CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U nº 198, de 14/10/2016
ASSOCIAÇÃO EDUCACIONAL LUTERANA DO BRASIL

Gedilson Pessoa da Silva

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA AUXILIAR NO ENSINO DE ÁRVORE BINÁRIA NA DISCIPLINA DE ESTRUTURA DE DADOS

Palmas – TO
2019

Gedilson Pessoa da Silva

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA AUXILIAR NO ENSINO DE ÁRVORE BINÁRIA NA DISCIPLINA DE ESTRUTURA DE DADOS

Trabalho de Conclusão de Curso II (TCC II) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: M.e Fabiano Fagundes.

Palmas – TO
2019

Gedilson Pessoa da Silva

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB PARA AUXILIAR NO ENSINO DE ÁRVORE BINÁRIA NA DISCIPLINA DE ESTRUTURA DE DADOS

Trabalho de Conclusão de Curso II (TCC II) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof.: M.e Fabiano Fagundes.

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. M.e Fabiano Fagundes
Orientador
Centro Universitário Luterano de Palmas – CEULP

_____

Profª. M.e Madianita Bogo Marioti
Centro Universitário Luterano de Palmas – CEULP

_____

Profª. M.e Heloise Acco Tives Leão
Centro Universitário Luterano de Palmas – CEULP

Palmas – TO
2019

DEDICATÓRIA

Dedico este trabalho de conclusão de curso, em primeiro lugar, a DEUS; aos meus queridos e amados pais que me deram a oportunidade de realizar mais este passo; à minha esposa e aos meus filhos que sempre estão comigo.

AGRADECIMENTOS

Este ano, 2019, é muito especial, pois depois de uma longa e árdua caminhada, chego ao final desta etapa, com sentimento de vitória. Demorou? Talvez sim! Talvez não! A única certeza é que Deus esteve comigo o tempo todo, o tempo todo Deus esteve comigo. Por isso, agradeço a Ele em primeiro lugar por mais esse passo dado em minha vida, por mais esta vitória conquistada.

Agradeço também à minha esposa por me suportar e procurar entender quando angustiado estava, por me ajudar no que podia, mesmo sem entender o projeto. Meus filhos lindos, Gabriel Arthur e Nícolas Arthur, por existirem em minha vida. Apesar de pequenos, contribuíram, mesmo sem saber, quando me beijavam e sorriam para mim. Papai ama muito esses dois homenzinhos.

Agradeço imensamente aos meus pais, Arthur Pessoa e Genesia Gomes. Pai, mãe, vocês são instrumentos de Deus para abençoar a minha vida. Serei grato até o meu último suspiro, por vocês terem me dado a oportunidade de hoje estar aqui, escrevendo essas palavras, simples, porém; de coração.

A palavra de DEUS diz em êxodo 20:12 "Honra teu pai e tua mãe, a fim de que tenhas vida longa na terra que o Senhor, o teu Deus, te dá". Formei e entrego a vocês a honra de ter adquirido este diploma. Esta vitória não é só minha, também é de vocês.

Aos professores: Fabiano, Cristina, Parcilene, Madianita, Jackson, Fernando, Ricardo, Teresa, Andrés, dentre outros que não me lembro. Recebam meus agradecimentos, por me ajudarem a crescer intelectualmente nesta instituição. Obrigado pelos conselhos e incentivos mesmo que indiretos muitas das vezes.

Durante minha trajetória pela a instituição CEULP/ULBRA criei várias amizades, porém duas especiais, que não poderia chegar ao final desse trajeto sem citá-las. Primeiro o Robinho, você mesmo meu amigo, infelizmente você se foi, mais ficou em meu coração. Obrigado por inúmeras vezes vir até minha casa para me ensinar a programar em ASP.NET. Suas dicas foram importantes para construção da minha base de conhecimento. Que Deus o tenha dentro de sua misericórdia e guarde-o em um lugar especial. Ao meu amigo, Marcio Fernandes, gente muito boa. Muita coisa boa levarei durante minha passagem por esta terra e uma delas será nossa amizade.

A toda minha família e amigos que direta e indiretamente contribuíram e compartilharam comigo meu sonho em formar. Por fim, agradeço à Débora Silva que me ajudou a corrigir e estruturar meu projeto. Que Deus derrame toda a graça Dele sobre a vida de todos.

RESUMO

Árvores binárias são estruturas de dados utilizadas na programação para armazenar informações de maneira mais eficiente. Uma árvore é uma extensão de grafos, com a característica de que cada nó (ou vértice) contém no máximo dois filhos, raízes de novas sub-árvores, ligados à ele através de arestas ou arcos. Árvore binária é bastante utilizada na computação para, por exemplo, armazenamento de grandes volumes de dados na memória do computador, trabalhar com compressão de dados através de algoritmos como o de Huffman, organizar informações de forma que facilite a inserção, exclusão e busca de forma mais eficaz, entre outras. A proposta deste trabalho foi mostrar o conceito de árvore binária, bem como o seu funcionamento através de uma ferramenta online, que possibilitasse criar, inserir, excluir, buscar nós e ilustrar graficamente ao aluno todos os conceitos relacionados ao tema. Para desenvolvimento da ferramenta foi utilizado o framework Bootstrap para criação da interface, Javascript para implementação dos códigos e para desenhar os elementos gráficos foi utilizado o framework svg.js. O ambiente foi desenvolvido tendo como objetivo principal servir como ferramenta auxiliar no estudo e aprendizado de árvore binária na disciplina estrutura de dados, permitindo ao estudante criar, de forma dinâmica, nós e executar funções relacionadas ao conceito abordado.

Palavras-chave: Árvore Binária, Árvore Binária de Busca, Estruturas de Dados, Nós.

ABSTRACT

Binary trees are data structures used in programming to store information more efficiently. A tree is an extension of graphs, with the characteristic that each node (or vertex) contains at most two children, roots of new sub-trees, connected to it through edges or arcs. Binary tree is widely used in computing to, for example, store large volumes of data in computer memory, work with data compression through algorithms like Huffman, organize information in a way that facilitates the insertion, deletion and search of form more effective, among others. The purpose of this work was to show the concept of binary tree, as well as its operation through an online tool, which allows to create, insert, delete, search for nodes and graphically illustrate to the student all concepts related to the theme. For the development of the tool was used the Bootstrap framework to create the interface, Javascript to implement the codes and to draw the graphics was used the framework svg.js. The environment was developed with the main purpose of serving as an auxiliary tool in the study and learning of binary tree in the data structure discipline, allowing the student to dynamically create nodes and perform functions related to the concept addressed.

Keywords: Binary Tree, Binary Search Tree, Data Structures, Nodes.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CEULP/ULBRA	Centro Universitário Luterano de Palmas
HTML5	Linguagem de Marcação de Hipertexto
IDE	Ambiente de Desenvolvimento Integrado
SVG	Gráficos Vetoriais Escaláveis

SUMÁRIO

1 INTRODUÇÃO

Na computação, dados, quando organizados e armazenados na memória do computador de forma compreensível, representam uma forma de estrutura de dados. Existem vários modelos de estrutura de dados na programação: arrays, filas, listas, grafos, árvores, entre outros. Estas são formas de organizar e armazenar dados logicamente, oferecendo otimização na recuperação das informações.

A forma como cada estrutura de dados organiza e armazena informações dentro de sua estrutura pode transformar tarefas complexas em tarefas extremamente simples. Assim, fica evidente que a escolha da estrutura certa para cada situação é o ponto fundamental para a obtenção do melhor resultado.

A árvore binária de busca são estruturas de dados não lineares, que permitem localizar valores contidos em sua estrutura percorrem-se vários caminhos em diferentes direções. Uma das principais características desta estrutura refere-se ao fato de ser uma estrutura de dados hierárquica, permitindo, assim, organizar e buscar elementos de forma mais rápida se comparada às demais estruturas de dados.

A estrutura de dados, árvore binária, é muito utilizada na computação e, segundo Lafore (1999, p. 285), a estrutura combina vantagens do *array* ordenado, pois permite realizar buscas de forma rápida e da lista encadeada, permitindo assim realizar operações de inserção e exclusão de itens rapidamente. Sua estrutura pode ser utilizada para representar expressões matemáticas e estruturas de pastas, bem como na criação de jogos, na inteligência artificial, em interfaces gráficas e em diversas outras situações. A estrutura é formada basicamente por um conjunto finito de elementos, geralmente representados por círculos e nomeados de nós. O primeiro elemento desse conjunto recebe o nome de raiz e é a partir desse nó que desencadeia toda a estrutura da árvore.

De forma análoga, no mundo real utiliza-se o conceito de árvores em situações como, por exemplo, em uma árvore genealógica para organizar descendentes familiares, onde a primeira geração representa a raiz e as demais gerações, filhos, netos, bisnetos e demais descendentes representam os nós da árvore. Outro exemplo a ser citado é a compressão de dados através do algoritmo de Huffman que se utiliza da estrutura de árvore para organizar os valores obtidos na compressão. Uma característica peculiar dessa técnica de compressão de dados está no fato de a árvore ser criada de baixo para cima, diferente do apresentado em estrutura de dados que é criada de cima para baixo.

As diversas formas de armazenamentos de dados em estruturas de dados são apresentadas e compreendidas na disciplina Estrutura de Dados. Através desta disciplina é possível estudar, compreender os conceitos comuns e particulares de cada estrutura e implementá-los para melhor assimilação.

Neste trabalho foi proposto o desenvolvimento de uma aplicação *online* que permita ao docente da disciplina Estrutura de Dados apresentar a seus alunos a estrutura de árvore binária de busca de forma mais dinâmica e intuitiva, permitindo, com isso, visualizar o conceito teórico aplicado. A ferramenta permite ao aluno aprender de forma gradual, obtendo informações relacionadas à árvore, como: profundidade, grau da árvore, quantidade de nós e demais informações pertinentes à estrutura.

Inicialmente, questionou-se se era possível desenvolver uma ferramenta utilizando-se das tecnologias de desenvolvimento de aplicações web, que permitisse desenhar árvore binária de busca, para auxiliar no aprendizado do aluno na Disciplina Estrutura de Dados. A partir daí buscou-se desenvolver uma ferramenta ilustrativa, que permitisse a criação de árvore binária de busca para ser utilizada pelo professor e pelo aluno na disciplina de Estrutura de Dados, a fim de auxiliar no aprendizado deste referido conceito. Para tanto, foi necessário desenvolver um referencial teórico dos conceitos envolvidos para então dar início ao desenvolvimento do sistema que, em suma, consiste em uma área onde é possível desenhar a árvore binária para demonstrar informações relacionadas à árvore que está sendo desenhada, como: profundidade, altura, grau da árvore etc.

É importante ressaltar que a ferramenta permitirá ao professor dar ênfase ao que está sendo ensinado de maneira mais dinâmica, além de permitir que o aluno tenha uma visão mais abrangente do conteúdo, uma vez que, permitirá desenhar e observar seu comportamento mediantes operações de criação e exclusão de nós na área de desenho.

Esse trabalho se subdivide da seguinte forma: a seção 2 traz a revisão de literatura com os conceitos e as definições das tecnologias utilizadas nesse trabalho. A seção 3 traz os materiais e métodos utilizados no desenvolvimento desse trabalho. Na seção 4 são abordados os resultados e a discussão. Por fim, tem-se a conclusão na seção 6 seguida das referências bibliográficas.

2 REFERENCIAL TEÓRICO

Nesta seção serão apresentados os conceitos relevantes sobre árvore binária de busca que é uma estrutura de dados bastante utilizada na programação para organizar dados na memória do computador.

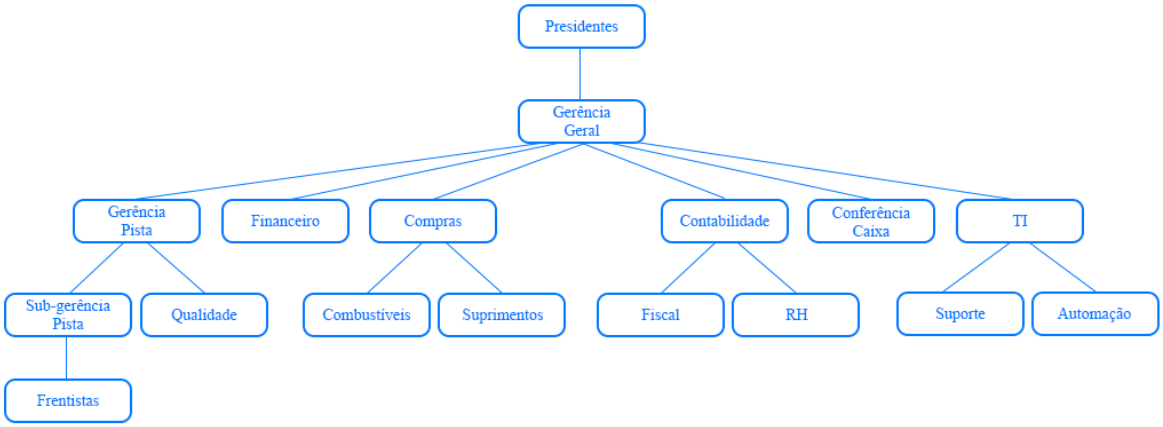
2.1 Árvores

De acordo com Veloso (1986), árvore pode ser definida como uma estrutura de dados que se caracteriza pela relação existente entre os dados, denominados nós, ou seja, é uma relação de hierarquia onde um conjunto de dados é hierarquicamente subordinado a outro.

Segundo Horowitz (1984), entende-se por árvore um conjunto finito de um ou mais nós de maneira que, obrigatoriamente, um deles seja denominado raiz, de modo que cada nó da árvore é a raiz de uma sub-árvore. Em uma definição simples e objetiva, árvore é um conjunto de nós que se relacionam, possuindo uma hierarquia de pai para filho.

Pode-se facilmente visualizar, de acordo com Szwarcfiter; Markenzon (1994), o conceito de árvore em um organograma de uma empresa, que é representado de forma hierarquizada. A Figura 1 representa o organograma de uma empresa no ramo de postos de combustíveis.

Figura 1 - Departamentos de uma empresa



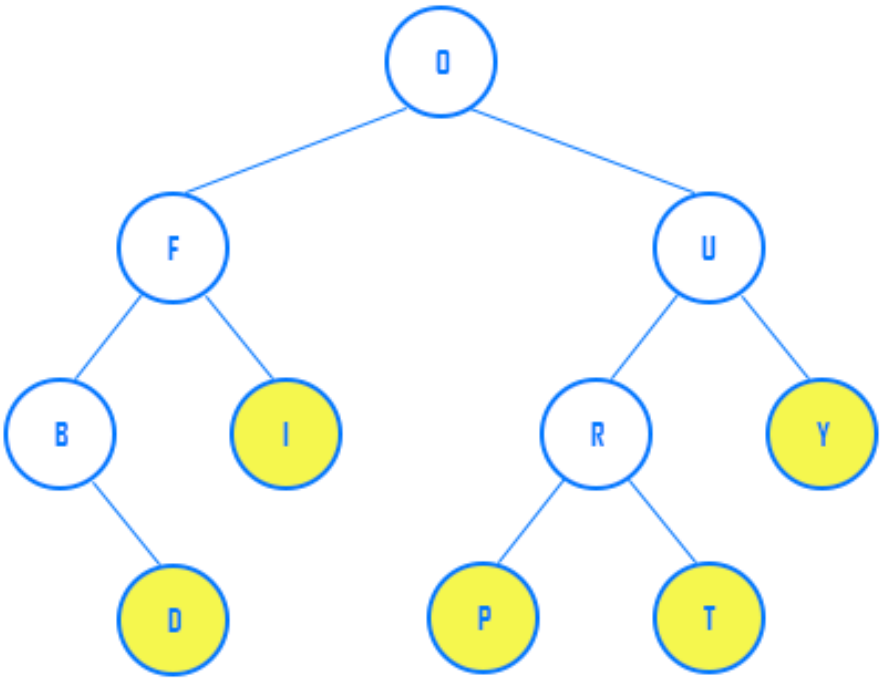
O organograma foi implementado conforme estrutura departamental da empresa ‘X’. Os departamentos, bem como sua hierarquia, estão definidos e organizados em uma árvore hierárquica, conforme ilustrado na Figura 1. O cargo mais alto da empresa representado pelo o nó raiz “Presidentes”, são ocupados pelos os proprietários que possui como subordinado direto o gerente geral. O gerente geral possui vários subordinados diretos e indiretos e vários departamentos, como pode ser observado no exemplo.

Nas seções a seguir, será apresentada a estrutura de dados árvore binária. Esta estrutura difere do conceito de árvore no quesito quantidade de filhos, uma vez que a quantidade de nós filhos de cada nó raiz está limitada à 2 filhos. Neste modelo de estrutura de dados cada nó poderá conter 0, 1 ou no máximo 2 filhos, sendo eles: filho esquerdo e filho direito.

2.1.1 Nó Folha

Nó é um ponto de conexão que liga as ramificações de uma árvore. Segundo Robert Lafore (1999, p. 289), "Um nó que não tem filhos é chamado de um nó com *folha* ou simplesmente *folha"*. A Figura 2 representa um exemplo de árvore e seus respectivos nós folhas.

Figura 2 - Ilustração de uma árvore binária – nó folhas



Na Figura 2, os nós "D", "I", "P", "T" e "Y" em destaque, são considerados nós folhas, pois contém grau igual a zero, ou seja, não possuem filhos.

Um nó de uma árvore é irmão de outro nó se os dois tiverem o mesmo pai, que é o nó ascendente. No exemplo, Figura 2, têm-se os seguintes nós irmãos: "F-U", "B-I", "R-Y" e "P-T". Na árvore todo nó, exceto o nó raiz possui apenas um único nó ascendente.

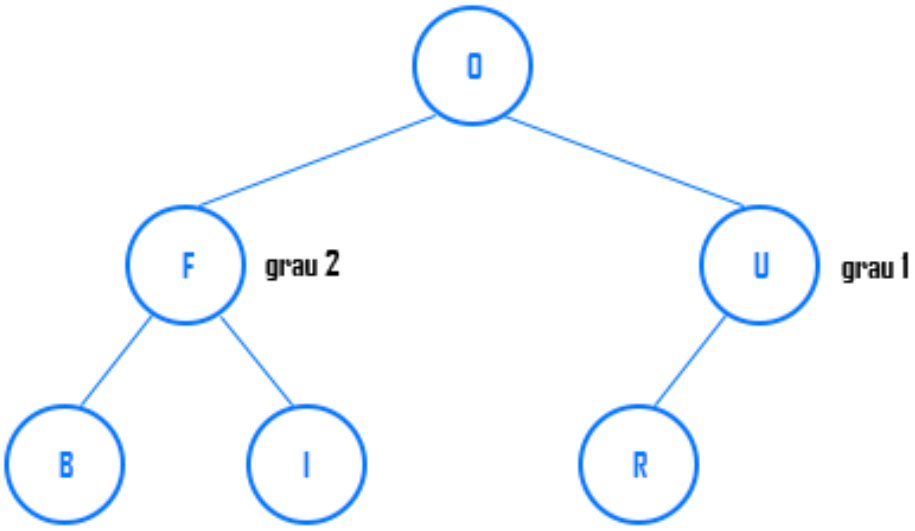
Dentro da estrutura da árvore o nó, obrigatoriamente, com exceção do nó raiz, possuirá apenas 1 nó pai. Como pode ser visto no exemplo, Figura 2, uma árvore é formada por sub-

árvores que respeitam as mesmas regras em todos os níveis. A partir do nó raiz têm-se as sub-árvores “F” e “U” que por sua vez geram novas sub-árvores a partir de seus respectivos nós.

2.1.2 Grau de um Nó

Segundo Laureano (2012, p.127), grau do nó ou grau de saída de um nó é a quantidade de nós filhos ou de sub-árvores do nó. No caso de árvores binárias o grau máximo de um dado nó será 2, visto que a quantidade máxima de filhos para cada nó da estrutura está limitado a 2. A Figura 3 representa um exemplo de grau de um nó.

