svg_draw_Balance(No);

```





```

33 .         case 'LR': {
34 .             NoB = No.no_esquerda.no_esquerda;
35 .             NoC = No.no_esquerda.no_esquerda.no_direita;
36 .             NoA.no_esquerda = NoC.no_direita;
37 .             NoB.no_direita = NoC.no_esquerda;
38 .             NoC.no_esquerda = NoB;
39 .             NoC.no_direita = NoA;
40 .             No.no_esquerda = NoC;
41 .             break;
42 .         }
43 .         case 'RR': {
44 .             NoB = No.no_esquerda.no_direita;
45 .             NoC = No.no_esquerda.no_direita.no_direita;
46 .             NoA.no_direita = NoB.no_esquerda;
47 .             NoB.no_direita = NoC;
48 .             NoB.no_esquerda = NoA;
49 .             No.no_esquerda = NoB;
50 .             break;
51 .         }
52 .         case 'RL': {
53 .             NoB = No.no_esquerda.no_direita;
54 .             NoC = No.no_esquerda.no_direita.no_esquerda;
55 .             NoA.no_direita = NoC.no_esquerda;
56 .             NoB.no_esquerda = NoC.no_direita;
57 .             NoC.no_esquerda = NoA;
58 .             NoC.no_direita = NoB;
59 .             No.no_esquerda = NoC;
60 .             break;
61 .         }
62 .     }
63 .     await svg_paint_no(NoA, 'noWarning', false);
64 .     await svg_paint_no(NoB, 'noWarning', false);
65 .     await svg_paint_no(NoC, 'noWarning', false);
66 .     await sleep(velocidade * 3);
67 .     await e_moverArvore(true);
68 . }
69 . else if (No.no_esquerda.valor > desbalanceado.valor) {
70 .     await e_RotacaoNo(No.no_esquerda, 'E', desbalanceado);
71 . }
72 . else {
73 .     await e_RotacaoNo(No.no_esquerda, 'D', desbalanceado);
74 . }
75 . }
76 . }
77 . else {
78 .     if (No.no_direita !== null) {
79 .         if (No.no_direita.valor === desbalanceado.valor) {
80 .             if (No.no_direita.balance > 0) {
81 .                 tipoTransformacao = 'L';
82 .                 if (No.no_direita.no_esquerda.balance >= 0) {
83 .                     tipoTransformacao += 'L';
84 .                 }
85 .                 else tipoTransformacao += 'R';
86 .             }
87 .             else {
88 .                 tipoTransformacao = 'R';
89 .                 if (No.no_direita.no_direita.balance > 0) {
90 .                     tipoTransformacao += 'L';
91 .                 }
92 .                 else tipoTransformacao += 'R';
93 .             }
94 .             NoA = No.no_direita;
95 .             $('#acao-principal').text("Tipo de rotacao: " +
tipoTransformacao);
96 .             switch (tipoTransformacao) {
97 .                 case 'LL': {
98 .                     NoB = No.no_direita.no_esquerda;
99 .                     NoC = No.no_direita.no_esquerda.no_esquerda;

```

```

100.         NoA.no_esquerda = NoB.no_direita;
101.         NoB.no_esquerda = NoC;
102.         NoB.no_direita = NoA;
103.         No.no_direita = NoB;
104.         break;
105.     }
106.     case 'LR': {
107.         NoB = No.no_direita.no_esquerda;
108.         NoC = No.no_direita.no_esquerda.no_direita;
109.         NoA.no_esquerda = NoC.no_direita;
110.         NoB.no_direita = NoC.no_esquerda;
111.         NoC.no_esquerda = NoB;
112.         NoC.no_direita = NoA;
113.         No.no_direita = NoC;
114.         break;
115.     }
116.     case 'RR': {
117.         NoB = No.no_direita.no_direita;
118.         NoC = No.no_direita.no_direita.no_direita;
119.         NoA.no_direita = NoB.no_esquerda;
120.         NoB.no_direita = NoC;
121.         NoB.no_esquerda = NoA;
122.         No.no_direita = NoB;
123.         break;
124.     }
125.     case 'RL': {
126.         NoB = No.no_direita.no_direita;
127.         NoC = No.no_direita.no_direita.no_esquerda;
128.         NoA.no_direita = NoC.no_esquerda;
129.         NoB.no_esquerda = NoC.no_direita;
130.         NoC.no_esquerda = NoA;
131.         NoC.no_direita = NoB;
132.         No.no_direita = NoC;
133.         break;
134.     }
135. }
136. await svg_paint_no(NoA, 'noWarning', false);
137. await svg_paint_no(NoB, 'noWarning', false);
138. await svg_paint_no(NoC, 'noWarning', false);
139. await sleep(velocidade * 3);
140. await e_moverArvore(true);
141. }
142. else if (No.no_direita.valor > desbalanceado.valor) {
143.     await e_RotacaoNo(No.no_direita, 'E', desbalanceado);
144. }
145. else {
146.     await e_RotacaoNo(No.no_direita, 'D', desbalanceado);
147. }
148. }
149. }
150. }

```

Para este algoritmo foi necessário a duplicação da lógica da mesma forma que no algoritmo da Tabela 8, de remoção de nó, considerando que os elementos da árvore seriam alterados, por limitações da linguagem Javascript, não poderia ser passado o nó desbalanceado como referência, e sim o pai dele, de forma a afetar a árvore fora do escopo local da função.

Para efetuar a rotação, primeiramente é necessário encontrar o pai do elemento que se encontra desbalanceado, para isto é feita uma busca de acordo com o valor que se encontra no objeto “desbalanceado” (linhas 5, 69 a 74 da Tabela 12), preenchido no algoritmo da Tabela

11 durante o cálculo de índices de balanceamento. O parâmetro “lado” é utilizado devido à iteração recursiva estar sendo feita através do nó pai, logo, é necessário saber se o nó desbalanceado seria o filho “E” (Esquerdo) ou “D” (Direito). Quando o nó desbalanceado é encontrado no lado esquerdo, o bloco de código com as linhas 5 a 68 são executadas, caso contrário, são executadas as linhas 77 a 141, estes blocos são os responsáveis pela rotação da árvore.

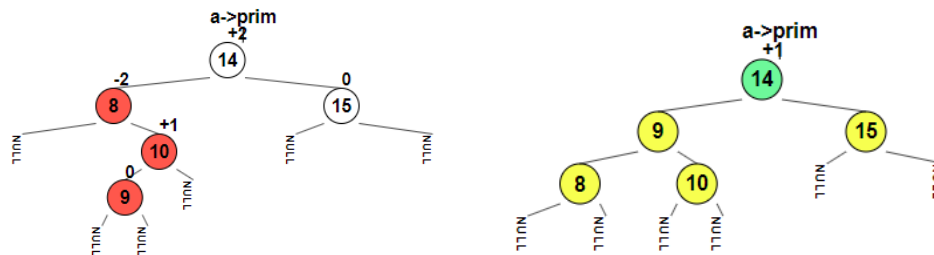


Figura 29: Balanceamento e Rotação de árvore AVL

Fonte: Autor

Primeiro é necessário definir o tipo de rotação, se será LL (Left-Left), LR (Left-Right), RL (Right-Left), como na Figura 29, ou RR (Right-Right), para isso, é necessário verificar os índices de balanceamento dos nós abaixo do nó desbalanceado, para existir um desbalanceamento significa que a altura de uma das duas subárvores é superior por duas unidades ou mais, o que significa que sempre existirá no mínimo dois nós abaixo do nó desbalanceado para efetuar a rotação. Se o nó desbalanceado possuir um índice maior a zero, significa que a subárvore esquerda é maior, a variável recebe “tipoTransformacao” igual a “L”, do contrário, recebe “tipoTransformacao” igual a “R”, em seguida, é feito o mesmo teste para o nó da esquerda (no caso de ter sido “L”) ou da direita (no caso de ter sido “R”), se o índice de balanceamento desse nó for maior a zero, novamente para “tipoTransformacao” é adicionado “L” ou caso contrário é adicionado “R”, dessa forma, é possível ter o tipo de rotação a ser efetuada (Linhas 7 a 20 ou 80 a 93 da Tabela 12).

Posteriormente, de acordo com o tipo de rotação, são efetuadas as substituições apresentadas na seção 3.2.4 de acordo com a variável “tipoTransformacao” (Linhas 21 a 62 ou 94 a 135 da Tabela 12). Para ajudar com a visualização ao usuário, os três nós participantes da rotação são realçados pela cor vermelha e espera-se um tempo até que a árvore seja movimentada conforme a nova altura (linhas 63 a 67 ou 136 a 140 da Tabela 12). O deslocamento dos nós através da nova altura são representados nos algoritmos das Tabelas 5 e 6.

### 5.1.8 Impressão da árvore

Tabela 13: Algoritmo de Percurso e impressão da árvore.

Fonte: Autor