Figura 3 - Grau de um nó

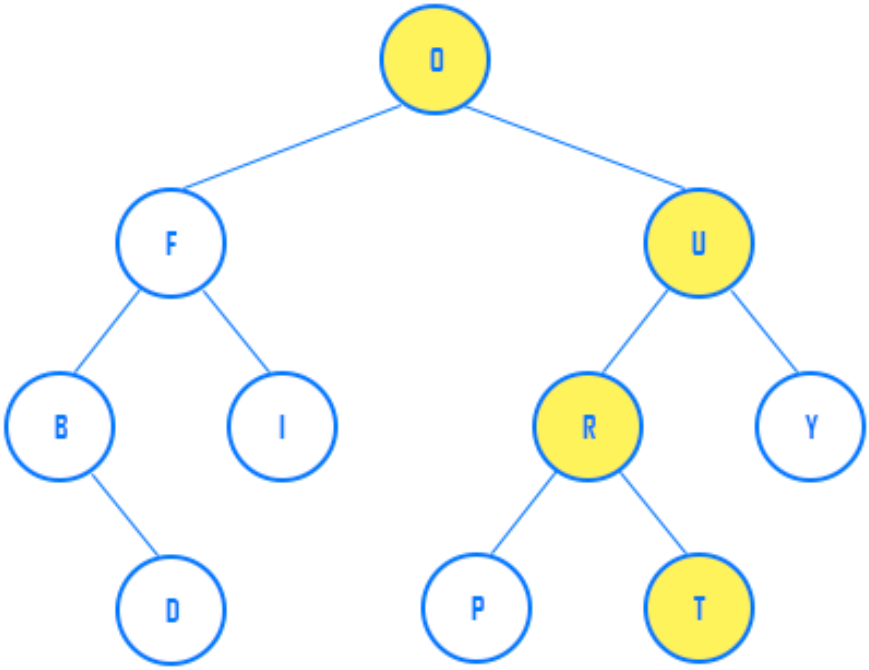


No exemplo, Figura 3, o grau do nó “F” é 2 devido possuir 2 filhos. No entanto, o grau do nó “U” é 1, pois possui apenas 1 filho. É evidente que na estrutura de dados árvore binária a quantidade de grau deverá ser 0, 1 ou no máximo 2.

2.1.3 Nó Ancestral

Segundo Ascencio; Araújo (2010, p. 281), nós ancestrais “são os nós que estão acima de um nó e possuem ligação direta ou indireta.”. Na árvore, todos os nós que estiverem no caminho entre o nó escolhido e o nó raiz, são denominados ancestrais. Na Figura 4 representa um exemplo de nós ancestrais de um dado nó.

Figura 4 - Ilustração de uma árvore binária – nó ancestral

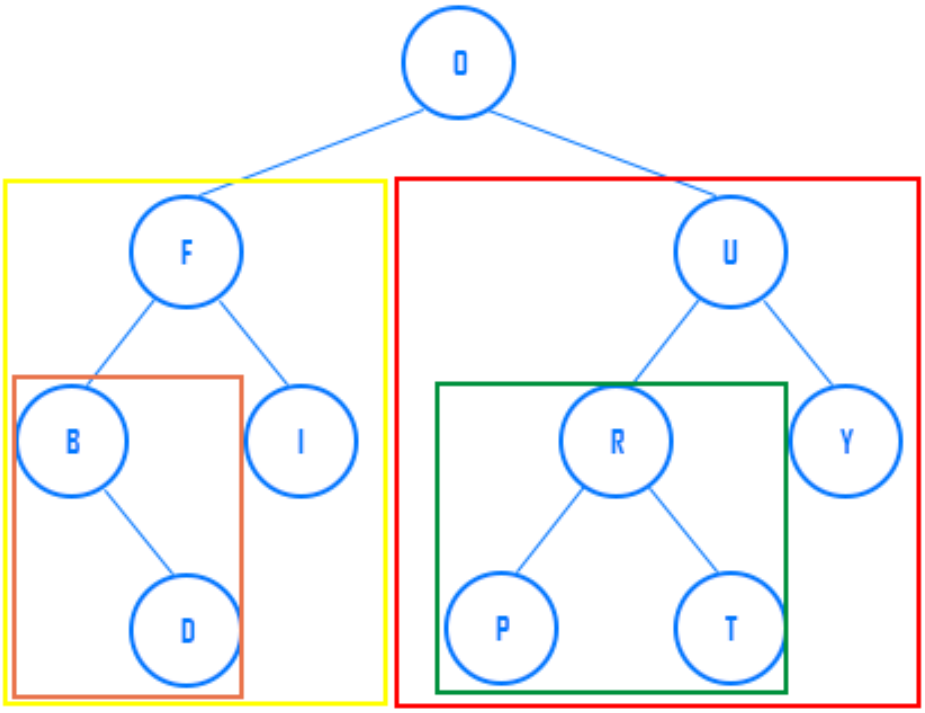


No exemplo, Figura 4, os nós “O”, “U” e “R” são considerados ancestrais do nó “T”, visto que estão interligados, fazendo parte do percurso que interliga o nó “T” ao nó raiz “O”.

2.1.4 Nó Descendente

Na definição de Ascencio; Araújo (2010, p. 281), nós descendentes “são os nós que estão abaixo de um nó e possuem ligação direta ou indireta”. De forma simplista, os nós descentes são todos os nós que pertencem à hierarquia de um determinado nó superior. A Figura 5 representa os nós descentes da árvore e de suas sub-árvores.

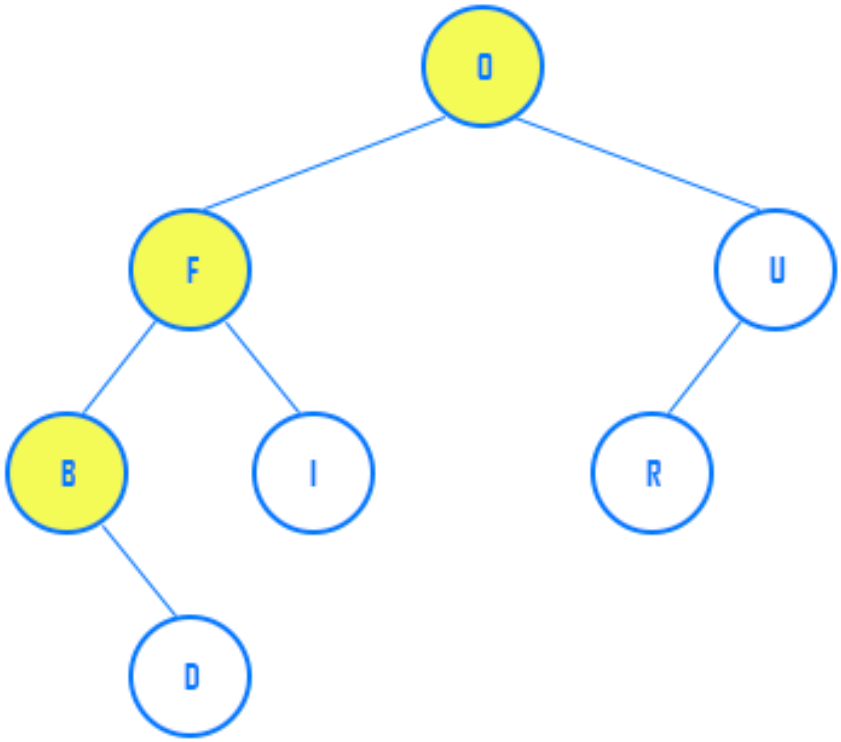
Figura 5 - Ilustração de uma árvore binária – nó descendente



No exemplo, Figura 5, os nós descendentes do nó raiz “O” são: “F”, “U”, “B”, “I”, “R”, “Y”, “D”, “P” e “T”; descendentes do nó “F”: “B”, “I” e “D”; descendentes do nó “B”: “D”; descendentes do nó “U”: “R”, “Y”, “P” e “T”; descendentes do nó “R”: “P” e “T”. Os nós “D”, “P”, “T” e “Y” são classificados como folhas da árvore e por terem grau zero não possuem descendentes.

Nó descendente esquerdo são todos os nós que estão abaixo do nó, possuindo ligação direta ou indireta, e que fazem parte da sub-árvore esquerda. A Figura 6 representa os descendentes esquerdo de um determinado nó.

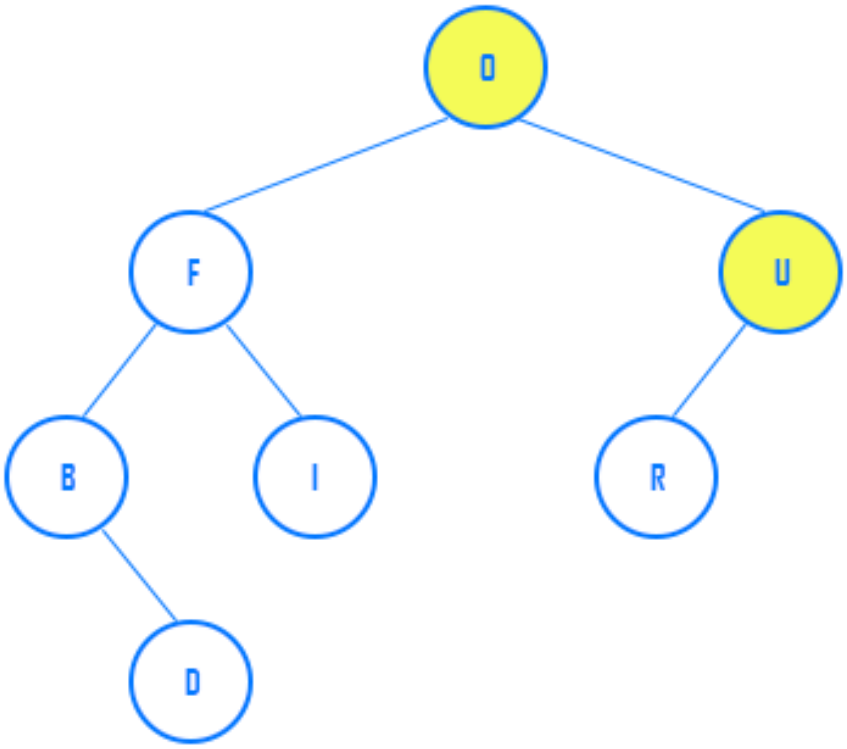
Figura 6 - Nó Descendente Esquerdo



Na Figura 6 é possível observar os descendentes do nó “O”, “F” e “B”. O nó “O” possui os seguintes descendentes esquerdos, “F” e “B” respectivamente. O nó “F” contém como descendente esquerdo o nó “B” e o nó “B” não possui descente esquerdo.

Nó descendente direito são todos os nós que estão abaixo do nó, possuindo ligação direta ou indireta, e que fazem parte da sub-árvore direita. A Figura 7 representa os nós descendentes direito de um dado nó.

Figura 7 - Nó Descendente Direito



Na Figura 7 é possível observar os descendentes do nó “O” e “U”. O nó “O” possui como descendente direito o nó “U”. O nó “U” não possui descendente direito como pode ser observado no exemplo ilustrado pela Figura 7.

2.2 Árvores Binária

Conforme afirmam Henderson (2009, p. 478) e Szwarcfiter (1994, p. 67), na estruturação de dados o tipo mais comum de árvore utilizada é a árvore binária.

Entende-se por árvore binária:

> “[..] um conjunto finito de elementos que está vazia ou é particionado em três subconjuntos disjuntos. O primeiro subconjunto contém um único elemento, chamado ***raiz*** da árvore. Os outros dois subconjuntos são em si mesmos árvores binárias, chamadas ***subárvores esquerda*** e ***direita*** da árvore original. Uma subárvore esquerda ou direita pode estar vazia. (TENENBAUM; LANGSAM; LANGSAM, 1995, p. 303).

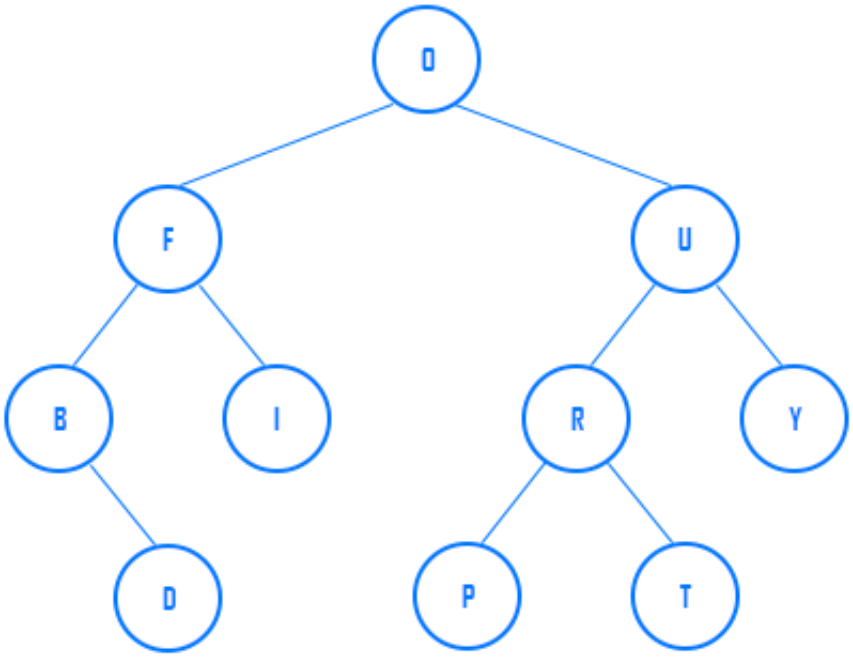
Na definição de Veloso (1986), árvores binárias são estruturas do tipo árvore, onde o grau de cada nó é menor ou igual a dois. Na computação a árvore é utilizada para o armazenamento de grande volume de dados na memória do computador de forma hierárquica,

permitindo tomada de decisões bidirecionais em cada nó, facilitando assim sua posterior recuperação.

Árvores binárias caracterizam-se pelo fato de que cada nó pode possuir no máximo dois ramos, ou seja, não existe nenhum nó com grau superior a dois. Veloso (1984), destaca que as árvores binárias, diferenciam-se entre a sub-árvore do lado esquerdo e as sub-árvores do lado direito.

A Figura 8 representa um exemplo simples de árvore binária.

Figura 8 - Ilustração de uma árvore binária



No exemplo da Figura 8 é apresentada uma árvore binária contendo algumas letras do alfabeto, mas poderiam ser números. Sua estrutura é formada por um nó especial chamado raiz, no exemplo representado pela letra “O”, sendo que este nó pode ou não conter filhos. Como observado no exemplo, o nó “O” contém os filhos “F” e “U” respectivamente. Uma árvore binária como o nome sugere, obrigatoriamente, deve conter 0, 1 ou no máximo 2 filhos.

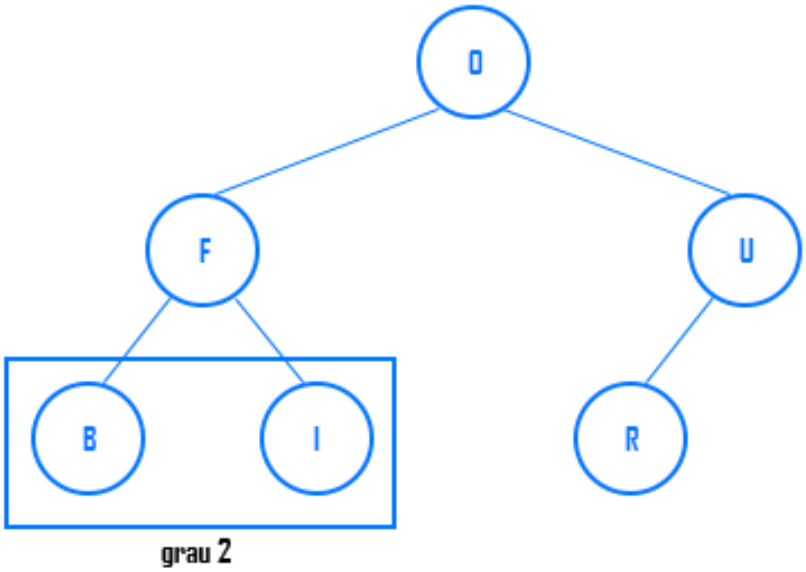
Na árvore cada nó possui uma chave, *key*, que conterá informações únicas, como: CPF, RG ou sequências de letras ou números únicos, permitindo, assim, sua posterior identificação. Além da chave, têm-se ponteiros *esquerdo* e *direito* para seus respectivos filhos esquerdo e direito, podendo ser *null* caso não contenha filhos. Caso a árvore seja uma árvore

binária de busca, todos os nós da sub-árvore esquerda, serão obrigatoriamente menores que seu nó ascendente e todos os nós da sub-árvore direita serão maiores que seu nó ascendente.

2.2.1 Grau da Árvore

De acordo com Lopes (1999, p. 21), o grau da árvore binária "é fornecido pelo nó que tem maior grau". Toda árvore binária poderá ser definida com no máximo grau 2, visto que o número máximo de filhos é restrito a 2. O grau da árvore será sempre a quantidade máxima de filhos atingido por um determinado nó dentro da estrutura. A Figura 9 representa o grau de uma árvore binária.

Figura 9 - Grau da árvore binária

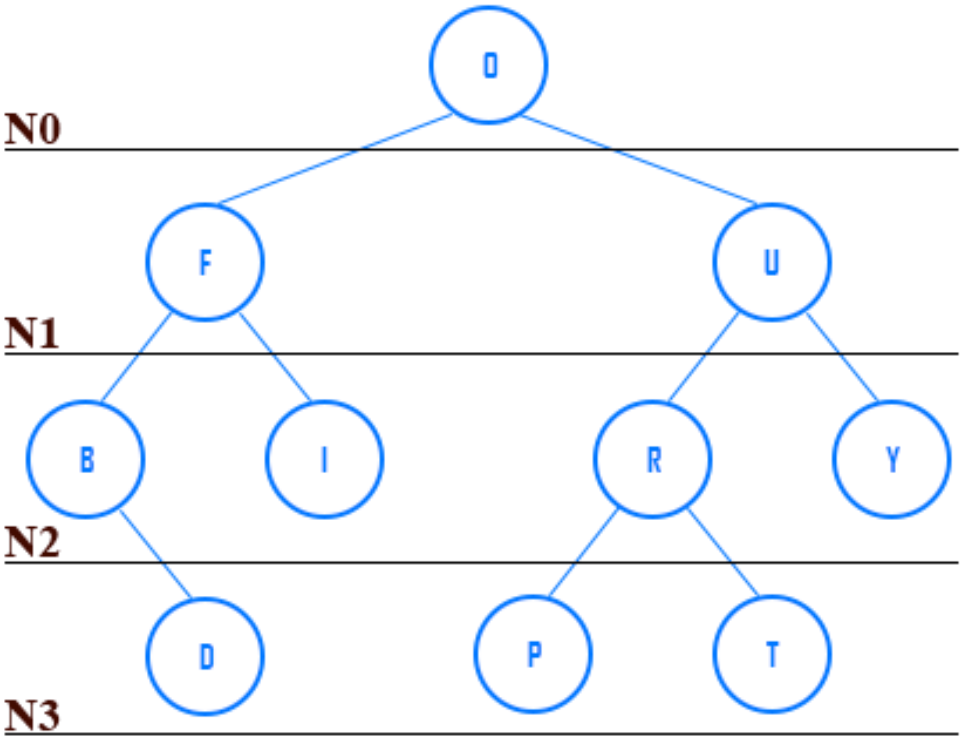


Na árvore binária ilustrada na Figura 9, é possível observar que o grau da árvore é 2, pois a quantidade de filhos que o nó "F" possui no exemplo é 2. A árvore terá grau 0 quando for vazia e grau 1 quando a quantidade de filhos do nó que contiver mais filhos for igual a 1.

2.2.2 Nível do Nó

De acordo com Robert Lafore (1999, p. 290), o nível de um nó particular se refere a quantas gerações o nó está da raiz. A Figura 10 representa um exemplo que mostra a classificação dos níveis do nó.

Figura 10 - Ilustração de uma árvore binária – nível nó



Como mencionado anteriormente, o nível de um nó é definido através da sua distância até o nó raiz. No exemplo, o nível do nó “I” é 2, do nó “F” é 1, do nó “D” é 3 e do nó raiz “O” é 0. O nível de um dado nó (x) será definido através do nível do seu nó pai mais 1.

Para se determinar a quantidade de nós de um determinado nível de uma árvore binária, faz uso da expressão 2^n, onde n representa o nível onde o nó se encontra. Ao aplicar a expressão, no nível 2, temos: $2^n=2^2=4$, ou seja, no nível 2 é possível ter no máximo 4 nós. É importante ressaltar que o nível da árvore inicia-se em 0, nível este onde está localizado a raiz da árvore, sendo incrementado em mais 1 a cada nível que é gerado.

2.2.3 Altura/Profundidade De Uma Árvore Binária

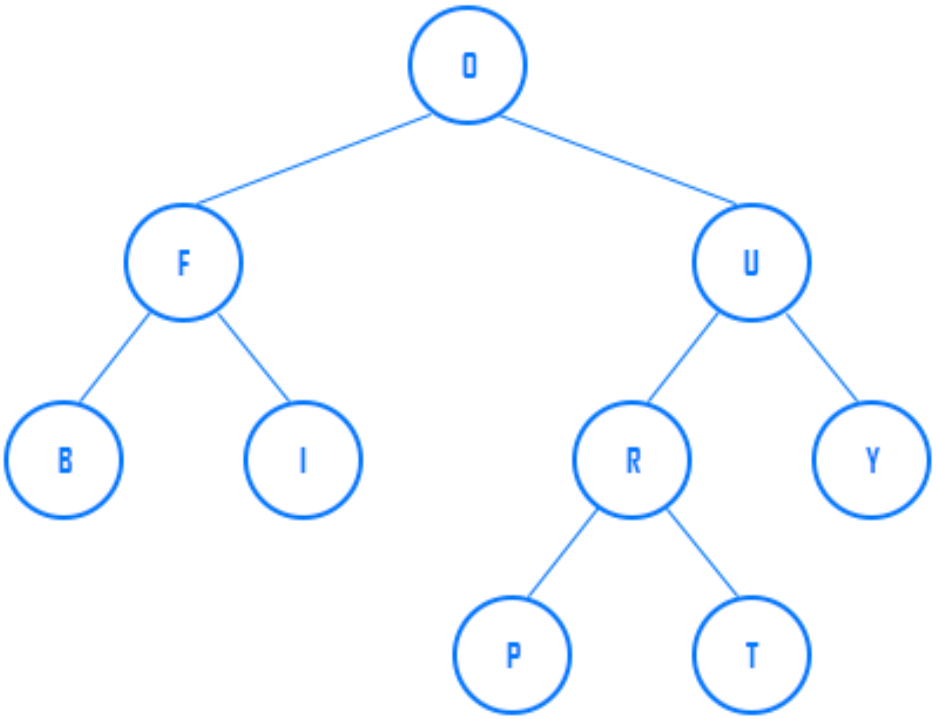
A altura da árvore é definida através da distância do nó raiz até o filho, descendente, mais distante, em outras palavras é a quantidade de interações necessárias até se chegar ao nó mais distante da raiz. A altura de uma árvore inicia-se em 0, logo, no exemplo da Figura 10 tem-se uma árvore com altura/profundidade 3. Uma árvore terá altura igual a 0, quando possui apenas um nó, ou seja, quanto tiver apenas a raiz em sua estrutura. A altura da árvore pode ser calculada aplicando a função $h=1+\log_2(n)$, onde n indica a quantidade de nós que a árvore possui. Logo aplicando a função sobre a árvore, Figura 10, $h=1+\log_2(n)=1+\log_2(10)=4$.

2.2.4 Árvore Estritamente Binária

Segundo Tenenbaum; Langsam; Langsam (1995, p. 306), “Se todo nó que não é folha numa árvore binária tiver sub-árvores esquerda e direita não-vazias, a árvore será considerada uma árvore estritamente binária”.

Em outras palavras, uma árvore é estritamente binária quando todos os seus nós, possuem 0 ou 2 filhos independentemente do nível que estejam. Na ilustração, Figura 10, a árvore não pode ser considerada uma árvore estritamente binária, pois o nó “B” possui um único filho. A Figura 11 representa um exemplo de árvore estritamente binária.

Figura 11 - Ilustração de uma árvore binária – estritamente binária



Na árvore, é possível observar que todos os nós possuem 0 e no máximo 2 filhos, podendo ser classificada então como árvore estritamente binária. Para obter o número de nós de uma árvore estritamente binária utiliza-se a expressão 2n-1, onde n é o número de nós folhas. Aplicando-se a expressão sobre a árvore, Figura 11, tem-se: 2n-1 = 2.5-1=9. Logo, a árvore tem em sua estrutura 10 nós.

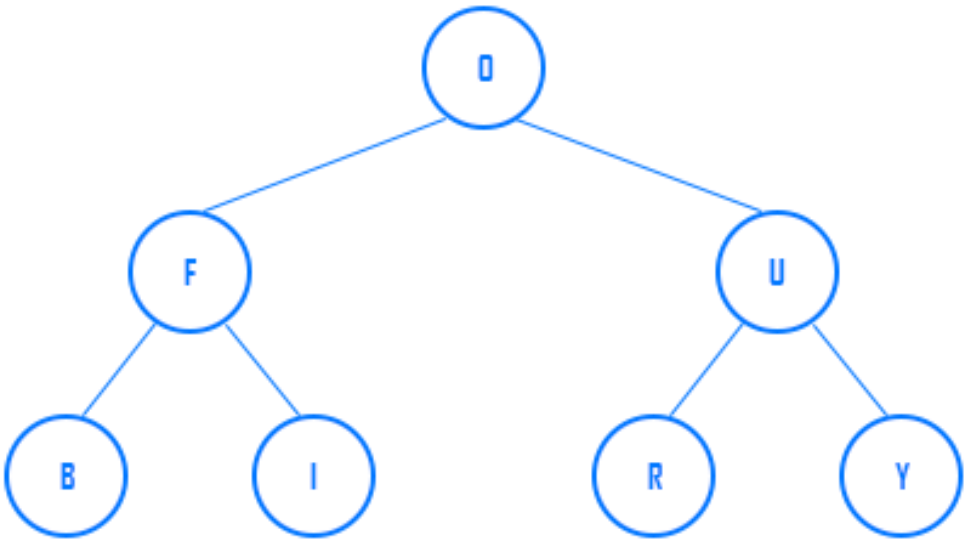
2.2.5 Árvore binária completa

Segundo Ascencio; Araújo (2010, p. 284), árvore completa é a “árvore em que todos os nós com menos de dois filhos ficam no último e no penúltimo nível”. No exemplo, Figura

11, a árvore pode ser considerada completa, pois todos os nós com menos de dois filhos contemplam os parâmetros obrigatórios para sua classificação.

Uma árvore pode ser considerada estritamente binária e completa quando for uma árvore cheia, ou seja, todos os nós do penúltimo nível, tiverem 2 filhos. A Figura 12 representa um exemplo de árvore completa e estritamente binária.

Figura 12 - Ilustração de uma árvore binária de busca – estritamente binária e completa

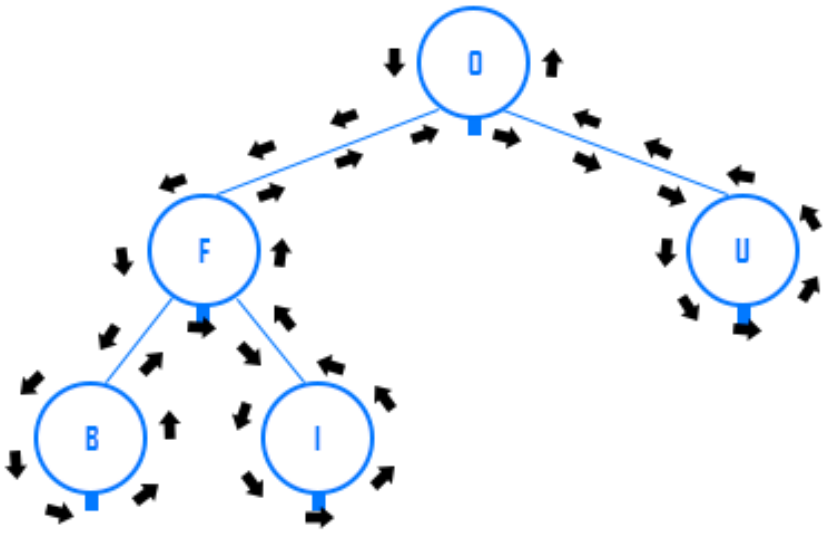


A Figura 12 ilustra um exemplo de árvore cheia, ou seja, todos os nós, exceto os nós folhas possuem 2 filhos. Somente neste caso, a árvore é classificada como árvore binária cheia.

2.2.6 Percurso em árvore binária: em-ordem

Segundo Robert Lafore (1999, p. 306), um percurso *em-ordem* de uma árvore de procura binária fará com que todos os nós sejam visitados na ordem ascendente, baseada em seus valores-chave. Este percurso será de grande valor quando a intenção for acessar ou obter todos os nós da árvore de forma ordenada. A Figura 13 representa um exemplo de percurso em em-ordem na árvore binária.

Figura 13 - Ilustração de uma árvore binária – percurso em em-ordem

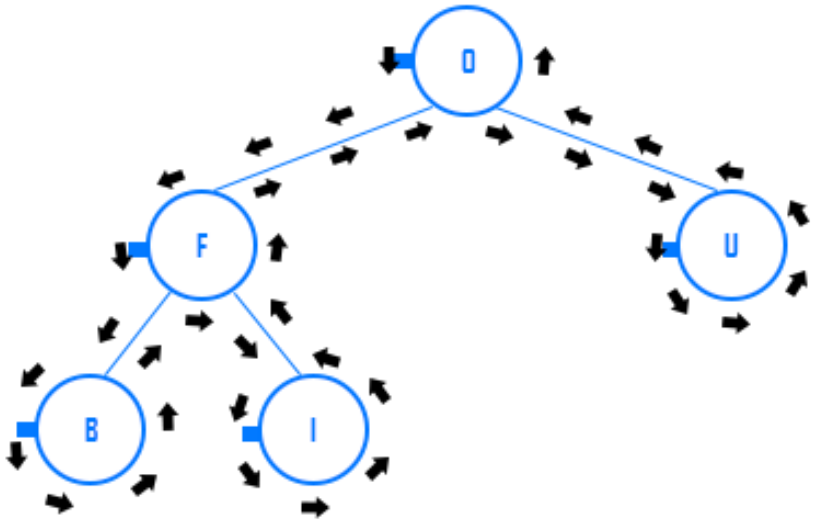


Para melhor entendimento do funcionamento de um percurso em-ordem, observe que na Figura 13, todos os nós, estão marcados com um traço na parte inferior. Para percorrer a árvore em em-ordem, acompanham-se as setas da esquerda para a direta seguindo a direção das arestas e respeitando as hierarquias dos nós. Todas as vezes que a seta passar abaixo do nó imprime-se o valor. Ao final do trajeto serão obtidos os valores “B”, “F”, “I”, “O” e “U” da árvore.

2.2.7 Percurso em árvore binária: pré-ordem

Segundo Ascencio; Araújo (2010, p. 285), “cada árvore é mostrada com a raiz, o ramo da esquerda e posteriormente o ramo da direita”. Diferentemente do percurso em-ordem, neste processo são apresentados na sequência que são acessados. O ponto inicial da busca será a raiz e serão visitados todos os nós das sub-árvore do lado esquerdo e posteriormente todos os nós das sub-árvores do lado direito. A Figura 14 representa um exemplo de percurso em pré-ordem na árvore binária.

Figura 14 - Ilustração de uma árvore binária – percurso em pré-ordem

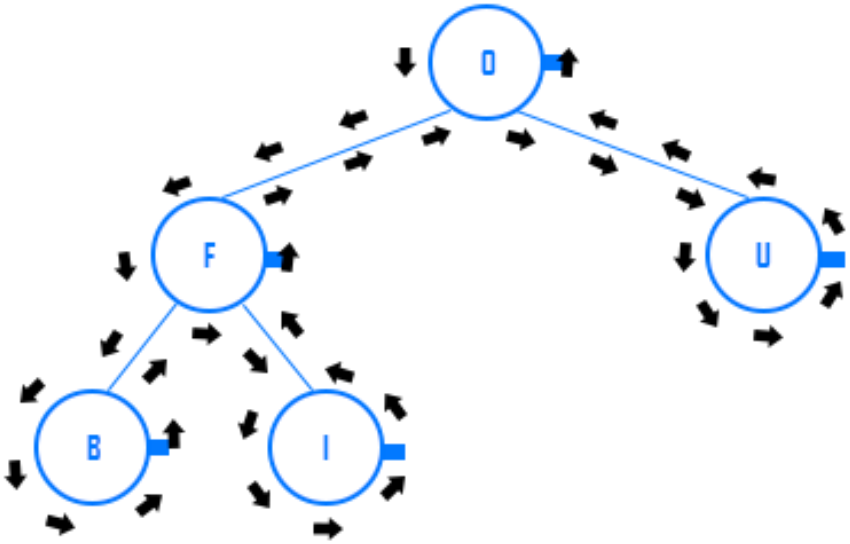


Para compreensão do funcionamento do percurso pré-ordem, observe que, todos os nós estão marcados com um traço do lado esquerdo. Para percorrer a árvore em pré-ordem, respeitando a hierarquia dos nós, percorrem-se todos os nós esquerdos de todas as sub-árvores e posteriormente todos os nós direitos de todas as sub-árvores. As setas ilustradas na Figura 14, mostram o sentido a se percorrer no percurso pré-ordem. Assim, todas as vezes que a seta passar do lado esquerdo do nó, imprime-se o valor. Ao final do trajeto serão obtidos os valores “O”, “F”, “B”, “I” e “U” da árvore.

2.2.8 Percurso em árvore binária: pós-ordem

Todos os percursos em árvore binária serão iniciados tendo como ponto de partida a raiz da árvore. Porém, o que diferem uma da outra é a forma com que as informações são extraídas e impressa ao final do percurso. A Figura 15 representa o percurso pós-ordem em uma árvore.

Figura 15 - Ilustração de uma árvore binária – percurso em pós-ordem



Na Figura 15, todos os nós estão marcados com um traço do lado direito, diferentemente da Figura 13 e Figura 14. As setas, ilustradas, indicam o sentido a seguir dentro da estrutura. Todas as vezes que a seta cruzar o traço, abstrai-se valor do nó. Como resultado, ao final do percurso serão apresentados os valores “B”, “I”, “F”, “U” e “O”.

2.3 Árvore Binária de Busca

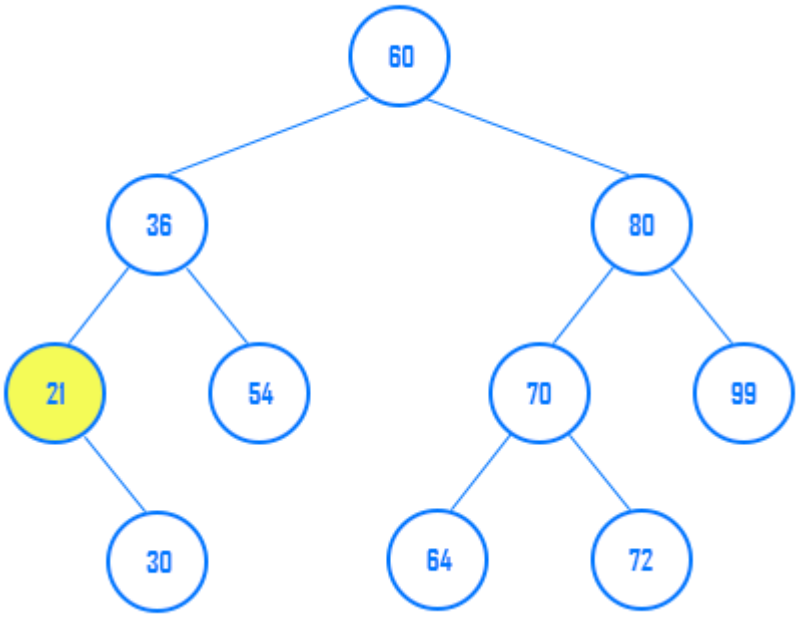
Uma árvore binária é classificada como árvore binária de busca quando, todos os nós do lado esquerdo são menores que seu nó ascendente, nó pai, e todos os nós do lado direito são maiores que seu nó ascendente, nó pai[1].

2.3.1 Valor mínimo e máximo de uma árvore binária de busca

Para localizar dentro de uma árvore binária de busca o nó de menor valor, partindo do princípio que todo nó do lado esquerdo é menor que seu nó pai, realiza-se o percurso visitando todos os nós das sub-árvores esquerdas subsequentes até chegar ao nó folha, ou seja, último nó esquerdo. Este nó conterá o menor valor de todos os nós existentes na árvore. A Figura 16 reapresenta uma ilustração do valor mínimo de uma árvore binária de busca.

[1] Disponível: http://wiki.icmc.usp.br/images/e/e9/ABB.pdf. Acesso em: 26 dez. 2018.

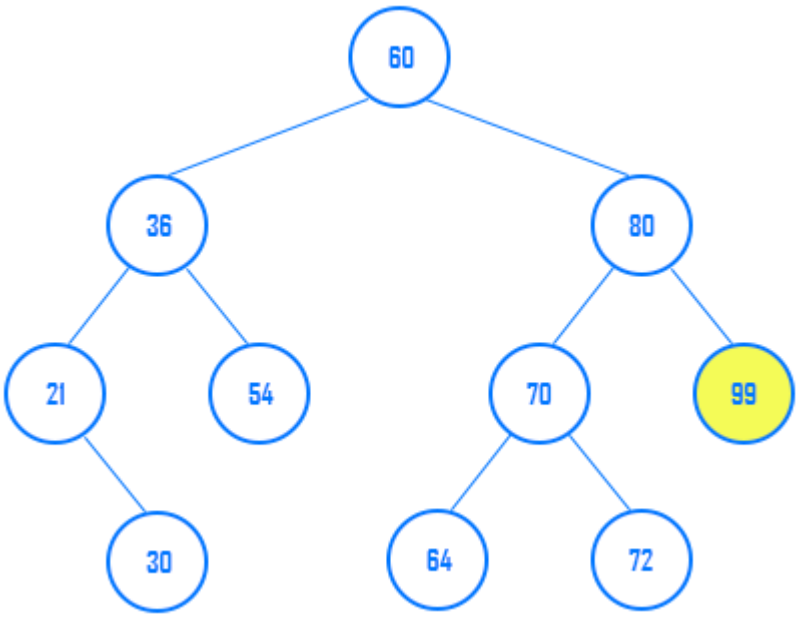
Figura 16 - Valor mínimo da árvore binária de busca



Na Figura 16 o nó de menor valor dentro da estrutura da árvore apresentada é o nó 21, isto devido ser o último nó localizado no lado esquerdo na hierarquia da árvore apresentada.

A mesma ideia é adotada para localizar o nó de maior valor, porém visitando todos os nós das sub-árvores direita subsequentes até chegar ao nó folha. Este nó conterá o maior valor de todos os nós existentes na árvore. A Figura 17 representa o valor máximo da árvore.

Figura 17 - Valor máximo da árvore binária de busca



Como se observa na Figura 17 o nó de maior valor dentro da estrutura da árvore apresentada é o nó 99, pois este é o último nó do lado direito na hierarquia da árvore.

2.3.2 Buscando valor na árvore binária de busca

A busca de um determinado valor em uma árvore binária de busca é dada a partir de um processo recursivo. O primeiro passo da busca consiste na verificação se a árvore está vazia ou não. Estando vazia, presume que o valor a ser localizado não existe na estrutura da árvore. Por outro lado, se a árvore não for uma estrutura vazia, as etapas a seguir deverão ser refeitas em todos os nós da estrutura até localizar o valor desejado:

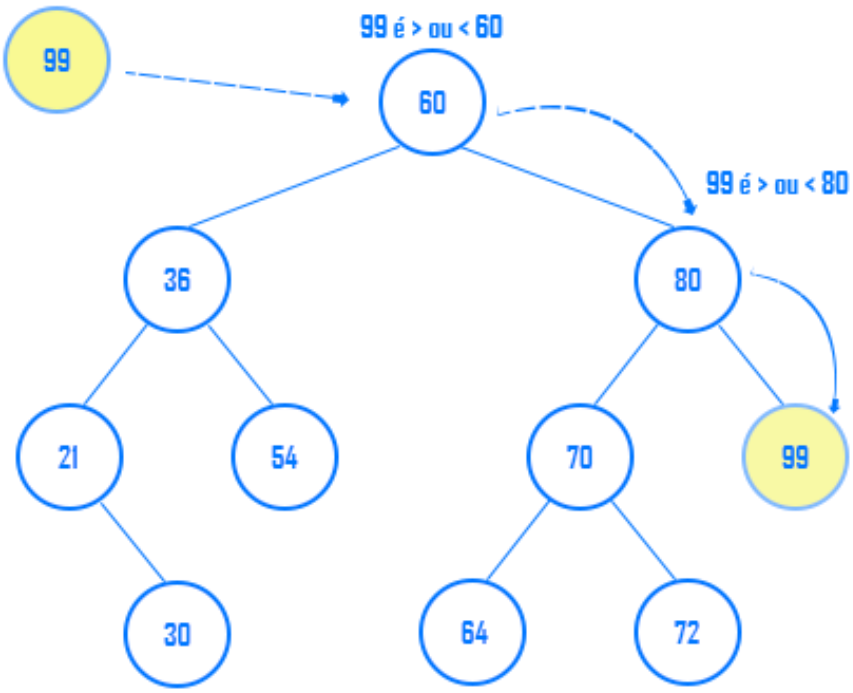
- comparar o valor a ser localizado com o valor do nó;
- se for igual, retorna a informação ao usuário que o valor existe;
- caso não seja igual, verifica se o valor é menor ou maior que o valor do nó atual;
- se for menor, verifica se existe um nó folha esquerdo;
- caso não exista nó folha esquerdo, retorna a informação ao usuário que o valor não existe;
- caso contrário, verifica se o valor a ser localizado é igual ao nó folha esquerdo;
- se for igual retorna a informação ao usuário que o valor existe;
- caso não seja igual, verifica se o valor é menor ou maior que o valor do nó atual.