```

1 . async function e_Imprimir_RED(No) {
2 .     if (No !== null) {
3 .         await svg_paint_no(No, 'noPath', false);
4 .         b_logIndex(", " + No.valor, false);
5 .         await e_Imprimir_RED(No.no_esquerda);
6 .         await e_Imprimir_RED(No.no_direita);
7 .     }
8 . }
9 .
10. async function e_Imprimir_ERD(No) {
11.     if (No !== null) {
12.         await e_Imprimir_ERD(No.no_esquerda);
13.         await svg_paint_no(No, 'noPath', false);
14.         b_logIndex(", " + No.valor, false);
15.         await e_Imprimir_ERD(No.no_direita);
16.     }
17. }
18.
19. async function e_Imprimir_EDR(No) {
20.     if (No !== null) {
21.         await e_Imprimir_EDR(No.no_esquerda);
22.         await e_Imprimir_EDR(No.no_direita);
23.         await svg_paint_no(No, 'noPath', false);
24.         b_logIndex(", " + No.valor, false);
25.     }
26. }

```

A impressão de todos os elementos em uma árvore de busca binária é uma ótima forma de exercitar as formas mais comuns de percurso em árvores binárias. Para efetuar estes percursos foram construídos algoritmos recursivos conforme os seguintes passos:

No percurso Pré-Ordem (Raiz,Esquerda,Direita ou “RED”) é visitado primeiramente o nó acessado, posteriormente sua subárvore esquerda e finalmente sua subárvore direita (Linhas 1 a 8 da Tabela 13). No percurso In-Ordem (Esquerda,Raiz,Direita ou “ERD”) é visitado primeiramente a subárvore esquerda, posteriormente o nó acessado e finalmente a subárvore direita (Linhas 10 a 17 da Tabela 13). No Percurso Pós-Ordem (Esquerda,Direita,Raiz ou “EDR”) é visitada primeiramente a subárvore esquerda, posteriormente a subárvore direita e finalmente o nó acessado (Linhas 19 a 26 da Tabela 13). A simulação do percurso percorrido pode ser visualizada na Figura 30.

Para auxiliar na visualização do usuário, conforme os nós vão sendo impressos, eles são realçados em azul (cor definida para a iteração de busca), de forma que seja perceptível o percurso que é feito por cada um dos algoritmos (Linhas 3,13,23 da Tabela 13).

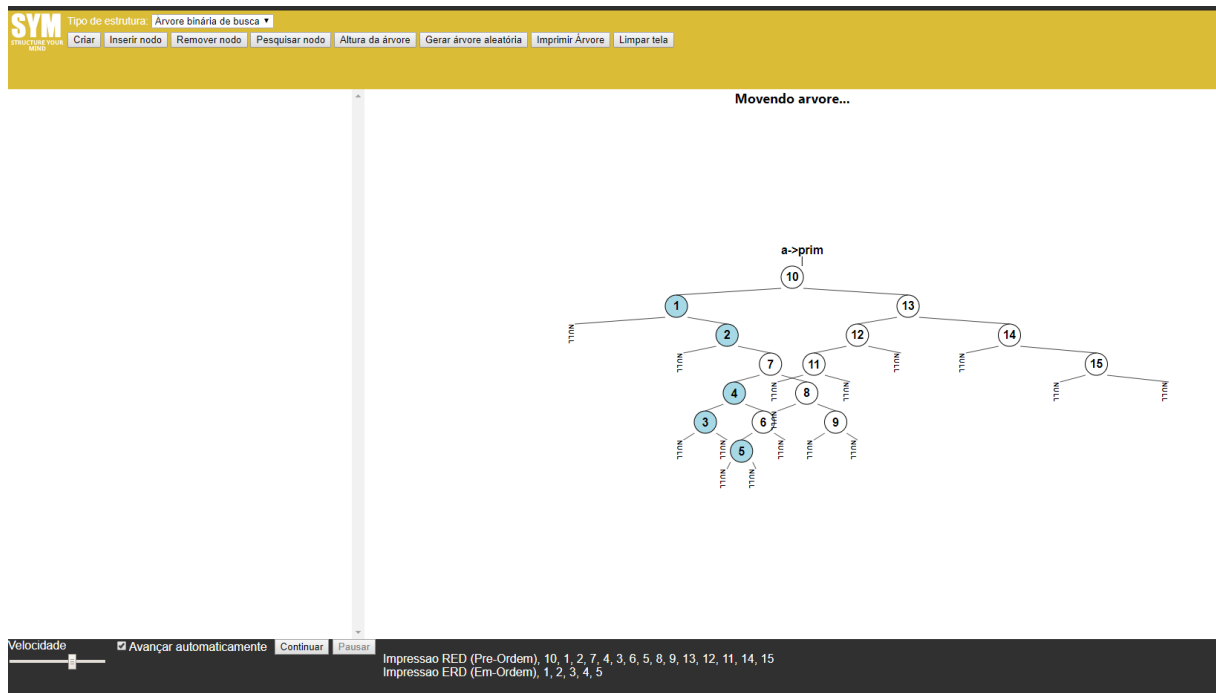


Figura 30: Impressão e Percurso pela árvore

Fonte: Autor

### 5.1.9 Controle da velocidade

Por todo o programa, estão espalhadas pausas que permitem a visualização do usuário antes que o sistema faça uma outra operação, por exemplo: no algoritmo da Tabela 13, de impressão e percurso da árvore, cada vez que o nó é impresso, também é realçado com a cor azul. A função “svg\_paint\_no”, expressada na Tabela 14, realça o nó com a cor especificada. Ela possui um comando para que o programa espere uma determinada quantidade de tempo, neste caso controlado pelo parâmetro de velocidade (em milissegundos).

Tabela 14: Algoritmo para realçar o nó através da cor da classe especificada.

Fonte: Autor

```

1 . async function svg_paint_no(No, classe, cascade, wait = true) {
2 .     if (No !== null) {
3 .         if (wait === true) {
4 .             await sleep(velocidade);
5 .         }
6 .         No.svg_nopont.circulo.removeClass('noNormal');
7 .         No.svg_nopont.circulo.removeClass('noDisabled');
8 .         No.svg_nopont.circulo.removeClass('noPath');
9 .         No.svg_nopont.circulo.removeClass('noSuccess');
10 .        No.svg_nopont.circulo.removeClass('noWarning');
11 .        No.svg_nopont.circulo.removeClass('noHeight');
12 .        No.svg_nopont.circulo.addClass(classe);
13 .        if (cascade === true) {
14 .            svg_paint_no(No.no_esquerda, classe, true, false);
15 .            svg_paint_no(No.no_direita, classe, true, false);
16 .        }
17 .    }

```

```
18. }
```

Se o parâmetro “wait” for verdadeiro, a função “sleep” é chamada e serão esperados os milissegundos de acordo com a variável de escopo global, “velocidade”.

Tabela 15: Função de espera que controla a velocidade do programa.

Fonte: Autor

```
1. async function sleep(ms) {
2.   do {
3.     await new Promise(resolve => setTimeout(resolve, ms));
4.   }
5.   while (paused === true);
6.   return;
7. }
```

Quando o usuário clica no botão pausar, assincronamente a variável de escopo global “paused” é convertida para verdadeira, dessa forma, o programa fica executando um laço de repetição com o comando de espera de tempo do Javascript, “setTimeout” (linhas 2 a 5 da Tabela 15), este laço só é interrompido quando, o usuário clica no botão continuar e assincronamente, o valor de “paused” é convertido para falso. Por ser um laço de repetição “do...while”, é garantida a espera de tempo pelo menos uma vez, o que permite que o usuário visualize as mudanças acontecendo na árvore.

## 5.2 TECNOLOGIAS UTILIZADAS

As tecnologias utilizadas na implementação dos novos módulos do sistema foram as mesmas utilizadas por SOUZA (2019, p.44), HTML, CSS e Javascript. Também foram utilizados alguns frameworks que facilitaram a reutilização do código, aumentando a produtividade. Os Frameworks utilizados foram o JQuery para manipulação dos elementos em geral e o SVG para a manipulação dos elementos gráficos.

### 5.2.1 HTML

O HyperText Markup Language (HTML) é uma linguagem de marcação (implementada através de tags), utilizada na criação de páginas web.

Utilizando esta linguagem foram criados o layout e os principais elementos como botões, cabeçalho, rodapé e as divisões centrais do sistema. O código pode ser verificado na Tabela 16.

Tabela 16: Trecho em HTML do arquivo arvore.html.

Fonte: Autor.