As etapas apresentadas devem ser refeitas a todos os nós das sub-árvores esquerda até encontrar o valor ou chegar ao nó mais profundo da estrutura. O mesmo deve ser feito se o valor for maior que o nó atual, porém as interações deverão ser realizadas nas sub-árvores da direita até localizar o valor ou chegar ao nó mais profundo. Para árvores equilibradas, o sistema executará *logn* operações para localizar o valor desejado e no caso mais crítico *n* operações.

2.3.3 Inserindo um valor na árvore binária de busca

O primeiro passo para inserir um valor na árvore é verificar se o valor já existe em sua estrutura, ou seja, para toda nova inserção é realizada uma busca pelo valor a ser inserido e, caso o valor não exista, é verificado se o valor é menor ou maior que valor do último nó verificado e insere-o na sub-árvore da esquerda ou direta, respeitando a organização dos elementos na árvore. A figura 18 representa o processo para inserir um novo elemento na árvore binária de busca.

Figura 18 - Inserindo um valor na árvore binária de busca



Na Figura 18, o valor 99 será inserido na árvore binária de busca. Para isso, deve-se comparar o valor a ser inserido com os demais valores já existentes na estrutura para identificar o exato lugar no qual o valor será armazenado. No exemplo, o valor é comparado com o primeiro nó da árvore, a raiz, onde é feita a comparação se o valor é maior ou menor que o valor do nó existente. Se for menor, o valor deverá seguir o caminho esquerdo, caso contrário; seguirá o caminho direito até chegar ao último nó da sequência. Como é observado na Figura 18, o valor 99 foi alocado do lado direto do nó 80, visto que o valor é maior que 60, nó raiz, e maior que 80, nó subsequente. A regra deve ser respeitada para todas as inserções de valores na árvore binária de busca. O valor inserido sempre será o último valor, portanto, será classificado como nó folha.

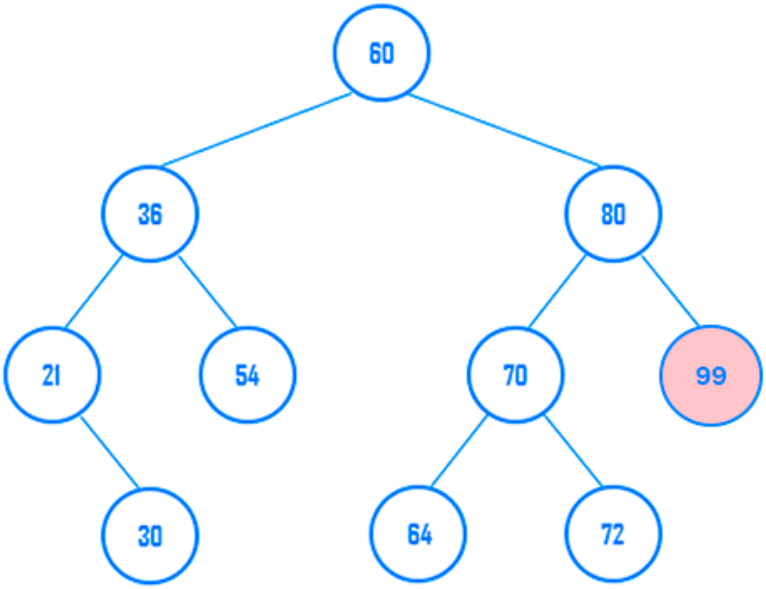
2.3.4 Exclusão De Valor Na Árvore Binária De Busca

O processo de exclusão é bem mais complexo que as operações de inserção e busca de um valor na estrutura da árvore. Esta complexidade é por ser necessário, dependendo do nó que foi removido, reorganizar os demais nós do ramo da sub-árvore. De acordo com Laureano; Marcos (2012, p. 133), as seguintes etapas devem ser consideradas para excluir um nó em uma árvore binária de busca.

2.3.4.1 Exclusão Nó Folha

Este caso é o mais simples de todas as operações de exclusão. A Figura 19 representa um exemplo de exclusão de nó folha.

Figura 19 - Exclusão nó folha árvore binária de busca

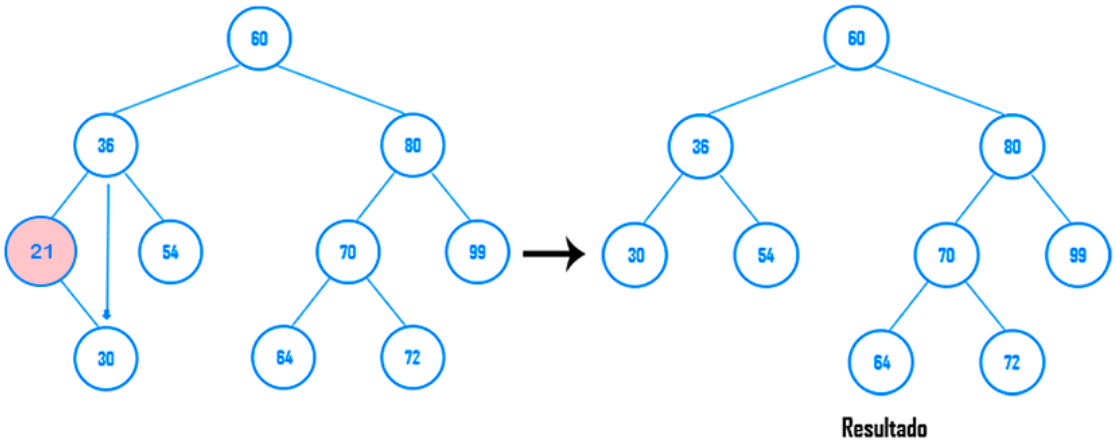


Para o caso de remoção de nós folhas o processo é bem simples, bastando apenas removê-lo da estrutura da árvore sem nenhuma complicação. No exemplo da Figura 19, o nó 99 será excluído e para isso basta apenas localizá-lo dentro da estrutura da árvore e removê-lo.

2.3.4.2 Exclusão De Nó Com Um Filho

O segundo caso é um pouco mais complexo, visto que há a necessidade de rearranjar o nó filho dentro da estrutura da árvore. A Figura 20 representa um exemplo de exclusão de nó com um filho.

Figura 20 - Exclusão de nó com um filho

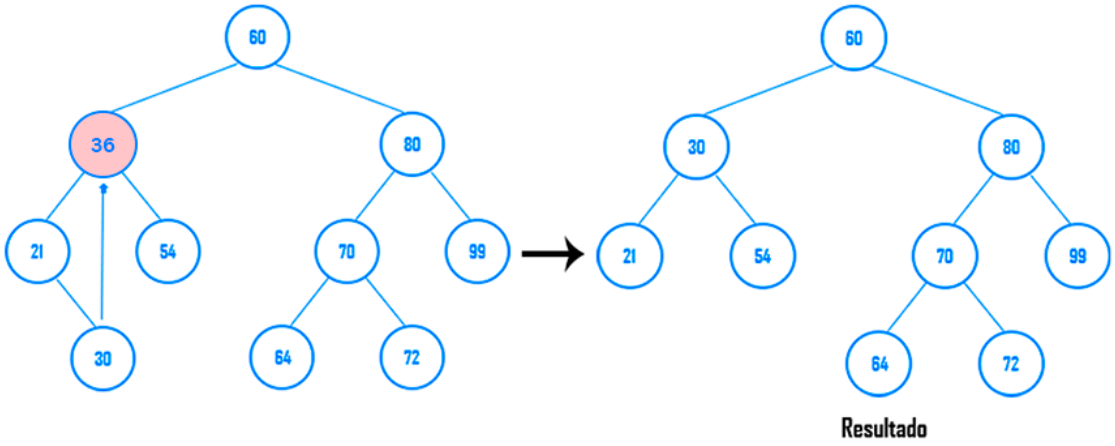


No exemplo, Figura 20, tem-se uma situação um pouco mais complexa. Nesta situação, ao excluir o nó 21, o nó filho deve assumir o lugar do seu nó pai, ou seja, neste exemplo o nó 30 assumirá o lugar do nó 21, tornando-se filho do nó 36. Ao lado, pode ser observada a árvore resultante após a remoção do nó 21. Como demonstrado no exemplo, é possível observar que o nó 30 assumiu a posição do seu pai, nó 21, removido da estrutura da árvore, tornando assim filho do seu nó avô 36.

2.3.4.3 Exclusão De Nó Com Dois Filhos

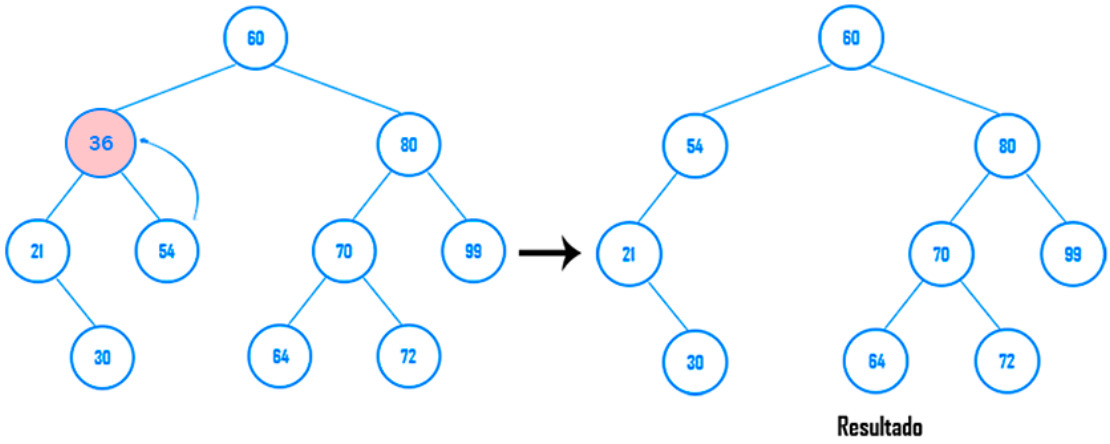
Neste caso, a exclusão poderá ser realizada de duas maneiras. A Figura 21 ilustra um exemplo de exclusão de nó com dois filhos. Segundo Laureano; Marcos (2012, p. 133), dois passos podem ser seguidos nesta situação, que é: substituir o valor do nó a ser removido pelo valor do nó mais à esquerda da sub-árvore da direta ou substituir o valor do nó pelo valor do nó mais à direita da sub-árvore da esquerda. A Figura 21 exemplifica a operação.

Figura 21 - Exclusão de nó com dois filhos - exemplo 01



No exemplo da Figura 21, o nó 36 será removido. Neste caso o nó 30 assumirá o lugar do nó 36, pois é o nó mais à direita da sub-árvore da esquerda do nó 36. O resultado é apresentado na figura ao lado. Outro exemplo pode ser visto na Figura 22.

Figura 22 - Exclusão de nó com dois filhos - exemplo 02



Neste exemplo, o nó que substituirá o nó 36 será o nó 54. Como o nó 54 não tem filhos, ele mesmo assumirá o lugar do seu pai, nó 36. Ao lado, pode ser observada a árvore resultante após a remoção do nó 36.

3 METODOLOGIA

Nesta seção são apresentados os materiais e a metodologia que foi utilizada para o desenvolvimento do projeto.

3.1 Materiais

Para compreensão e elaboração do referencial teórico foram realizadas pesquisas em livros impressos, livros digitais encontrados na *Internet*, artigos, materiais didáticos e vídeos que abordam o assunto relacionado à árvore binária, buscando, assim, o entendimento aprofundado sobre a estrutura. Além do referencial teórico sobre árvore binária, também foram realizadas pesquisas que contemplam as seguintes tecnologias:

- *HTML – HyperText Markup Language*: linguagem de marcação nativa nos navegadores que permite criar elementos visual que são interpretados e renderizados pelos navegadores;
- *Javascript*: linguagem de programação interpretada, que permite tornar o sistema mais dinâmico e interessante, permitindo manipular os elementos dispostos na área de desenho bem como executar as ações dos controles da ferramenta;
- *Bootstrap*: *framework* front-end utilizado para criar interfaces web no padrão W3C;
- *Framework SVG (svg.js)*: *framework* para manipular o componente *SVG* do *HTML*. Através do *framework* foi possível interagir e animar os nós da árvore na área de desenho e;
- *CSS – Cascading Style Sheets*: linguagem utilizada para estilizar os elementos do HTML. É através desta linguagem que foi possível formatar a parte visual da ferramenta.

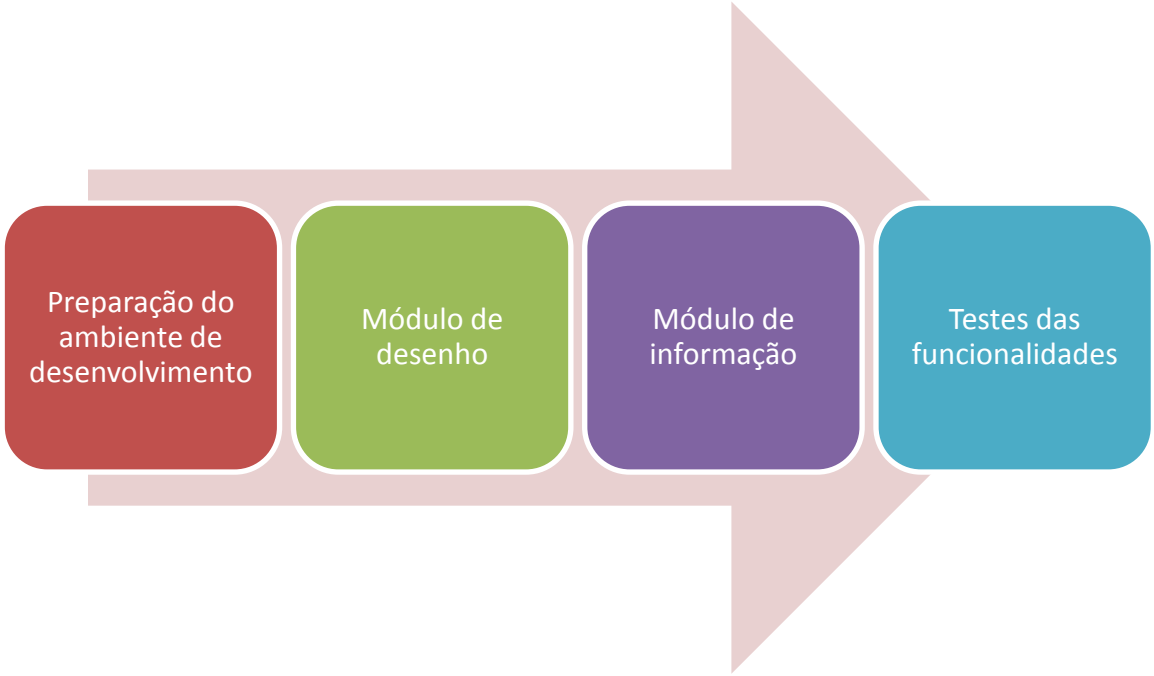
As tecnologias mencionadas foram utilizadas para desenvolver o ambiente que permitirá desenhar árvores binárias. A interface da ferramenta foi desenvolvida utilizando o *framework Bootstrap*. A utilização do *framework* permitiu criar uma interface prática, otimizada e responsiva, ou seja, a tecnologia permitirá ao aplicativo ter sua interface ajustada automaticamente a diversos formatos e tamanho de monitores.

Todo o ambiente, controles, módulo de desenho e demais recursos da ferramenta executam em *front-end*, ou seja, na máquina do usuário. Para implementar a ferramenta, foi utilizada a IDE JetBrains PyCharm 2018.1.4. A IDE foi escolhida por ser compatível com as tecnologias apresentadas, além de apresentar uma interface amigável e conter recursos adicionais que facilitarão a implementação.

3.2 Procedimentos

Os processos para desenvolvimento deste projeto foram sistematizados em: preparação do ambiente de desenvolvimento, módulo de desenho de árvore binária de busca e módulo de informação. Durante todo o processo de desenvolvimento foram realizadas reuniões com o professor responsável pelo projeto e professor da disciplina de Estruturas de Dados, M.e Fabiano Fagundes, para orientações e entendimento do projeto. A Figura 23 ilustra as etapas que foram seguidas para conclusão deste projeto.

Figura 23 - Etapas para desenvolvimento da ferramenta



Conforme pode ser observado na Figura 23, inicialmente foi preparado o ambiente de desenvolvimento. Nesta primeira etapa, as ferramentas necessárias para desenvolvimento da aplicação foram devidamente instaladas e configuradas.

A segunda etapa do projeto foi desenvolver o módulo que permitisse desenhar árvore binária de busca, ou seja, a *Interface* da aplicação. Na tela da aplicação são disponibilizados controles que permitem criar os nós e as arestas que compõem a estrutura de dados, árvore binária de busca. Para isso, foi necessário estudar e compreender as seguintes tecnologias: *HTML* (*Hypertext MarkupLanguage*) (Bootstrap); CSS (*CascadingStyleSheets*) e SVG (*Scalable Vector Graphics*). Através do componente SVG foi possível renderizar as formas geométricas necessárias para desenhar a árvore binária de busca.

A terceira etapa foi o desenvolvimento do módulo de informação. Nesta etapa foram implementadas as classes, objetos e funções necessárias para analisar e extrair as informações

da estrutura de dados, árvore binária de busca, criada em código *Javascript*. Além dos modelos citados anteriormente, foi implementada em *Javascript* a estrutura da árvore binária de busca que refletirá o desenho criado na área de desenho. Todas as alterações realizadas sobre o desenho criado na área de desenho são transferidas para a árvore binária de busca implementada em *Javascript*.

Durante todas as etapas que contemplam o módulo de desenho e módulo de informação, foram realizados testes de funcionalidade buscando corrigir eventuais erros existentes na codificação. Com isso, foi possível analisar se as informações retornadas pelo sistema, através do módulo de informação, estavam corretas.

Além dos testes durante a implementação dos recursos foram realizadas, junto ao professor de Estrutura de Dados, Fabiano Fagundes, testes de funcionalidade. Tais testes foram necessárias para identificar, junto ao professor, eventuais erros conceituais além de identificar se o aplicativo atenderia às necessidades para uso em salas de aula ou extra-classe, pelos alunos.

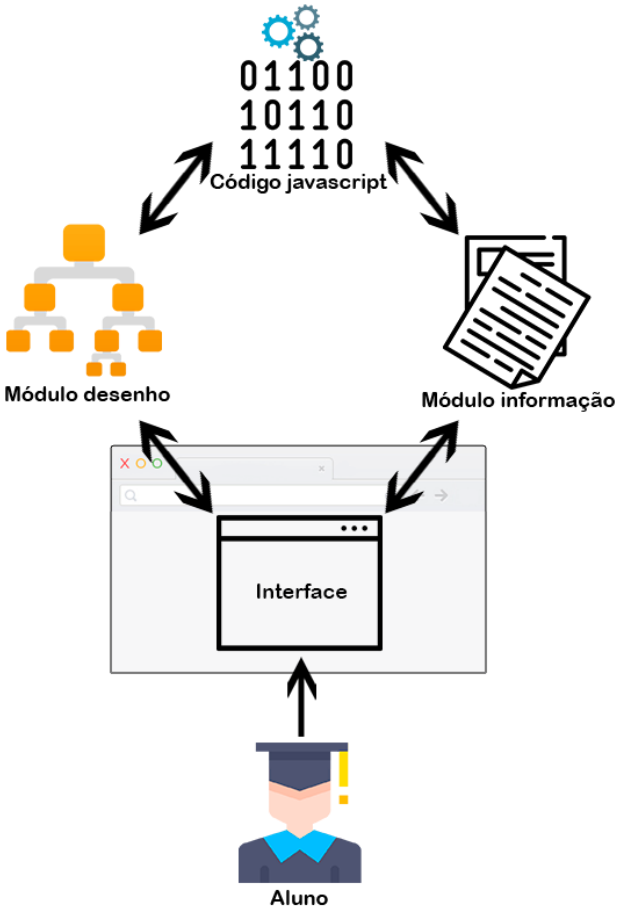
4 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados os resultados obtidos no desenvolvimento da aplicação web, bem como as soluções adotadas para implementação do que foi proposto neste trabalho. Os tópicos estão organizados na da seguinte forma: estrutura da aplicação, aplicação e codificação.