```

1 . <header id="barra-superior">
2 .     <div id="logo"></div>
3 .     <div id="controles">
4 .         Tipo de estrutura:
5 .         <select onchange="b_mudararvore_click()" id="tipo-lista">
6 .             <option value="BB">Árvore binária de busca</option>
7 .             <option value="AVL">Árvore AVL</option>
8 .             <option value="lse">Lista simplesmente encadeada
desordenada</option>
9 .             <option value="lse_ord">Lista simplesmente encadeada
ordenada</option>
10.             <option value="lde">Lista duplamente encadeada</option>
11.             <option value="pilha">Pilha</option>
12.             <option value="fila">Fila</option>
13.         </select>
14.         <br />
15.         <div id="botoes">
16.             <button id="criar" onclick="b_criar_click()">Criar</button>
17.             <button id="adicionar" onclick="b_adicionar_click()"
disabled>Inserir no</button>
18.             <button id="remover" onclick="b_remover_click()" disabled>Remover
no</button>
19.             <button id="pesquisar" onclick="b_pesquisar_click()"
disabled>Pesquisar no</button>
20.             <button id="altura" onclick="b_altura_click()" disabled>Altura da
árvore</button>
21.             <button id="gerar" onclick="b_gerar_click()" disabled>Gerar árvore
aleatória</button>
22.             <button id="imprimir" onclick="b_imprimir_click()"
disabled>Imprimir Árvore</button>
23.             <button id="limpar" onclick="b_limpar_click()">Limpar
tela</button>
24.             <button id="toggle" onclick="b_Toggle_Columns()">Esconder
Laterais</button>
25.         </div>
26.     </div>
27. </header>

```

## 5.2.2 CSS

Cascade Style Sheets (CSS) implementa o código de estilo das páginas web, é a principal linguagem responsável pela estilização dos elementos em HTML. Possibilita a reutilização dos estilos e permite que o código de HTML seja mais limpo.

Na Tabela 17 é possível perceber as classes que estilizam as cores dos nós, nas linhas 1 a 23. Nas linhas 25 a 41 estão definidos elementos de estilos próprios para as animações dos elementos gráficos.

Tabela 17: Trecho em CSS do arquivo estilo-arvore.css.

Fonte: Autor.

```

1 . .noNormal {
2 .     fill: transparent;
3 . }
4 .
5 . .noDisabled {

```

```

6 .      fill: #D3D3D3;
7 . }
8 .
9 . .noPath {
10.      fill: #ADD8E6;
11. }
12.
13. .noSuccess {
14.      fill: #90EE90;
15. }
16.
17. .noWarning {
18.      fill: #FF6347;
19. }
20.
21. .noHeight {
22.      fill: #FFFF00;
23. }
24.
25. .svgOp {
26.      animation-name: FadeIn;
27.      animation-iteration-count: 1;
28. }
29.
30. @keyframes FadeIn {
31.      0% {
32.          opacity: 0;
33.      }
34.
35.      99% {
36.          opacity: 0;
37.      }
38.
39.      100% {
40.          opacity: 1;
41.      }
42. }

```

### 5.2.3 Javascript

Javascript é uma linguagem de programação que permite o controle dinâmico, executado no navegador do usuário, dos elementos da página. As Tabelas 4 a 15 apresentam exemplos de algoritmos desenvolvidos em Javascript.

### 5.2.4 JQuery

JQuery é uma biblioteca de reutilização de funções desenvolvidas em Javascript, permite a otimização de produtividade fornecendo ao desenvolvedor as funções gerais mais comuns utilizadas em Javascript para manipulação de elementos.

Na Tabela 12, linha 22, é possível, através do método “text()” do JQuery, utilizar apenas uma linha de comando para mudar o texto de um elemento html.

### 5.2.5 SVG

O SVG é outra biblioteca escrita em Javascript para a reutilização de código, focada na manipulação dos elementos gráficos em HTML, com tag “<svg>”. Através desta biblioteca



foi possível desenhar os nós, números, as linhas e os elementos “NULL”, bem como transportar, colorir e rotacionar estes elementos. Na Tabela 6, linhas 41 a 45, é possível observar o código para a criação, movimentação e rotação com animações do elemento “NULL” direito de um nó.

## 6 CONCLUSÃO

É possível concluir a importância do uso das metodologias ativas na educação, e de ferramentas de apoio ao ensino, especialmente naquelas disciplinas que possuem maior índice de reprovação, como é o caso da disciplina de estruturas de dados nos cursos de computação. Estas ferramentas precisam ter como principal objetivo a interação individual do aluno com o conteúdo, onde este será engajado no aprendizado quando o ensino é feito de forma gradual no seu próprio ritmo.

Três grandes dificuldades se apresentaram no desenvolvimento do trabalho, a primeira, foi a de determinação da medida de espaçamento entre os nós da árvore, devido ao aumento exponencial do número de nós conforme a altura da árvore cresce, era importante encontrar uma fórmula que permitisse a organização dos nós por nível de altura sem que eles ficassem muito afastados, o que comprometeria a visão da árvore, ou que ficassem muito juntos e embaralhados. Após inúmeras tentativas, a abordagem de maior sucesso foi a de uso de fronteiras. Os nós da fronteira são aqueles nós que se encontram nas extremidades de cada nível, estes nós devem ser afastados para fora o máximo possível, enquanto os nós do interior devem ser afastados um dos outros o mínimo possível. Isto se deve à conclusão de que um nó do interior que se afasta demais para um lado tende a se aproximar demais do outro.

A segunda grande dificuldade foi devida à utilização da linguagem Javascript para o desenvolvimento das estruturas das árvores. Conforme visto no trabalho, as árvores binárias de busca são estruturas implementadas por algoritmos recursivos, onde os parâmetros (e os nós) são passados por referência e os nós são alterados dentro da iteração. O Javascript não possui a funcionalidade de passagem de método por referência, apenas a de passagem de um objeto que será modificado tanto dentro quanto fora do escopo da função onde ele foi passado, o grande problema se deve a que, se tratando de uma árvore, cada objeto tem pais e filhos, e se é passado o filho de um objeto para a função de Javascript, a referência desse filho não é alterada fora do escopo da função, sendo necessário, a passagem do objeto pai e que seja modificado dentro da função, o filho. É importante levar em consideração que cada objeto tem mais de um filho, então, se o pai era passado para a função, dentro da função era necessário tratar com cuidado qual era o filho que seria modificado. A implementação do código se deu de forma mais complexa e com menos reutilização por conta desta particularidade da linguagem Javascript, e a implementação das estruturas de árvores binárias de busca e de balanceamento de altura se tornou ainda mais desafiadora.