4.1 Estrutura da Aplicação

A Figura 24 apresenta a estrutura da aplicação para melhor entendimento do funcionamento dos módulos desenho e informação.

Figura 24 - Estrutura da aplicação

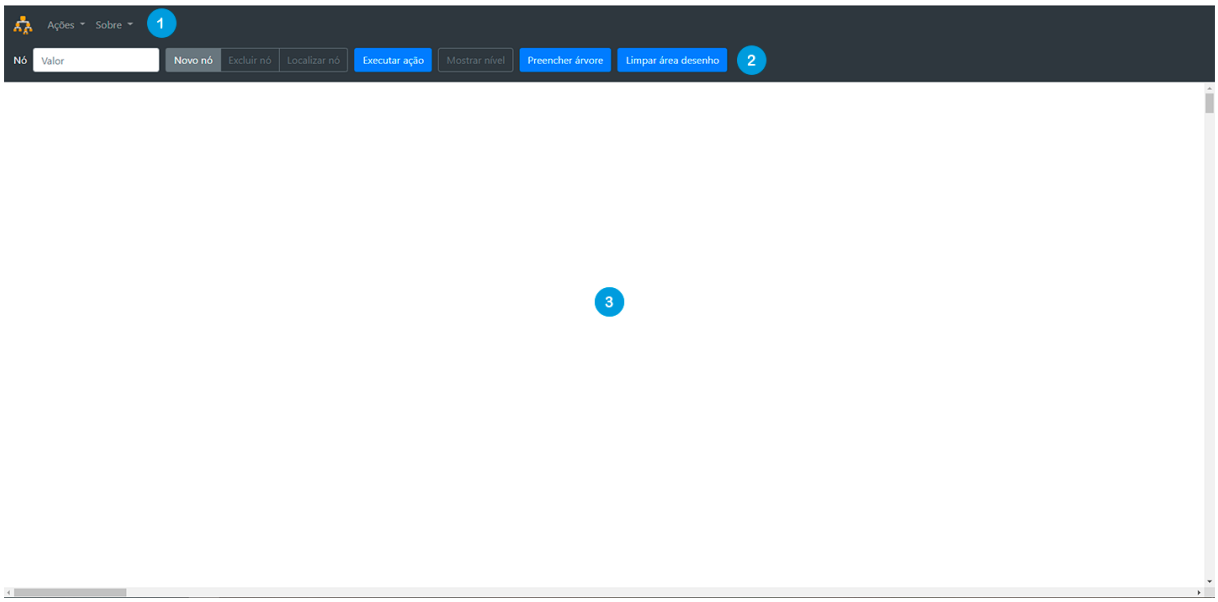


A Figura 24, o aluno tem acesso à interface da aplicação por meio do navegador Web. O módulo de desenho permite ao aluno criar a árvore gráfica com base na estrutura criada em código *Javascript*. O módulo informação realiza processamento sobre a estrutura da árvore em código *Javascript*, extrai as informações relacionadas à árvore e apresenta estas informações na interface para o aluno.

4.2 A Aplicação

A aplicação foi desenvolvida para ser utilizada através de navegadores Web, tornando assim a aplicação mais acessível aos alunos e professores, visto que não há necessidade de realizar uma pré-instalação para sua utilização. As tecnologias escolhidas têm por objetivo simplificar o uso da ferramenta, pois são nativas nos principais navegadores existentes. A interface da aplicação foi desenvolvida utilizando o *framework Bootstrap*, sendo totalmente responsiva, ou seja, a interface se auto ajusta a diversos tamanhos de tela e dispositivos. A Figura 25 apresenta a interface principal da aplicação.

Figura 25 - Interface da aplicação



A tela principal da aplicação está dividida em três partes:

1. *Menu principal*: disponibiliza recursos para realizar operações sobre a árvore, como: busca em pré-ordem, em-ordem e pós-ordem, destacar nós folhas e destacar o menor e o maior nó da árvore. Além desses recursos o aluno terá acesso ao menu que permitirá ter acesso a todas as informações gerais da árvore desenhada;

2. *Barra de menu*: disponibiliza os recursos de inserção, exclusão e localização de um nó na estrutura da árvore, além de recursos para visualizar os níveis da árvore graficamente, preencher a árvore com valores aleatórios e limpar a área de desenho;

3. *Área de desenho*: área onde é apresentada a árvore binária de busca criada pelo o aluno.

4.3 Área de Desenho

Para criar a árvore gráfica foi utilizado o *Framework* SVG, com o qual é possível criar elementos gráficos e animá-los na área de desenho. A área de desenho também é um elemento

SVG e, é neste, que são adicionados os demais componentes que fazem parte da estrutura da árvore. Além dos elementos da árvore gráfica, nós, arestas e rótulos, todos os outros que aparecem na área de desenho e que são utilizados nas animações para complementar as informações do desenho criado são elementos SVG.

Além da árvore gerada na área de desenho, existe uma segunda árvore em código *Javascript*. Duas árvores idênticas, uma em forma gráfica e outra em código, são geradas simultaneamente, sendo uma o espelho da outra. Todas as operações como, por exemplo, inserção, exclusão, percursos e demais operações são realizadas sobre a árvore em código e posteriormente é gerada a animação sobre a árvore gráfica simulando a ação solicitada pelo aluno.

Para criar um nó na árvore, o aluno deve informar o valor do nó, escolher a opção "Novo nó" e clicar no botão "Executar ação". Após o aluno informar o valor do nó e clicar em "Executar ação" o nó é criado na estrutura da árvore binária em código *Javascript* e logo em seguida é criado o nó gráfico na área de desenho. O processo de exclusão e busca do nó é semelhante ao processo de inserir, bastando apenas o aluno informar o valor do nó, selecionar a ação e clicar no botão "Executar ação". As ações estão disponibilizas na barra de ferramentas como pode ser observado na Figura 25.

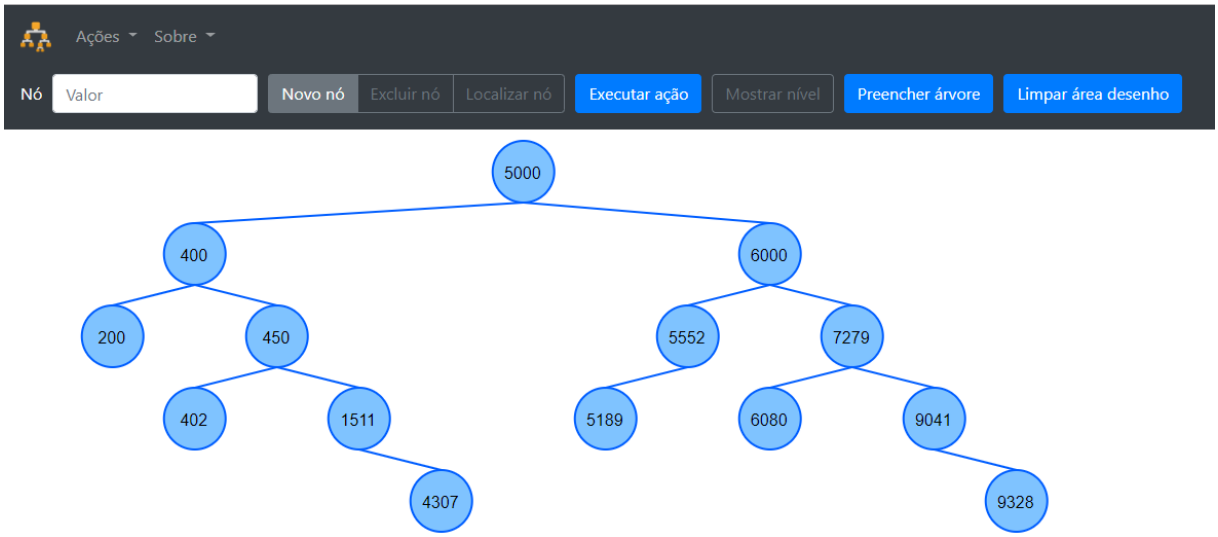
Após criar o nó gráfico é chamado o método responsável pelo seu posicionamento na área de desenho. O posicionamento do nó é realizado no momento da inserção do nó na estrutura da árvore, com base nas posições x e y do nó pai, caso o nó seja um nó filho. O método de posicionamento dos nós na área de desenho contém regras pré-estabelecidas para garantir a não sobreposição dos nós e o não cruzamento das arestas. Para isso, foram definidos parâmetros como: não pode ter nó com valor menor que o valor de outro nó, cuja posição do eixo 'x' deste nó seja maior que o valor da posição do eixo 'x' do outro nó, e que esteja no mesmo eixo y; não podem existir nós com eixo x e y iguais. Quando ocorre a negação de alguma regra é chamado um segundo método que reposiciona os nós na área de desenho até que todas as condições sejam válidas.

Após criar o nó, sendo este nó filho de outro nó, é criada a aresta de ligação. A aresta é um elemento line SVG e representa a hierarquia entre os nós. A animação da aresta é feita tomando como referência suas extremidades, nó pai e nó filho. Porém não é atribuída uma animação diretamente na aresta. A animação é atribuída ao nó e o movimento do nó atualiza as posições x e y da extremidade da aresta, eixo x e y, ligadas ao nó.

Assim, toda vez que é atribuída uma animação de movimento sobre o nó, a aresta é dimensionada, alterando tamanho, inclinação e posições de suas exterminadas, eixos x1, y1, x2 e y2, tendo como referências os nós de suas extremidades. O código detalhado responsável por posicionar os nós na área de desenho pode ser visualizado na seção codificação.

A Figura 26 apresenta a área de desenho com a estrutura de uma árvore binária de busca criada.

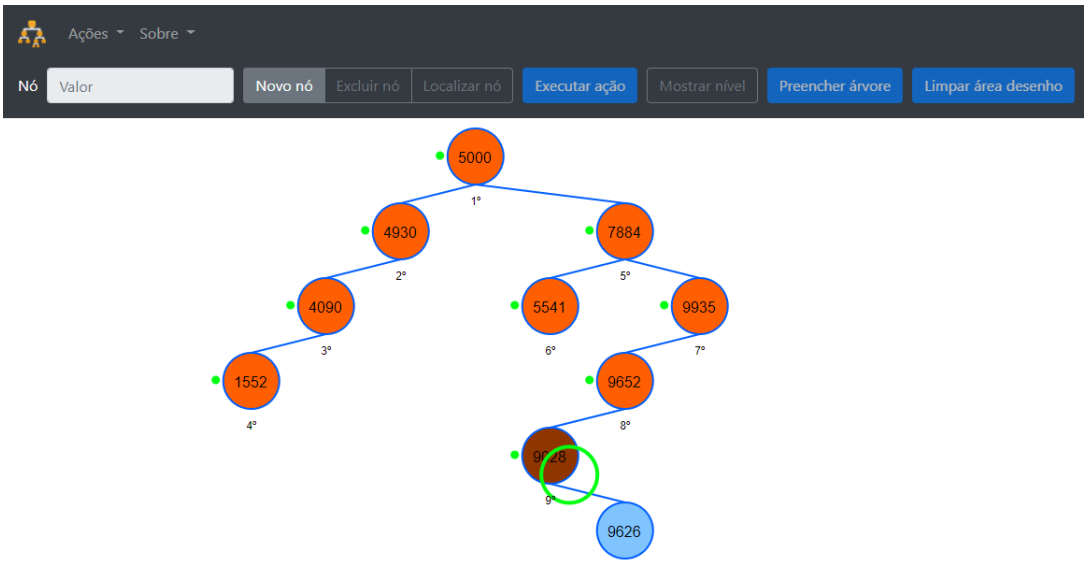
Figura 26 - Área de desenho



Toda interação do aluno com a aplicação é realizada através do teclado e mouse. Todos os controles estão distribuídos na barra de menu e barra de ferramentas. É possível observar, na Figura 26, uma árvore binária de busca desenhada na área de desenho. O nó da árvore é representado pela forma geométrica círculo com um rótulo contendo seu respectivo valor e as arestas são linhas comuns sendo utilizadas para fazerem as ligações entre o nó pai e nó filho. Outros elementos SVG contendo informações poderão ser adicionados à área de desenho para sinalizar ou complementar as informações já existentes, permitindo, assim, o melhor entendimento da estrutura da árvore binária ou ação processada sobre a árvore.

A Figura 27 apresenta elementos adicionais na área de desenho.

Figura 27 - Elementos adicionais



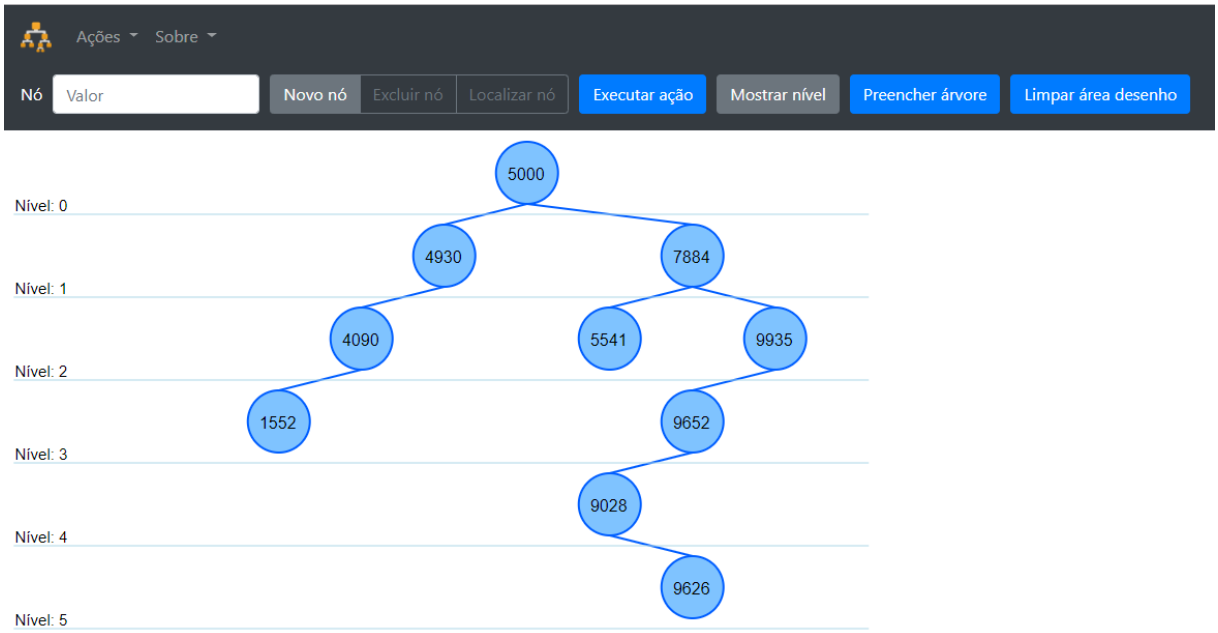
Como pode ser observado na Figura 27, além da árvore binária na área de desenho foram adicionados alguns novos elementos para melhorar a experiência do aluno na aplicação. Os elementos são adicionados em determinadas ações para complementar as informações já existentes ou sinalizar determinados estados. Para a animação de percurso, por exemplo, é criado um círculo verde contendo as mesmas dimensões dos nós da árvore. Ao círculo é atribuída uma animação de movimento fazendo com que este elemento visite todos os nós da árvore.

À medida que o círculo verde visita os nós, de acordo com o tipo de percurso selecionado, é criado um pequeno círculo e um pequeno valor numérico, sinalizando a ordem de acesso. No exemplo, o pequeno círculo verde está posicionado do lado esquerdo do nó e o valor número de acesso posicionado na parte inferior do mesmo nó.

4.4 Mostrar Nível

A aplicação disponibiliza o recurso mostrar nível, que permite criar linhas gráficas através do elemento `line` SVG, para cada nível da árvore. A Figura 28 apresenta linhas que indicam os níveis da árvore.

Figura 28 - Nível da árvore



Ao clicar na opção “Mostrar nível” o método `criarLinhaNivelArvore` da classe arvore_`animacao` é chamado. Para criar as linhas dos níveis da árvore, o método foi organizado de acordo com a seguinte estrutura:

- recupera a profundidade da árvore através do método profundidade. O valor é necessário para saber quantas linhas, níveis, serão criados;
- recupera a posição y do nó raiz. A posição y do nó raiz mais o raio do nó é utilizado para determinar a posição da primeira linha de nível a ser adicionada na área de desenho;
- recupera a posição x do nó que está mais à direita da árvore. Este valor será utilizado para determinar o comprimento da linha.

Com os valores indicados capturados é criado um laço de repetição tendo como limite a profundidade da árvore. Para cada interação é criada uma nova linha com respectivo rótulo indicando seu nível. Tanto ao rótulo quanto à linha é atribuído uma animação para que sua inserção na área de desenho ocorra de forma suave.

4.5 Preencher Árvore

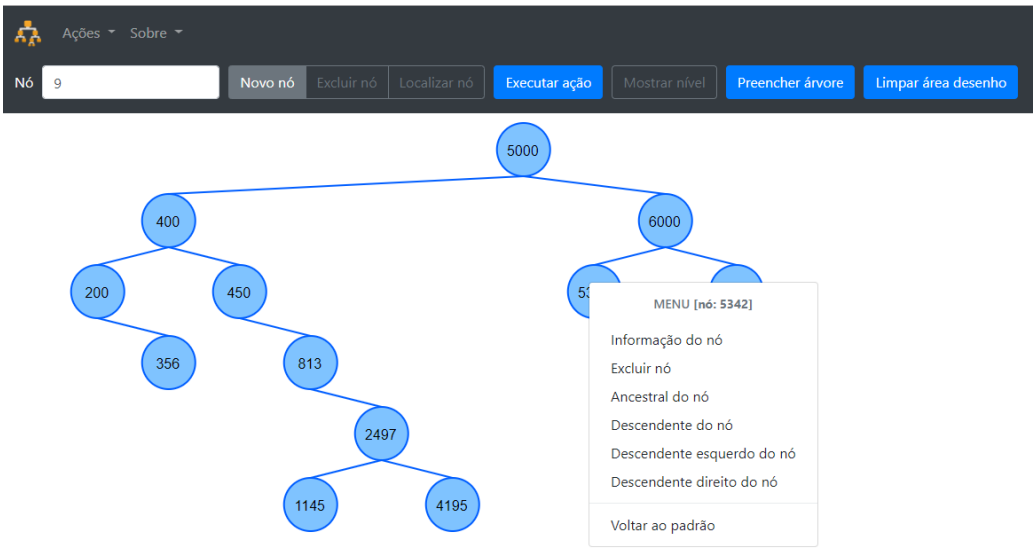
A aplicação disponibiliza o recurso preencher árvore, que está disponível na barra de ferramenta e permite gerar valores aleatórios que são inseridos na estrutura da árvore. A cada execução é armazenada em uma variável a quantidade de valores que deverão ser inseridos na árvore de forma aleatória. Este valor é utilizado no laço de repetição e dentro do laço é gerado

também, de forma aleatória, valores que variam de 1 a 10000. Os resultados são adicionados na estrutura da árvore.

4.6 Menu do Nó

A seguir será apresentado o menu do nó, que é o recurso pode ser acessado clicando com o botão direito do mouse sobre o nó na área de desenho. A Figura 29 apresenta o menu que será apresentado após clicar sobre o nó.