A terceira grande dificuldade se deu à pouca disponibilidade de reutilização de código ou metodologias por parte do código disponibilizado pelo desenvolvedor original do SYM, pois o seu código não seguia os bons costumes de desenvolvimento, e portanto, foi de muita dificuldade a aprendizagem do funcionamento do seu código, se tornando mais viável o desenvolvimento desde zero (desta vez seguindo os bons costumes de desenvolvimento), de todas as funcionalidades para a implementação das árvores binárias de busca e de balanceamento de altura. Esta foi uma grande dificuldade porque impediu que a grande maioria dos aprendizados de desenvolvimento feitos por SOUZA (2019) pudessem servir como base para o desenvolvimento dos novos módulos.

São propostos como trabalhos futuros a implementação da árvore do tipo B, árvore rubro-negra, e da simulação da depuração do código na caixa lateral esquerda, e da simulação do uso da memória na caixa lateral direita, da mesma forma que foi implantado no SYM para as estruturas de dados anteriores (Pilha, fila, lista encadeada ordenada e desordenada), estas caixas laterais são essenciais para o aprendizado pois permitem ao aluno visualizar programaticamente como a linguagem lida com as estruturas.

Também é proposto que a ferramenta seja levada à sala de aula e apresentada pelos professores, disponibilizando-a para que os alunos também possam acessá-la conforme a sua necessidade para complementação do estudo. Os alunos devem ser entrevistados e sugestões de melhorias para a ferramenta devem ser consideradas e implementadas. Após a passagem de cada semestre e da utilização da ferramenta nas salas de aula, propõe-se a comparação estatística da taxa de aprovação dos alunos para verificar se a ferramenta obteve sucesso em melhorar a capacidade de aprendizado na disciplina, e desta forma, evitando a frustração e evasão por parte dos alunos.

## REFERÊNCIAS BIBLIOGRÁFICAS

ASCENSIO, Ana Fernandes Gomes; ARAÚJO, Graziela Santos de. *Estruturas de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++*. Pearson Prentice Hall, 2010. ISBN: 978-85-430-1456-2.

ASCENSIO, Ana Fernandes Gomes; CAMPOS, Edilene Aparecida Veneruchi de. *Fundamentos da programação de computadores*. Prentice Hall, 2002. ISBN: 978-85-645-7416-8.

BACKES, André. *Árvore AVL*. 2019. Faculdade de computação, Universidade Federal de Uberlândia. Disponível em: <<http://www.facom.ufu.br/~backes/gsi011/Aula11-ArvoreAVL.pdf>>. Acesso em 12 dez. 2019.

BAUER, Edimar Jacob. *Árvore binária de pesquisa oculta com crescimento dinâmico*. 2018. Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

BRASIL, *Causas da evasão de alunos nos cursos de graduação presencial da UFPE*. Universidade Federal de Pernambuco. Disponível em: <[https://www.ufpe.br/documents/38954/371376/r\\_evaso\\_16.pdf/53642e52-41fb-4b43-b098-98db6a470176](https://www.ufpe.br/documents/38954/371376/r_evaso_16.pdf/53642e52-41fb-4b43-b098-98db6a470176)>. Acesso em: 30 nov. 2019.

BRASIL, *Censo da educação superior 2017*. INEP. Disponível em: <<http://portal.mec.gov.br/docman/setembro-2018-pdf/97041-apresentac-a-o-censo-superior-ultimo/file>>. Acesso em: 30 nov. 2019.

CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. *Introdução a Estruturas de Dados*. Elsevier, 2016. ISBN: 978-85-352-8345-7.

CERP IoT – INTERNET OF THINGS EUROPEAN RESEARCH CLUSTER. *Internet of Things: Strategic Research Roadmap*. Disponível em <[http://www.internet-of-things-research.eu/pdf/IoT\\_Cluster\\_Strategic\\_Research\\_Agenda\\_2009.pdf](http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2009.pdf)>. Acesso em 01 de dez. 2019.

COSTA, Raphael Raniere de Oliveira; MEDEIROS, Soraya Maria de; MARTINS, José Carlos Amado; MENEZES, Rejane Maria Paiva de; ARAÚJO, Marília Souto de. *O uso da simulação no contexto da educação e formação em saúde e enfermagem: Uma reflexão acadêmica*. Revista Espaço para a Saúde, Londrina, v.16, n. 1, p. 61, jan. 2005.

CUNHA, Adilson Marques da; SILVA, Gláucia Braga e; MONTE-MOR, Juliano de Almeida; DOMICIANO, Marco Antonio Pizani; VIEIRA, Ricardo Godoi. *Estudo de Caso abrangendo o Ensino Interdisciplinar de Engenharia de Software*. Disponível em: <[https://www.researchgate.net/profile/Adilson\\_Cunha/publication/228872356\\_Estudo\\_de\\_Caso\\_abrangendo\\_o\\_Ensino\\_Interdisciplinar\\_de\\_Engenharia\\_de\\_Software/links/02e7e5149c6489c821000000.pdf](https://www.researchgate.net/profile/Adilson_Cunha/publication/228872356_Estudo_de_Caso_abrangendo_o_Ensino_Interdisciplinar_de_Engenharia_de_Software/links/02e7e5149c6489c821000000.pdf)>. Acesso em: 01 dez. 2019.

DALLAVALLE, Silvia Inês; CAZARINI, Edson Walmir. *Regras do Negócio, um fator chave*

*de sucesso no processo de desenvolvimento de Sistemas de Informação*. Disponível em: <[https://s3.amazonaws.com/academia.edu.documents/30964332/ENEGEP2000\\_E0237.pdf?response-content-disposition=inline%3B%20filename%3DRegras\\_do\\_negocio\\_fator\\_chave\\_de\\_sucesso.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20191201%2Fus-east-1%2Fs3%2Faws4\\_request&X-Amz-Date=20191201T015529Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=f3a6ac7cc49bb989a94e2efcba591bd5d7549e641c5c57f1edfcb2d31eccc2a3](https://s3.amazonaws.com/academia.edu.documents/30964332/ENEGEP2000_E0237.pdf?response-content-disposition=inline%3B%20filename%3DRegras_do_negocio_fator_chave_de_sucesso.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20191201%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20191201T015529Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=f3a6ac7cc49bb989a94e2efcba591bd5d7549e641c5c57f1edfcb2d31eccc2a3)>. Acesso em: 01 dez. 2019.

FLANAGAN, David. *Javascript, o guia definitivo*. 6 ed. Bookman, 2013. ISBN: 978-0-596-80552-4.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. *Lógica de programação: a construção de algoritmos e estruturas de dados*. 3 ed. Prentice Hall, 2005. ISBN: 85-7605-024-2.

GEIB; Carlos Felipe. *O uso de simuladores de redes de computadores nos cursos de tecnologia da informação*. Disponível em: <[https://www.univates.br/editora-univates/media/publicacoes/215/pdf\\_215.pdf#page=37](https://www.univates.br/editora-univates/media/publicacoes/215/pdf_215.pdf#page=37)>. Acesso em: 01 dez. 2019.

KAPP, Karl M. *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education*. Peiffer, 2012. ISBN: 978-1-118-09634-5.

LAUREANO, Marcos. *Estrutura de dados com Algoritmos e C*. Brasport, 2009. ISBN: 978-85-7452-355-2.

LIMA JUNIOR, José Augusto Teixeira de; VIEIRA, Carlos Eduardo Costa; VIEIRA, Priscila de Paula. *Dificuldades no processo de aprendizagem de Algoritmos: uma análise dos resultados na disciplina de AL1 do Curso de Sistemas de Informação da FAETERJ - Campus Paracambi*. O Globo. Disponível em: <<http://revistas.unifoa.edu.br/index.php/cadernos/article/viewFile/293/346>>. Acesso em: 30 nov. 2019.

LUCRÉDIO, Daniel. *Uma Abordagem Orientada a Modelos para Reutilização de Software*. 2009. Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional.

MORÁN, José. *Mudando a educação com metodologias ativas*. Disponível em: <<http://rh.newwp.unis.edu.br/wp-content/uploads/sites/67/2016/06/Mudando-a-Educacao-com-Metodologias-Ativas.pdf>>. Acesso em: 30 nov. 2019.

NETTO, Dorgival; MEDEIROS, Luiz Mario; PONTES, Daniel de; MORAIS, Edilson de. *Game Logic: Um jogo para auxiliar na aprendizagem de lógica de programação*. Disponível em: <<https://sol.sbc.org.br/index.php/wei/article/view/3546>>. Acesso em: 01 dez. 2019.

SANTOS, Walter dos. *Uso de simuladores como ferramenta no ensino e aprendizagem de redes de computadores em um novo modelo de ensino*. 2016. Dissertação de Mestrado

apresentada ao Programa de Pós-graduação stricto sensu no curso de Mestrado Profissional em Sistemas de Informação e Gestão do Conhecimento da Universidade FUMEC, como requisito para obtenção do título de Mestre em Sistemas de Informação e Gestão do Conhecimento.

SILVA, Gerlinda Hermita Nobre da; FONSECA, Maria Lenita Cavalcante da. *A importância da tecnologia como suporte pedagógico no processo ensino aprendizagem*. Disponível em: <<http://bdta.ufra.edu.br/jspui/handle/123456789/243>>. Acesso em: 01 dez. 2019.

SOUZA, Alexandre Mota. *SYM: Uma aplicação para auxílio no ensino da disciplina de Estrutura de Dados*. 2019. Monografia apresentada ao curso de Ciência da Computação da Universidade Veiga de Almeida, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

TSAKALIDIS, Athanasios K. *AVL-Trees for Localized Search*. 1985. Disponível em <<https://www.sciencedirect.com/science/article/pii/S0019995885800346>>. Acesso em 03 de dez. 2019.

VELASCO, Vanessa Fajardo e Clara. *Metade dos calouros na faculdade em 2010 trocaram de turma, de instituição ou abandonaram o curso*. O Globo. Disponível em: <<https://g1.globo.com/educacao/noticia/metade-dos-calouros-na-faculdade-em-2010-trocaram-de-turma-de-instituicao-ou-abandonaram-o-curso.ghtml>>. Acesso em: 30 nov. 2019.

VOSS, G. B.; MEDINA, R. D.; ARAUJO, F. V.; NUNES, F. B.; OLIVEIRA, T. *Proposta de utilização de laboratórios virtuais para o ensino de redes de computadores: articulando ferramentas, conteúdos e possibilidades*. Novas Tecnologias na Educação. Disponível em: <<https://seer.ufrgs.br/renote/article/view/36397>>. Acesso em: 01 dez. 2019.