Figura 29 - Menu suspenso direito

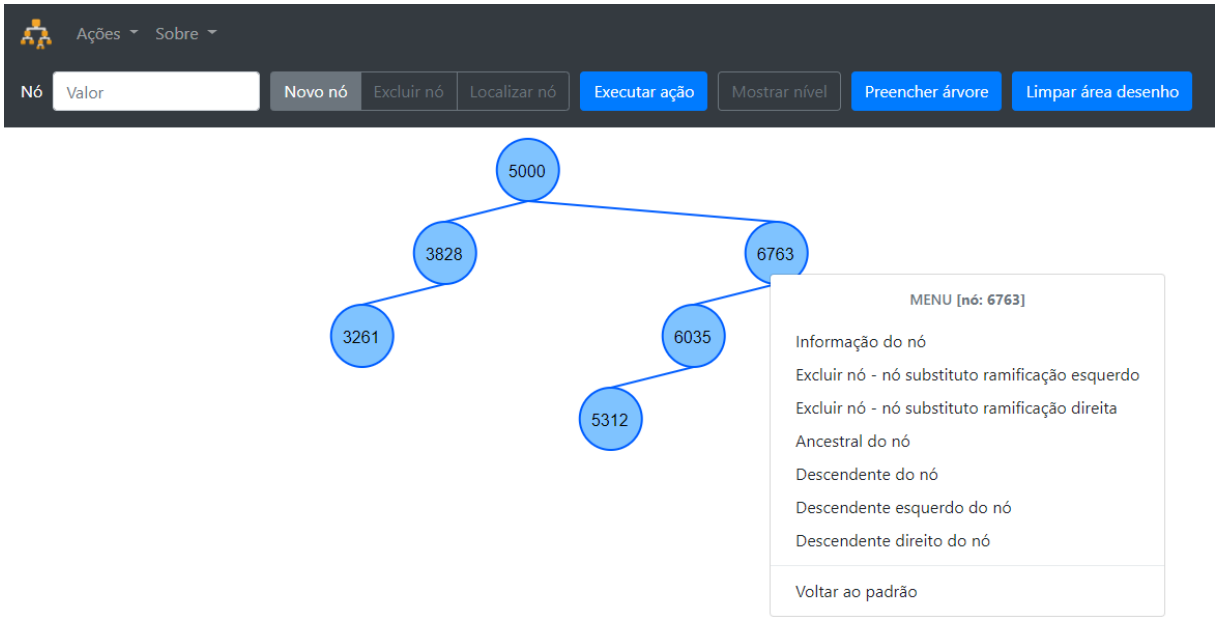


Como pode ser observado na Figura 29, o menu acionado com o clique do botão direito do mouse sobre o nó disponibiliza opções que permitem interagir com o nó selecionado. Este menu contém a opção informação do nó, recurso que permite obter dados relacionados ao nó selecionado. As informações obtidas são semelhantes às mostradas quando o aluno clica com o botão esquerdo do mouse sobre um dado nó.

As demais opções do menu, como: excluir nós, obter nós ancestrais, descendentes, descendentes esquerdos e direitos, além de voltar às cores padrões iniciais da árvore, permitem interagir com os nós da árvore binária. Por exemplo, se o aluno escolher a opção “Ancestral do nó”, será chamado o método que mapeia todos os ancestrais do nó, ou seja, todos os nós que estão no caminho entre o nó selecionado e o nó raiz. Para estes nós são atribuídas animações de coloração na área de desenho. O mapeamento é realizado através de um método recursivo que visita todos os nós, armazenando-os em uma lista para posterior processamento. Lógica semelhante é utilizada para as opções descendente do nó, descendente esquerdo do nó e descendente direito do nó.

Para a opção “Excluir nó” é verificado primeiro se o nó possui nenhum ou apenas um único filho. Caso ocorra de o nó possuir dois filhos a opção “Excluir nó” é removida e são adicionadas duas novas opções: Excluir nó – nó substituto ramificação esquerda e Excluir nó – nó substituto ramificação direita. A Figura 30 apresenta o menu com as duas opções, pois o nó selecionado possui dois filhos.

Figura 30 - Menu excluir nó com dois filhos



Como é possível observar, Figura 30, o menu apresentado contém duas novas opções excluir. As opções são apresentadas devido ao fato do nó selecionado possuir dois filhos. Assim o aluno poderá escolher dentre as duas opções de remoção de nós com dois filhos, adotando o substituto na ramificação esquerda do nó a ser excluído ou em sua ramificação direita.

As três opções de exclusão chamam o mesmo método, que recebe o valor a ser excluído por parâmetro, localiza o nó a ser excluído, atualiza o referenciamento dos parentes, reposiciona a arestas de acordo com a nova hierarquia, caso necessário, e ao final retorna o nó excluído. Com a referência do nó que será excluído retornada, é chamado o método responsável pela animação na classe correspondente. O método exclui todos os elementos, círculo, aresta(s) e rótulo do nó excluído, atribuindo animações pré-estabelecidas.

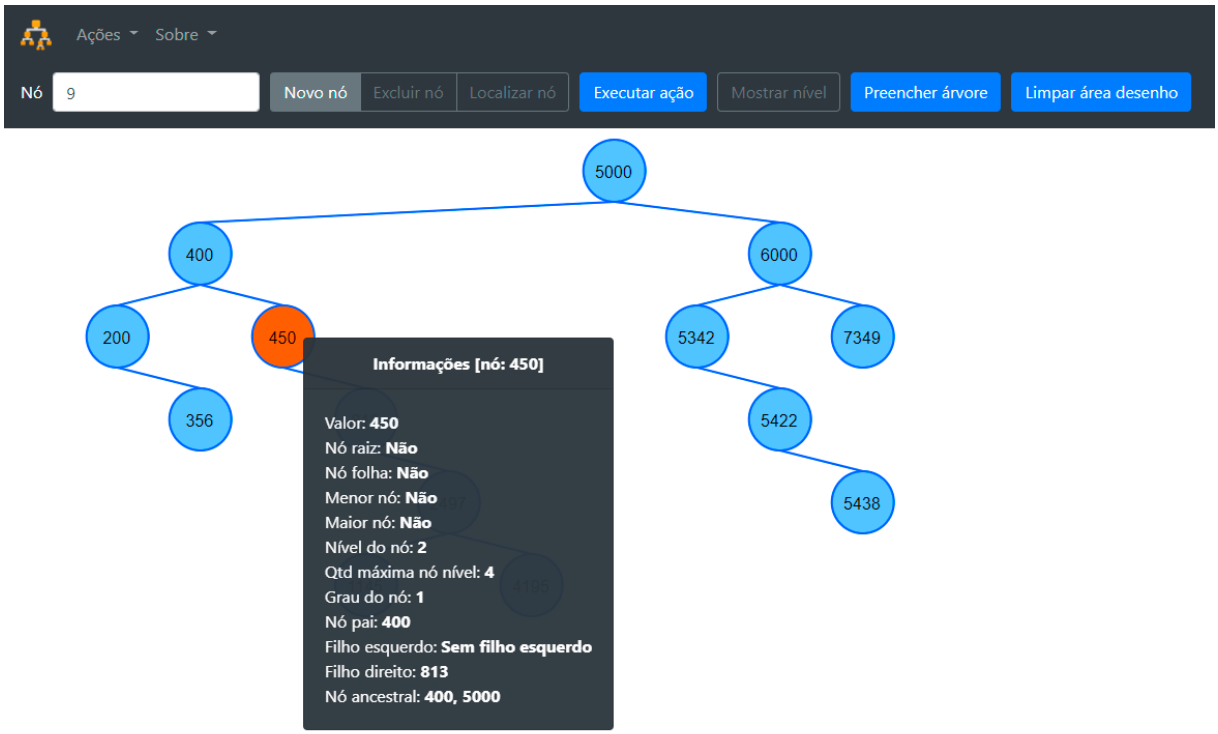
4.7 Módulo Informação

Nesta seção será apresentado o módulo informação que acessa todos os métodos da classe árvore binária em código, extraindo todas as informações relacionadas a esta estrutura. As informações são organizadas em duas etapas, sendo elas: informações relacionadas ao nó

selecionado e informações relacionadas à estrutura da árvore. Assim é possível obter não somente informações relacionadas à árvore no contexto geral, mas também obter informações relacionadas a um dado nó dentro da estrutura desenhada.

A Figura 31 apresenta uma caixa de texto que surge após o aluno clicar com o botão esquerdo do mouse sobre o nó.

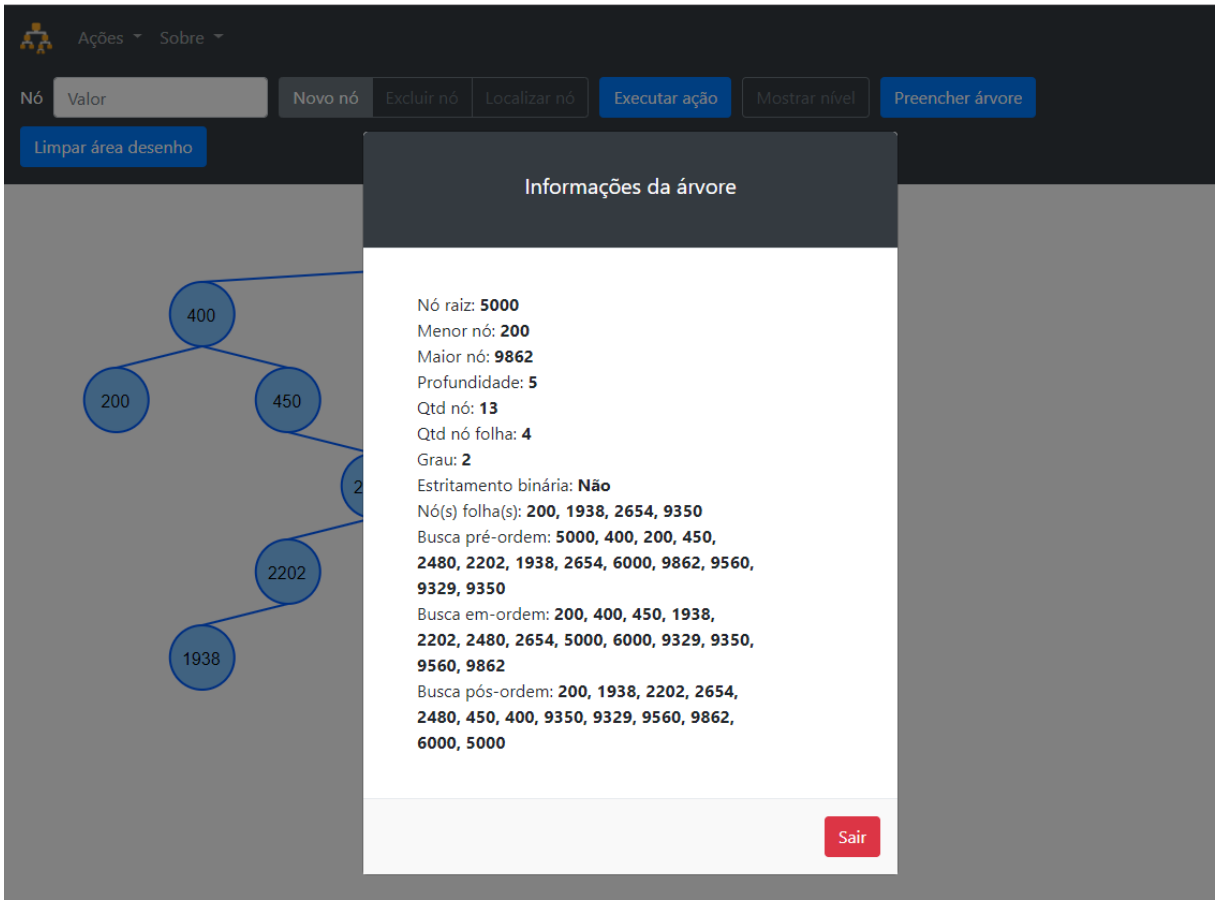
Figura 31 – Caixa informações - clique botão esquerdo do mouse



Ao pressionar o botão esquerdo no mouse sobre um determinado nó, será apresentada ao aluno uma caixa de texto contendo informações relacionadas ao nó selecionado. Como exemplo é possível observar, na Figura 31, a caixa de texto contendo todas as informações do nó 450. Informações como valor, nó raiz, nó folha, menor nó, maior nó, dentre outras relacionadas ao nó são extraídas da árvore criada em código *Javascript*.

As informações relacionadas à árvore no contexto geral podem ser acessadas através do menu ações, informações da árvore. A Figura 32 apresenta o modal contendo informações relacionadas à árvore binária desenhada na área de desenho.

Figura 32 – Informações da árvore

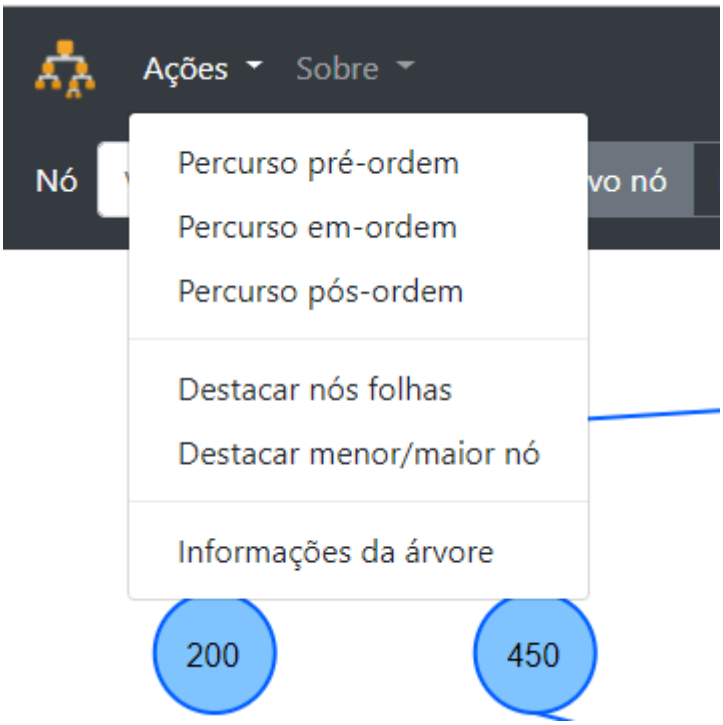


Através do recurso apresentado na Figura 32, é possível ter acesso às informações gerais da árvore desenhada. A caixa de texto apresenta informações como nó raiz, menor nó, maior nó, profundidade, quantidade de nó na árvore, quantidade de nó folha, grau da árvore dentre outras que são extraídas dos métodos da classe da árvore em código. As informações extraídas dos métodos da classe árvore são devidamente tratadas e organizadas em uma estrutura HTML e formatadas com *Bootstrap* para melhor apresentação. A estrutura HTML gerada é adicionada a um modal sendo apresentada ao aluno.

4.8 Ações da Barra de Menu

A barra de Menu disponibiliza alguns recursos que permitem ao aluno interagir sobre a árvore binária desenhada na área de desenho. A Figura 33 apresenta o menu ações e todas as operações possíveis disponibilizadas no menu para interagir com o desenho criado.

Figura 33 - Menu ações



Através do menu ações, o aluno terá acesso a recursos que permitem realizar os percursos sobre a árvore, destacar todos os nós folhas, destacar o menor e o maior nó da árvore e por fim ter acesso às informações gerais da árvore, clicando na opção “Informações da árvore”. A seção a seguir apresenta os recursos: percursos, destacar nós folhas e destacar menor/maior nó.

4.8.1 Percursos da Árvore

Para realizar a animação dos percursos foi desenvolvido um método na classe animação que recebe por parâmetro o tipo do percurso. O método corrente chama o método responsável pelo mapeamento do percurso na classe árvore binária, retornando todos os nós na sequência correspondente ao percurso solicitado. O resultado do retorno da função é armazenado em uma lista para ser utilizada posteriormente na animação. Esta lista é necessária pois é através dela que a animação sabe o momento exato que deverá marcar o nó da árvore durante o processo do percurso.

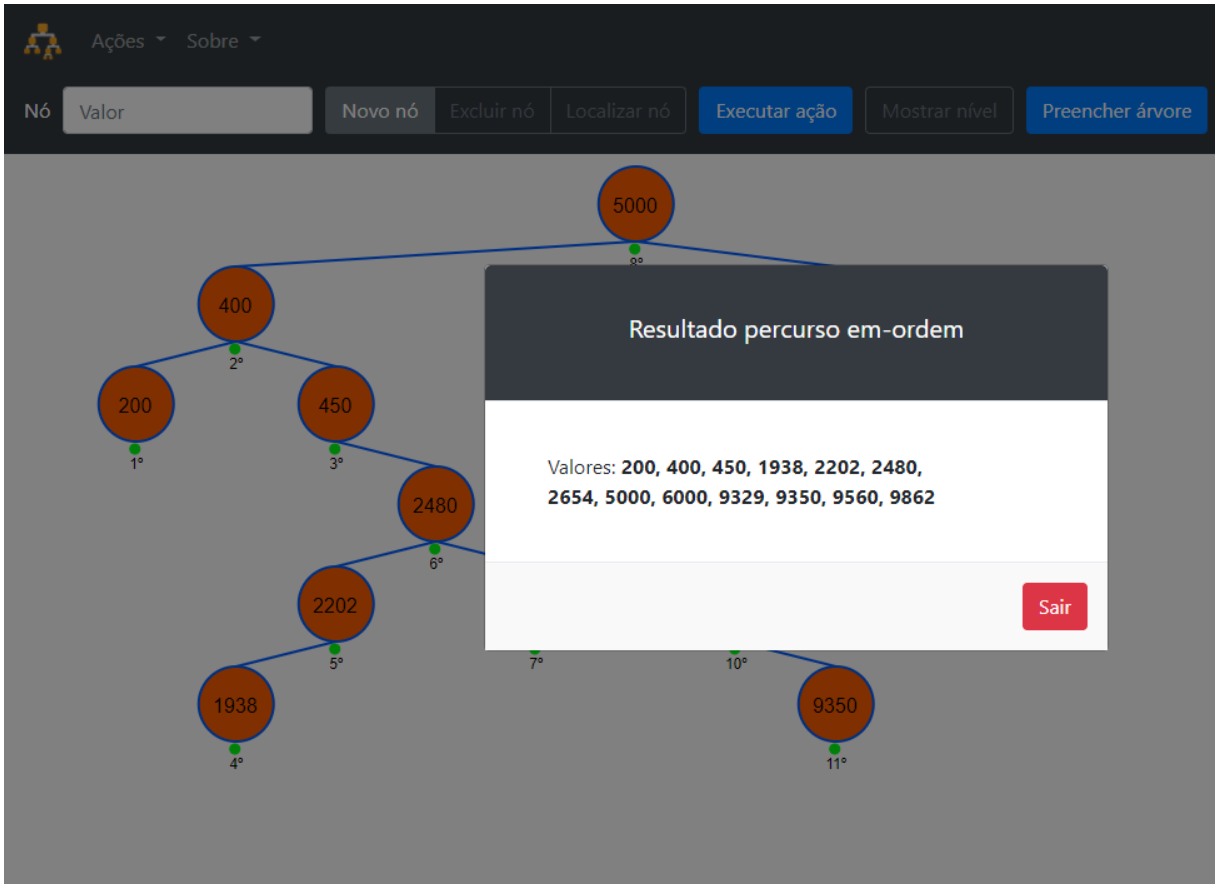
Em seguida, são percorridos todos os nós da árvore novamente, armazenado o caminho de ida e volta que o círculo verde utilizado na animação deverá percorrer. Para isso, foi recriado o método percurso, onde se percorre toda a árvore e armazena todos os nós da estrutura em uma lista. A lista conterá todos os nós, podendo ocorrer de o nó aparecer mais de uma vez dentro da lista em índices diferentes, visto que, o caminho foi mapeado contendo ida e volta. Esse processo é necessário pois pode ocorrer de o nó conter ramificações diversas em

sua estrutura e a animação, deve ir até nó mais profundo à esquerda, retornar para o nó superior e posteriormente ir até o nó mais profundo à direita.

Com os resultados obtidos nos processos anteriores, o próximo passo é realizar a animação de percurso sobre a árvore na área de desenho. Para isso, é criado um elemento SVG círculo de cor verde, contendo as mesmas dimensões dos nós da árvore. O momento que a aplicação destaca o nó com o valor da sequência e com um pequeno círculo verde, será definido através da lista, contendo os nós na sequência, conforme o percurso escolhido. O realce do nó procederá no momento que o nó for visitado e seu valor for igual ao primeiro nó da lista de resultado do percurso escolhido. Neste momento o nó será realçado e o primeiro elemento da lista percurso será excluído, não sendo mais necessário para o restante da animação.

Ao final da animação é apresentado um modal contendo todos os valores no percurso selecionado. A Figura 34 apresenta o modal contendo o resultado final do percurso solicitado pelo o aluno.

Figura 34 - Percursos



O nó é realçado com uma cor laranja, atribuído um círculo verde e o respectivo valor da sequência que o nó foi acessado no percurso escolhido. O código criado para realizar a animação pode ser visualizado na seção codificação.

Para o recurso que mostra os nós folhas é chamado o método destacar nó folha na classe animação que percorre todos os nós e verifica se os nós possuem filhos. Caso o nó não tenha filhos então ele é adicionado em uma lista e ao final da interação é aplicada uma animação nos nós retornados na lista.

Para o recurso destacar menor e maior nó foi seguida a mesma lógica escolhida no recurso destacar nós folhas. Quando selecionado o recurso é chamado o método correspondente na classe animação, onde é realizada uma busca recursiva para localizar o menor e, em seguida, o maior nó da árvore. Para o resultado da busca é aplicada uma animação de coloração, destacando os nós na árvore gráfica.

4.9 Codificação

A seção codificação apresenta parte da codificação da aplicação. Serão apresentados nesta seção a estrutura da classe `nó`; alguns métodos da classe `ArvoreBinariaBusca`, devido grande parte dos métodos possuírem estruturas semelhantes; e apresentar também os principais métodos da classe `AnimacaoArvore`.

4.9.1 Estrutura do Nó

O código ilustrado, Figura 35, refere-se a estrutura do nó em código *Javascript*.

Figura 35 - Estrutura do nó em *Javascript*

```
1  function No(id){
2      this.id = id;
3      this.valor = "";
4      this.filho_esquerdo = null;
5      this.filho_direito = null;
6      this.no_pai = null;
7      this.aresta_esquerda = null;
8      this.aresta_direita = null;
9      this.aresta_pai = null;
10 }
```

A estrutura é composta por um `id` que armazenará a sua identificação, linha 2; valor que armazenará o valor propriamente dito do nó, linha 3; `filho_esquerdo` e `filho_direito` utilizados para guardar as referências aos respectivos filhos, esquerdo e direito, linha 4 e 5; `no_pai`, linha 6, para referenciar o nó pai se existir, recebendo null caso seja o nó raiz; `aresta_esquerda`, `aresta_direita` e `aresta pai`, para armazenar a

referência aos elementos gráficos gerados na área de desenho, linha 7 a 9. As arestas são necessárias para manipular os elementos gráficos relacionados às linhas de cada nó, facilitando sua manipulação quando necessário. Além dos atributos listados, Figura 35, são criados dinamicamente mais dois atributos: `noGrafico` para armazenar a referência ao elemento `circle` que representa o nó propriamente dito e o `noRotulo` para armazenar a referência ao elemento `text` que representa o rótulo do nó.

4.9.2 Classe Árvore Binária

Nesta seção serão apresentados alguns métodos que fazem parte da estrutura da classe "`arvore_binaria_busca.js`". A classe contém todas os métodos necessários que permitem a interação sobre a estrutura, porém, serão apresentados apenas alguns, visto que os métodos seguem uma lógica e estrutura de desenvolvimento semelhantes.

A Figura 36 apresenta a estrutura dos métodos criados para adicionar um nó na árvore.

Figura 36 - Classe árvore binária inserir nó

```
/*estância o nó da árvore ****/
this.estanciaNo = function (valor) {
    var no = null;

    if (!this.raiz) {
        no = new No('raiz');
    } else {
        no = new No('no_' + valor);
    }

    no.valor = valor;
    no.filho_esquerdo = null;
    no.filho_direito = null;

    return no;
};

/*inicia a inserção do novo nó na estrutura da árvore binária ****/
this.adicionar = function (valor) {
    var noCorrente = this.estanciaNo(valor);

    if (!this.raiz) {
        return this.raiz = noCorrente;
    } else {
        return this.inserirNo(noCorrente);
    }

    return this;
};

/*função para inserir o nó  na posição correta na estrutura da árvore ****/
this.inserirNo = function (noCorrente) {
    var valor = noCorrente.valor;

    /*criar uma váriavel contendo a função recursiva para inserir o nó*/
    var funcaoInserirNo = function (no) {
        //verifica se o valor é igual ao valor do nó atual da interação.
        //se for igual não pode inserir, pois já existe na estrutura da árvore.
        if (valor === no.valor) {
            return;
        } else if (valor > no.valor) {
            //se for maior e não tiver filho direito, então insere como filho direito
            //do nó atual.
            if (!no.filho_direito) {
                noCorrente.no_pai = no;
                no.filho_direito = noCorrente;

                return no.filho_direito;
            } else
            //se tiver filho direito, então chama a função novamente passando como
            //parâmetro o filho direito do nó atual.
                return funcaoInserirNo(no.filho_direito);
        } else if (valor < no.valor) {
            //se for menor e não tiver filho esquerdo, então insere como filho esquerdo
            //do nó atual.
            if (!no.filho_esquerdo) {
                noCorrente.no_pai = no;
                no.filho_esquerdo = noCorrente;

                return no.filho_esquerdo;
            } else
            //se tiver filho esquerdo, então chama a função novamente passando como
            //parâmetro o filho esquerdo do nó atual.
                return funcaoInserirNo(no.filho_esquerdo);
        }
    };

    //chama função recursiva
    return funcaoInserirNo(this.raiz);
};
```

Para realizar inserções dos nós na estrutura da árvore é utilizada a estrutura apresentada na Figura 36. Para todo nó que o aluno insira na árvore é chamada a função `adicionar` que instancia o nó e passa esta estância para o método `inserirNo`. O método `inserirNo` percorre a árvore binária e, caso o nó não exista na estrutura, insere-o na posição correta. Todo o processo é realizado de forma recursiva conforme pode ser observado na Figura 36.

A Figura 37 apresenta a função que verifica se um determinado nó existe na estrutura da árvore e em seguida a função que recupera o nó cujo valor seja igual ao passado, por parâmetro.

Figura 37 - Código contêm e recupera nó

```
75      /*verifica se contem o valor na árvore ****/
76      this.contemNo = function (valor) {
77          var no = this.raiz;
78
79          var funcaoLocalizarNo = function (no) {
80              if (!no) return false;
81
82              if (valor === no.valor) {
83                  return true;
84              } else if (valor > no.valor) {
85                  return funcaoLocalizarNo(no.filho_direito);
86              } else if (valor < no.valor) {
87                  return funcaoLocalizarNo(no.filho_esquerdo);
88              }
89          };
90
91          //chama a função recursiva
92          return funcaoLocalizarNo(no);
93      };
94
95      /*recupera o nó na estrutura da árvore binária ****/
96      this.recuperarNo = function (valor) {
97          var no = this.raiz;
98
99          var funcaoLocalizarNo = function (no) {
100             if (!no) return false;
101
102             if (valor === no.valor) {
103                 return no;
104             } else if (valor > no.valor) {
105                 return funcaoLocalizarNo(no.filho_direito);
106             } else if (valor < no.valor) {
107                 return funcaoLocalizarNo(no.filho_esquerdo);
108             }
109         };
110
111         //chama a função recursiva
112         return funcaoLocalizarNo(no);
113     };
```

Os métodos são semelhantes no processo de busca, pois ambos percorrem toda a árvore de forma recursiva até localizar o valor desejado, porém, diferem um do outro no valor que cada método pode retornar. O método `contemNo` retorna um operador lógico, sendo `true` se o nó existir na árvore ou `false` caso o nó são exista. O método `recuperarNo` tem como retorno o operador lógico `false` quando não é encontrado o nó ou o próprio nó quando este existe na estrutura da árvore.

Outro método interessante pode ser observado na Figura 38, onde todo o código necessário para realizar os percursos sobre a estrutura da árvore binária é apresentado.

Figura 38 - Código percurso pré, em e pós ordem

```
115     /*percurso em pre-ordem, em-ordem e pos-ordem ****/
116     this.percursoPreEmPosOrdem = function (tipo) {
117         //variavel para armazenar o resultado da interação na estrutura da árvore
118         var resultado = [];
119         var no = this.raiz;
120
121         var funcaoPercursoPreEmPosOrdem = function (no, tipo) {
122             tipo == 'pre' && resultado.push(no.valor);
123             no.filho_esquerdo && funcaoPercursoPreEmPosOrdem(no.filho_esquerdo, tipo);
124             tipo == 'em' && resultado.push(no.valor);
125             no.filho_direito && funcaoPercursoPreEmPosOrdem(no.filho_direito, tipo);
126             tipo == 'pos' && resultado.push(no.valor);
127         };
128
129         //chama a função recursiva
130         funcaoPercursoPreEmPosOrdem(no, tipo);
131
132         return resultado;
133     };
```

Como pode ser observado, Figura 38, as operações de percurso em pré-ordem, em-ordem e pós-ordem são realizadas em uma única função. O que define o tipo de percurso da árvore é o valor do parâmetro passado para o método e a ordem em que o valor do nó é coletado antes de passar para o próximo nó. Todo o processo é realizado de forma recursiva, ou seja, o método é invocado para cada nó pertencente a estrutura da árvore.

A seguir, Figura 39, é apresentado o processo para encontrar o menor e o maior nó da árvore.

Figura 39 - Código maior e menor nó

```
135     /*retorna o menor ou maior valor da árvore ****/
136     this.menorMaiorValor = function (noInicial, tipo) {
137
138         var funcaoMenorMaiorValor = function (no, tipo) {
139
140             if (tipo == 'menor') //se o tipo for menor
141             //verifica se tem filho esquerdo, se sim então chama a função novamente passando
142             //por parâmetro o filho esquerdo. se não tem filho esquerdo então o menor valor
143             //será o valor do nó atual.
144                 return !no.filho_esquerdo ? no.valor : funcaoMenorMaiorValor(no.filho_esquerdo, tipo);
145             else if (tipo == 'maior') //se o tipo for maior
146             //verifica se tem filho direito, se sim então chama a função novamente passando
147             //por parâmetro o filho direito. se não tem filho direito então o maior valor
148             //será o valor do nó atual.
149                 return !no.filho_direito ? no.valor : funcaoMenorMaiorValor(no.filho_direito, tipo);
150         };
151
152         //chama a função recursiva
153         return funcaoMenorMaiorValor(noInicial, tipo);
154     };
```

Uma lógica de estrutura semelhante é utilizada para localizar o maior e o menor nó da árvore. O parâmetro `tipo` determina qual valor do nó deve ser retornado e a ordem em que os nós deverão ser acessados. Se o aluno deseja obter o menor nó, então o método é chamado e, de forma recursiva, acessa todos os nós esquerdos, partindo da raiz da árvore. O processo só

termina quando chega ao nó mais à esquerda da estrutura da árvore. Este nó conterá o menor valor de todos os nós da estrutura da árvore. Uma lógica similar é adotada para localizar o maior nó, porém neste processo são acessados todos os nós direitos do nó raiz.

A seguir, Figura 40 apresenta o método utilizado para localizar e retornar todos os nós folhas da árvore.

Figura 40 - Código nó folha

```
208     //retornar os nós folhas da esquerda para a direita
209     this.noFolha = function () {
210         var resultado = [];
211         var no = this.raiz;
212
213         var funcaoNoFolha = function (no) {
214             !no.filho_esquerdo && !no.filho_direito && resultado.push(no.valor);
215             no.filho_esquerdo && funcaoNoFolha(no.filho_esquerdo);
216             no.filho_direito && funcaoNoFolha(no.filho_direito);
217         };
218
219         funcaoNoFolha(no);
220
221         return resultado;
222     };
```

A Figura 40 mostra o método e o processo recursivo para localizar os nós folhas da estrutura da árvore. No processo todos os nós são visitados, verificando se possuem nós filhos. Se o nó não possui filhos, este nó é classificado como nó folha sendo armazenado na lista `resultado`. Ao final do processo a lista é retornada, sendo aplicada uma animação em todos seus nós.

Para selecionar os nós ancestrais de um determinado nó, foi criado o método apresentado na Figura 41.

Figura 41 - Nó ancestral

```
248     //retorna os nós ancestrais do nó ****
249     this.noAncestal = function (valor) {
250         var resultado = [];
251
252         if (this.contemNo(valor)) {
253             var no = this.raiz;
254             resultado.push(no.valor);
255
256             var funcaoNoAncestral = function (no, valor) {
257
258                 if (no.filho_esquerdo && valor < no.valor)
259                     resultado.push(no.filho_esquerdo.valor);
260                 else if (no.filho_direito && valor > no.valor)
261                     resultado.push(no.filho_direito.valor);
262
263                 no.filho_esquerdo && valor < no.valor && funcaoNoAncestral(no.filho_esquerdo, valor);
264                 no.filho_direito && valor > no.valor && funcaoNoAncestral(no.filho_direito, valor);
265             };
266
267             funcaoNoAncestral(no, valor);
268         }
269
270         return resultado.reverse();
271     };
```

O ponto inicial para o mapeamento é a raiz da árvore, sendo verificado a cada nível se o valor é maior ou menor que o valor do nó passado por parâmetro até que seja localizado o nó da extremidade final. Todos os nós são armazenados em uma lista e, ao final do processo, é utilizado o método `reverse` para reverter a sequência da lista.

A Figura 42 apresenta a estrutura do método utilizado para retornar todos os nós descendentes.

Figura 42 - Descendente

```
273     //retorna os nós descentente do nó
274     this.noDescendente = function (valor) {
275         var resultado = [];
276         var no = this.recuperarNo(valor);
277
278         if (no) {
279             var funcaoNoDescendente = function (no, valor) {
280                 valor != no.valor && resultado.push(no.valor);
281                 no.filho_esquerdo && funcaoNoDescendente(no.filho_esquerdo, valor);
282                 no.filho_direito && funcaoNoDescendente(no.filho_direito, valor);
283             };
284
285             funcaoNoDescendente(no, valor);
286         }
287
288         return resultado;
289     };
```

Os nós descendentes de um dado nó são todos os nós que estão abaixo deste nó. Para realizar esta seleção foi utilizado um processo recursivo que visita todos os nós esquerdos e direitos de um dado nó, armazenando todos estes nós em uma lista `resultado`. Processo semelhante foi utilizado para selecionar somente os nós da esquerda ou direita.

A Figura 43 apresenta o método utilizado para determinar se a árvore é estritamente binária.

Figura 43 - Estritamente Binária

```
409     //arvore estritamente binária
410     this.estritamenteBinaria = function () {
411         var resultado = true;
412         var no = this.raiz;
413
414         var funcaoEstritamenteBinaria = function (no) {
415             resultado = ((!no.filho_esquerdo && no.filho_direito) || (no.filho_esquerdo && !no.filho_direito)) ? false : true;
416             resultado && no.filho_esquerdo && funcaoEstritamenteBinaria(no.filho_esquerdo);
417             resultado && no.filho_direito && funcaoEstritamenteBinaria(no.filho_direito);
418         };
419
420         funcaoEstritamenteBinaria(no);
421
422         return resultado;
423     };
```

A característica da árvore estritamente binária é possuir nenhum ou dois filhos independente dos níveis que estejam. Para realizar esta verificação na árvore da aplicação foi criado o método `estritamenteBinaria` que, de forma recursiva, visita todos os filhos verificando se o nó possui apenas um único filho, independente se este filho é esquerdo ou direito. Caso ocorra de encontrar tal situação na estrutura da árvore o resultado da função será retornado como `falso`.

4.9.3 Classe Árvore Animação

Nesta seção serão apresentados alguns métodos que fazem parte da estrutura da classe "`arvore_animacao.js`". A classe contém todas as funções necessárias para criar a árvore gráfica e realizar as animações utilizadas na aplicação. Todavia serão apresentados somente os principais métodos e as variáveis utilizadas durante sua execução.

A Figura 44 apresenta todas as variáveis declaradas na classe que são utilizadas durante a execução dos métodos.

Figura 44 - Variáveis da classe árvore animação

```
2     this.posicaoInicialXNo = this.posicaoInicialYNo = 10;
3     this.distanciaPadraoHorizontal = this.distanciaPadraoVertical = 80;
4     this.raioNo = 60;
5     this.corPreencimentoNo = "#0088ff80";
6     this.corPreencimentoNoExcluido = "#ff0003";
7     this.corPreencimentoNoSelecionado = "#ff5f00";
8     this.corBordaNoSelecionado = "#ff0c00";
9     this.corPreencimentoNoSelecionadoInicio = "#ffa900";
10    this.corBordaNo = "#005eff";
11    this.corBordaNoBusca = "#00ff11";
12    this.corLinhaNivel = "#ADD8E6";
13    this.larguraBordaNo = 2;
14    this.corRotuloNo = "#000000";
15    this.corRotuloNivelLinha = "#000000";
16    this.tamanhoFonteRotuloNo = 16;
17    this.tamanhoFonteRotuloNivelLinha = 15;
18    this.corLinhaAresta = '#005eff';
19    this.espessuraAresta = 2;
20    this.espessuraLinhaNivel = 1;
21    this.linhaNivelArvore = [];
22    this.animacao = false;
```

As variáveis auxiliam na apresentação dos elementos gráficos da árvore, permitindo alterações no visual destes elementos de forma rápida e prática. Isto porque, as alterações de seus valores refletem instantaneamente na apresentação destes elementos. Por exemplo, se há intensão de alterar a cor de preenchimento dos nós, basta atribuir o valor da nova cor em hexadecimal a variável `corPreenchimentoNo`.

A Figura 45 apresenta o método que é chamado logo após criar um novo nó na árvore em código *Javascript*.

Figura 45 - Código criar nó gráfico

```
this.criarNoGraficamento = function (no) {
    if (no) {
        var noGraf = area_desenho.circle(this.raioNo).fill(this.corPreencimentoNo).stroke(this.corBordaNo).stroke(
            {width: this.larguraBordaNo}).move(this.posicaoInicialXNo, this.posicaoInicialYNo).id('circ_' + no.valor);

        //guarda a referencia do elemento grafico que representa o nó, no nó da árvore
        no.noGrafico = noGraf;
        no.noRotulo = area_desenho.text(no.valor.toString()).move(this.posicaoInicialXNo + 25, this.posicaoInicialYNo + 18).font({
            family: 'Arial',
            size: this.tamanhoFonteRotuloNo,
            anchor: 'middle',
            fill: this.corRotuloNo,
            cursor: 'default'
        }).id('rot_' + no.valor);

        //adiciona o evento click no nó
        no.noGrafico.mousedown(eventoClickVertice);
        no.noRotulo.mousedown(eventoClickVertice);

        //verifica o nível do nó. Caso seja o primeiro nível então é o nó raiz
        if (bst.nivelNo(no.valor) == 0) {
            var widthArea = document.getElementById('areadesenho').clientWidth;
            no.posX = widthArea / 4;
            no.posY = this.posicaoInicialYNo;

            //cria a animação de movimento do nó
            this.criarAnimacaoMovimentacaoPadrao(no, no.posX, no.posY, true);
        } else if (no.no_pai && no.no_pai.filho_esquerdo && no.no_pai.filho_esquerdo.valor == no.valor) {
            no.posX = no.no_pai.posX - this.distanciaPadraoHorizontal;
            no.posY = no.no_pai.posY + this.distanciaPadraoVertical;

            //indentifica qual o caminho que o novo nó deverá pecorrer até
            //chegar na posição que ficará
            var caminho = this.getCaminhoNo(no.valor);
            //cria a animação de movimento do nó
            //this.criarAnimacaoMovimentacaoCaminhoNo(no, caminho);
            this.criarAnimacaoMovimentacaoPadrao(no, no.posX, no.posY, false);

            this.verificaPosicaoNoArvoreDireitaEsquerda(no);
        } else if (no.no_pai && no.no_pai.filho_direito && no.no_pai.filho_direito.valor == no.valor) {
            no.posX = no.no_pai.posX + this.distanciaPadraoHorizontal;
            no.posY = no.no_pai.posY + this.distanciaPadraoVertical;

            //indentifica qual o caminho que o novo nó deverá pecorrer até
            //chegar na posição que ficará
            var caminho = this.getCaminhoNo(no.valor);
            //cria a animação de movimento do nó
            //this.criarAnimacaoMovimentacaoCaminhoNo(no, caminho);
            this.criarAnimacaoMovimentacaoPadrao(no, no.posX, no.posY, false);

            this.verificaPosicaoNoArvoreDireitaEsquerda(no);
        }

        //verifica se o menor nó está dentro da área de desenho
        //em outras palavras, verifica se o X no menor nó e maior que 0;
        this.verificaLimiteEsquerda();
    }
};
```

A Figura 45 apresenta o método que cria o elemento gráfico nó na área de desenho. Após o nó ser criado na estrutura da árvore em código, o método `criarNoGraficamente` é chamado. Este método cria um elemento SVG círculo com as dimensões, cor de borda, cor de preenchimento, posição inicial x e y definidas nas variáveis declaradas na classe. Também é atribuído ao elemento um id que segue o padrão `circ_valorDoNó`.

Dando continuidade à execução do método é criado o elemento SVG `text`. Este elemento terá o valor do nó e será posicionado no centro do círculo. Em seguida, são adicionados eventos ao círculo e ao rótulo para possibilitar interação com o aluno quando este clicar sobre o nó com o botão esquerdo ou direito do mouse.

Tanto o elemento círculo quanto o elemento rótulo têm armazenadas as respectivas referencias dentro do nó. Este procedimento de guardar uma referência do nó gráfico dentro

do nó da árvore facilita sua posterior manipulação, visto que, quando o nó é recuperado também são recuperados em conjunto os elementos SVG círculo e rótulo deste nó.

O passo seguinte, é chamar o método que posiciona o nó dentro da área de desenho. O método posiciona o nó adicionando uma animação de movimento com base nos valores `posX` e `posY` do nó. Para finalizar o processo dois outros métodos são chamados:

- `VerificaPosicaoNoArvoreDireitaEsquerda`: verifica se o nó pode ficar na posição definida inicialmente, ou seja, se o nó não contraria as regras de posicionamento definidas. As regras de posicionamento, são: o nó não pode sobrepor outro nó; a aresta do nó não pode cruzar com aresta de outro nó; e não pode existir no eixo x, nó cujo valor seja maior que o valor de outro nó, e que sua posição dentro do eixo x seja menor que a posição x deste outro nó;
- `VerificaLimiteEsquerda`: verifica se existe algum nó nas ramificações esquerda da árvore que esteja fora dos limites da área de desenho. Para isso o método visita todos os nós do lado esquerdo da árvore verificando sua posição x. Ao final da interação verifica se o valor `posX` do nó mais à esquerda retornado é menor que 0. Caso seja menor que 0, todos os nós são movidos para a direita até que todos os elementos estejam posicionados dentro da área de desenho. A verificação só ocorre do lado esquerdo pois do lado direito existe a barra de rolagem que permite ao aluno visualizar os elementos fora da área de visualização à direita.

A Figura 46, apresenta o código do método `criarAnimaçãoMovimentoPadrao`. O método é responsável por criar a animação de movimento que posiciona o nó dentro da área de desenho.

Figura 46 - Código animação movimento nó

```
181     //cria a animação de movimentação dos nós
182     this.criarAnimacaoMovimentacaoPadrao = function (no, x, y, raiz) {
183         var array = {
184             ease: '>',
185             delay: '.2s',
186             duration: 1000,
187             elastic: true
188         };
189         var rNo = this.raioNo;
190         var cLinhaAresta = this.corLinhaAresta;
191         var eAresta = this.espessuraAresta;
192 
193         if (raiz) {
194             //anima o círculo 'nó'
195             no.noGrafico.animate(array).move(x, y).queue(function () {
196                 this.dequeue();
197             });
198         } else {
199             //executa uma determinada ação no final da animação
200             no.noGrafico.animate(array).move(x, y).after(function () {
201                 criarArestaNo(no.no_pai, no, rNo, cLinhaAresta, eAresta);
202             });
203         }
204 
205         //anima o rotulo
206         no.noRotulo.animate(array).move(no.posX + 29, no.posY + 16).queue(function () {
207             this.dequeue();
208         });
209     };
```

Para que a animação de posicionamento do nó possa acontecer é utilizado o método `animate`. O método `animate` recebe atributos e qual animação será utilizada. No exemplo foi utilizada a animação, `move`, movimento. No método é possível observar duas situações: animação para o nó raiz, que apenas cria uma animação de movimento para o elemento nó corresponde; e animação para nó filho, que difere um pouco da primeira situação no fato de que, ao final da animação de movimento, método `after`, é chamado o método `criarArestaNo`. O método `criarArestaNo` é responsável por criar a linha, aresta, que indica a hierarquia entre o nó pai e seu nó filho. Ao final do processo é atribuído ao rótulo do nó também uma animação de movimento para que o rótulo possa acompanhar o nó durante o processo de animação.

A Figura 47 apresenta o método responsável por organizar os nós da árvore, de tal forma que não fiquem posicionados inadequadamente na área de desenho.

Figura 47 - Código função verifica posição nó

```
228     //verifica o lado que o nó está na árvore para assim realizar a animação de ajuste de posição ****
229     this.verificaPosicaoNoArvoreDireitaEsquerda = function (no) {
230         var atualizaPosicao = false;
231
232         //verifica se o valor e valor que o valor do nó raiz, se sim, então o nó
233         //faz parte da sub-árvore direta do nó raiz
234         if (no.valor > bst.raiz.valor) {
235             if (no.posX == bst.raiz.posX || no.posX < bst.raiz.posX) {
236                 //se a posição x do nó novo for menor que a posição x do nó raiz e
237                 //este novo novo pertence a árvore do lado direito da raiz, então
238                 this.mudarPosicaoXDireitaNo(bst.raiz.filho_direito, null);
239             } else if (no.no_pai && no.no_pai.valor != bst.raiz.valor) {
240
241                 //enquanto tiver uma no resultante executar o while
242                 while (noResultado = this.verificaPosicaoNovoNo(no, 'D')) {
243                     atualizaPosicao = true;
244                     if (no.valor > noResultado.valor)
245                         this.atualizarPosicaoXNoDireito(noResultado);
246                     else
247                         this.atualizarPosicaoXNoDireito(no);
248                 }
249
250                 atualizaPosicao && this.animacaoMovimentoAjustePosicaoNo(bst.raiz.filho_direito);
251             }
252         } else {
253             if (no.posX == bst.raiz.posX || no.posX > bst.raiz.posX) {
254                 //se a posição x do nó novo for maior que a posição x do nó raiz e
255                 //este novo novo pertence a árvore do lado esquerdo da raiz, então
256                 this.mudarPosicaoXEsquerdaNo(bst.raiz.filho_esquerdo, null);
257             } else if (no.no_pai && no.no_pai.valor != bst.raiz.valor) {
258
259                 //enquanto tiver uma no resultante executar o while
260                 while (noResultado = this.verificaPosicaoNovoNo(no, 'E')) {
261                     atualizaPosicao = true;
262
263                     if (no.valor < noResultado.valor)
264                         this.atualizarPosicaoXNoEsquerdo(noResultado);
265                     else
266                         this.atualizarPosicaoXNoEsquerdo(no);
267                 }
268
269                 atualizaPosicao && this.animacaoMovimentoAjustePosicaoNo(bst.raiz.filho_esquerdo);
270             }
271         }
272     };
```

O método apresenta as rotinas necessárias para posicionar os nós da árvore de modo a não se sobrepor. Toda vez que é inserido um nó na estrutura da árvore, o método `verificaPosicaoNoArvoreDireitaEsquerda` é chamado para reposicionar os nós da árvore.

A lógica adotada foi dividir a árvore em dois lados, esquerdo e direito. Nenhum nó do lado esquerdo da árvore pode ter a posição x do nó maior que a posição x do nó raiz e nenhum nó da árvore do lado direito pode ter a posição x do nó menor que a posição x do nó raiz, ou seja, a regra básica é os nós do lado esquerdo não se cruzarem em nenhum momento com os nós do lado direito.

Quando é inserido um nó na árvore, verifica-se sua posição em relação ao nó raiz. Se a posição x do nó contraria o lado da árvore que deveria estar posicionado, então a função `mudarPosicaoXDiretaNo` ou `mudarPosicaoXEsquerdaNo` é chamada para reposicionar

todos os nós do lado da árvore correspondente ao nó. Ao final da rotina todos os nós estarão posicionados no lado correto da árvore.

Um laço de repetição foi criado para validar se o nó pode ficar na posição definida inicialmente. O laço chama o método responsável por alterar o valor da posição x do nó e de todos os nós que tenham o valor maior que deste nó, até que todos estejam em uma posição válida. Ao final do laço é atribuída aos nós a animação de movimento para então serem reposicionados dentro da área de desenho.

A Figura 48 apresenta o método criado na classe animação para remover o nó excluído.

Figura 48 - Método excluir nó gráfico

```
this.removerNoGraficoArvore = function (noBase, noRemovido) {
    if (noRemovido && noBase) {
        //se for nó folha
        if (!noRemovido.filho_esquerdo && !noRemovido.filho_direito && noRemovido.qtdFilho != 2) {
            noRemovido.noGrafico && noRemovido.noGrafico.animate(500).fill(arvore_animacao.corPreencimentoNoExcluido).animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg círculo
                var circG = document.getElementById(this.id());
                circG && document.getElementById('svgAreaDesenho').removeChild(circG);
            });
            noRemovido.noRotulo && noRemovido.noRotulo.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });
            noRemovido.aresta_pai && noRemovido.aresta_pai.animate(100).opacity(0).after(function () {
                //no final da animação excluir o elemento svg aresta pai
                var arestG = document.getElementById(this.id());
                arestG && document.getElementById('svgAreaDesenho').removeChild(arestG);
            });
        } else if (!noRemovido.filho_esquerdo && noRemovido.filho_direito && noRemovido.qtdFilho != 2) {
            noRemovido.noGrafico && noRemovido.noGrafico.animate(500).fill(arvore_animacao.corPreencimentoNoExcluido).animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg círculo
                var circG = document.getElementById(this.id());
                circG && document.getElementById('svgAreaDesenho').removeChild(circG);
            });
            noRemovido.noRotulo && noRemovido.noRotulo.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });

            !noRemovido.no_pai && noRemovido.aresta_esquerda && noRemovido.aresta_esquerda.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });
            !noRemovido.no_pai && noRemovido.aresta_direita && noRemovido.aresta_direita.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });
        } else if (noRemovido.filho_esquerdo && !noRemovido.filho_direito && noRemovido.qtdFilho != 2) {
            noRemovido.noGrafico && noRemovido.noGrafico.animate(500).fill(arvore_animacao.corPreencimentoNoExcluido).animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg círculo
                var circG = document.getElementById(this.id());
                circG && document.getElementById('svgAreaDesenho').removeChild(circG);
```

```
            });
            noRemovido.noRotulo && noRemovido.noRotulo.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });

            !noRemovido.no_pai && noRemovido.aresta_esquerda && noRemovido.aresta_esquerda.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });
            !noRemovido.no_pai && noRemovido.aresta_direita && noRemovido.aresta_direita.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });
        } else if (noRemovido.qtdFilho == 2) {
            //Se possue dois filhos
            if (noRemovido.noGrafico && noRemovido.excluirAresta) {
                var arestG = document.getElementById(noRemovido.excluirAresta.id());
                arestG && document.getElementById('svgAreaDesenho').removeChild(arestG);
            }

            noRemovido.noGrafico && noRemovido.noGrafico.animate(500).fill(arvore_animacao.corPreencimentoNoExcluido).animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg circulo
                var circG = document.getElementById(this.id());
                circG && document.getElementById('svgAreaDesenho').removeChild(circG);
            });
            noRemovido.noRotulo && noRemovido.noRotulo.animate(100).opacity(0).size(0, 0).after(function () {
                //no final da animação excluir o elemento svg rótulo
                var rotG = document.getElementById(this.id());
                rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
            });
        }

        var widthArea = document.getElementById('areadesenho').clientWidth;
        var posicaoNoRaiz = widthArea / 4;

        noBase.posX = posicaoNoRaiz;

        movimentarNoParaADireitaOuEsquerda(noBase, noBase.posX, noBase.posY);

        var funcaoAtualizaPosicaoXYNoEsquerdoDireito = function (n, valorRemovido) {
            if (n.no_pai && n.valor < n.no_pai.valor) {
                n.posX = n.no_pai.posX - arvore_animacao.distanciaPadraoHorizontal;
                n.posY = n.no_pai.posY + arvore_animacao.distanciaPadraoVertical; //testa
            } else if (n.no_pai) {
                n.posX = n.no_pai.posX + arvore_animacao.distanciaPadraoHorizontal;
                n.posY = n.no_pai.posY + arvore_animacao.distanciaPadraoVertical; //testa
            }

            n.filho_esquerdo && funcaoAtualizaPosicaoXYNoEsquerdoDireito(n.filho_esquerdo, valorRemovido);
            n.filho_direito && funcaoAtualizaPosicaoXYNoEsquerdoDireito(n.filho_direito, valorRemovido);
        };

        noBase && noBase.filho_esquerdo && funcaoAtualizaPosicaoXYNoEsquerdoDireito(noBase.filho_esquerdo, noRemovido.valor);
        noBase && noBase.filho_direito && funcaoAtualizaPosicaoXYNoEsquerdoDireito(noBase.filho_direito, noRemovido.valor);

        var funcaoAtualizaPosicaoPosicaoNo = function (n) {
            arvore_animacao.verificaPosicaoNoArvoreDireitaEsquerda(n);

            n.filho_esquerdo && funcaoAtualizaPosicaoPosicaoNo(n.filho_esquerdo);
            n.filho_direito && funcaoAtualizaPosicaoPosicaoNo(n.filho_direito);
        };

        noBase && noBase.filho_esquerdo && funcaoAtualizaPosicaoPosicaoNo(noBase.filho_esquerdo);
        noBase && noBase.filho_direito && funcaoAtualizaPosicaoPosicaoNo(noBase.filho_direito);

        bst.raiz.filho_esquerdo && this.animacaoMovimentoAjustePosicaoNo(bst.raiz.filho_esquerdo);
        bst.raiz.filho_direito && this.animacaoMovimentoAjustePosicaoNo(bst.raiz.filho_direito);

        //verifica se o menor nó está dentro da área de desenho
        //em outras palavras, verifica se o X no menor nó e maior que 0;
        //noRemovido.valor < noBase.valor && this.verificaLimiteEsquerda();
        this.verificaLimiteEsquerda();
    } else if (noBase == null && noRemovido.id == 'raiz') {
        noRemovido.noGrafico && noRemovido.noGrafico.animate(500).fill(arvore_animacao.corPreencimentoNoExcluido).animate(100).opacity(0).size(0, 0).after(function () {
            //no final da animação excluir o elemento svg circulo
            var circG = document.getElementById(this.id());
            circG && document.getElementById('svgAreaDesenho').removeChild(circG);
        });
        noRemovido.noRotulo && noRemovido.noRotulo.animate(100).opacity(0).size(0, 0).after(function () {
            //no final da animação excluir o elemento svg rótulo
            var rotG = document.getElementById(this.id());
            rotG && document.getElementById('svgAreaDesenho').removeChild(rotG);
        });
    }
};
}
```

O método apresentado na Figura 48 trata a exclusão do nó por sua quantidade de filhos. O primeiro passo verifica se o nó a ser excluído não possui filhos, ou seja, se trata de um nó folha. O processo é bem simples, bastando apenas excluir os elementos círculo, rótulo e aresta do nó na árvore gráfica.

O segundo e terceiro passo é verificar se o nó possui filho direito ou filho esquerdo, respectivamente nesta ordem e, por último verificar se o nó possui dois filhos. Os passos citados são condicionais e apenas uma dessas condicionais será verdadeira, ou seja, somente

uma das condicionais será executada para cada chamada do método. Em todas as etapas são excluídos os elementos visuais desnecessários ficando apenas o necessário na área de desenho. Por fim, as posições dos nós são recalculadas e ao final deste recálculo, são atribuídas animações de movimento para o reposicionamento dos nós.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi proposta e desenvolvida uma aplicação web que permite desenhar árvores binárias de busca e obter informações sobre o desenho criado. O desenvolvimento da aplicação foi motivado pela necessidade do grupo de pesquisa de poder trabalhar detalhes da implementação de uma ferramenta com recursos que permitam entender o conceito árvore binária de busca de forma mais dinâmica, auxiliando, com isso, o professor no ensino do respectivo conceito.

Outro ponto observado para buscar esta implementação deve-se à dificuldade que geralmente os alunos da disciplina possuem em entender e visualizar o conceito desta estrutura. Com o uso da aplicação o professor terá a oportunidade de dar ênfase ao ensino de maneira mais dinâmica, intuitiva e lúdica. O fato de ser uma ferramenta desenvolvida na própria instituição permitirá que adequações sejam feitas na mesma para que se adeque às necessidades da disciplina bem como ocorram inserções de novas funcionalidades.

Para desenvolvimento da interface da aplicação foi utilizado o *framework Bootstrap*, que permitiu criar uma interface responsiva e dentro dos padrões da W3C. Para criar a área de desenho foi utilizado o *framework* SVG, que permitiu criar os elementos gráficos nós, vértices, rótulos, além de das animações de movimento e efeitos de transição utilizadas sobre a árvore desenhada.

No desenvolvimento da aplicação muita atenção e tempo de desenvolvimento foi destinada a três rotinas: posicionamento dos nós na área de desenho, sendo necessário criar um algoritmo que organiza os nós de modo que não ocorra sobreposições e trate os cruzamentos das arestas; exclusão de nós com dois filhos, por ser necessário reorganizar a árvore gráfica de modo a permanecer uma árvore binária de busca, obedecendo às regras de não sobreposições de nós e não cruzamentos de arestas; a animação dos percursos pré-ordem, em-ordem e pós-ordem, sendo necessário mapear o caminho ida e volta de todos os nós da árvore e registrar o exato momento que deverá marcar o nó de acordo com o percurso selecionado.

Ao final do desenvolvimento foram realizados testes de funcionalidades junto ao professor responsável, onde a aplicação foi submetida a testes de validação para verificar e analisar se as informações retornadas pelo módulo de informação estavam corretas. Como resultado tem-se uma aplicação finalizada e funcional, podendo ser utilizada pelo professor da disciplina e por seus alunos.

Para trabalhos futuros, novas funcionalidades podem ser adicionadas à aplicação para assim melhorar a experiência do professor e aluno, como por exemplo: sistema de *login* adicionado à possibilidade de salvar a árvore criada pelo aluno no banco de dados e recursos para recarregá-la quando necessário; aperfeiçoar a função responsável por organizar os nós na área de desenho; criar novas animações melhorando a experiência do professor e do aluno na aplicação e implementar recursos que possibilitem o funcionamento da aplicação em dispositivos móveis, visto que não foi projetada para tais aparelhos.

6 REFERÊNCIAS

ASCÊNCIO, Ana Fernanda Gomes. **Estruturas de dados**: algoritmos, análise da complexidade e implementações em JAVA e C/C++ / Ana Fernanda Gomes Ascencio, Graziela Santos de Araújo. – São Paulo: Pearson Prentice Hall, 2010.

FALKEMBACH, G. A. M. **Aprendizagem De Algoritmos:** Dificuldades na Resolução de Problemas. Tese (Doutorado). Porto Alegre: PGIE/UFRGS, 2013.

GOMES, A., HENRIQUES, J., MENDES, A. J. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. **Educação, Formação & Tecnologias**; vol.1(1), pp. 93-103, 2008. Disponível em: <http://eft.educom.pt/index.php/eft/article/view/23/16>. Acesso em: 10 de agosto de 2018.

HOROWITZ, Ellis; SAHNI, Sartaj. **Fundamentos de estruturas de dados**. Trad. de Thomasz R. Rawicki. Rio de Janeiro: Campus, 1984.

JESUS, E. A., RABEE, A. L. A. (2010). Avaliação Empírica da Utilização de um Jogo para Auxiliar a Aprendizagem de Programação, **Anais do XXI Simpósio Brasileiro de Informática na Educação**, João Pessoa.

LAFORE, Robert (Robert W.). **Aprenda em 24 horas estrutura de dados e algoritmos**. Rio de Janeiro: Campus, 1999.

LAUREANO, Marcos. **Estruturas de dados com algoritmos e C**. Curitiba: BRASPOR, 2012.

LOPES, Arthur Vargas. **Estruturas de dados**: Para a Construção de Software / Arthur Vargas Lopes. – Canoas: Ed. ULBRA, 1999.

SVG.js. **SVG.js**. Disponível em: <http://svgjs.com/>. Acesso em: 12 jan. 2019.

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. **Estruturas de dados e seus algoritmos**. Rio de Janeiro: LTC, 1994. XIV, 320 p.

TENENBAUM, Aaron Ai; LANGSAM , Yedidyah; AUGENSTEIN, Moshe J. **Estruturas de Dados Usando C**. São Paulo: McGraw-Hill, In, 1995. p.303.

VELOSO, Paulo et al. **Estruturas de dados**. Rio de Janeiro: Campus, 1986.