**CENTRO UNIVERSITARIO SÃO LUCAS EDUCACIONAL AFYA**

SISTEMAS DE INFORMAÇÃO

DANIEL SOARES DO NASCIMENTO NETO

FELIPE GUZEN SPERB

GABRIEL FLÜGEL SILVA

Link GIT: <https://github.com/Flugelo/Algoritimo-de-Huffman>

**Estrutura de Árvores Binárias**

JI-PARANÁ/RO

2021

DANIEL SOARES DO NASCIMENTO NETO

FELIPE GUZEN SPERB

GABRIEL FLÜGEL SILVA

**Estrutura de Árvores Binárias**

Trabalho apesentado para obtenção de nota no 3º semestre do curso sistemas de informação pelo Centro Universitário São Lucas AFYA.

Orientador: Willian Alves de Oliveira Fachetti

JI-PARANÁ/RO

2021

**RESUMO**

Uma árvore binária é uma estrutura de dados útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo. Por exemplo, suponha que precisemos encontrar todas as repetições numa lista de números. Uma maneira de fazer isto é comparar cada número com todos que o precedem. Entretanto, isso envolve um grande número de comparações. O número de comparações pode ser reduzido usando-se uma árvore binária. O primeiro número na lista é colocado num nó estabelecido como a raiz de uma árvore binária com as subárvores esquerda e direita vazias. Para estabelecer as verificações e operações de caminhamento e repetição, podemos usar dos métodos disponíveis das TAD (Tipo Abstrato de Dados) para a Árvore Binária, dando maior eficiência e fluidez, tornando a estrutura de dados mais otimizada atendendo a necessidade do programa.

**Palavra-chave**: Árvore binária. Estrutura de dados. Operação.

**ABSTRACT**

A binary tree is a useful data structure when bidirectional decisions need to be made at each point in a process. For example, suppose we need to find all the repetitions in a list of numbers. One way to do this is to compare each number with all that precede it. However, this involves a large number of comparisons. The number of comparisons can be reduced using a binary tree. The first number in the list is placed in a node established as the root of a binary tree with the left and right subtrees empty. To establish the checks and operations of walking and repetition, we can use the available methods of TAD (Abstract Data Type) for the Binary Tree, giving greater efficiency and fluidity, making the data structure more optimized meeting the needs of the program.

**Keyword:** Binary tree. Data structure. Operation.

**SUMÁRIO**

1. **INTRODUÇÃO**......... ......................................................................................... 6

2. Definição de árvores.......................................................................................... 7

3. Tipos de dados.................................................................................................. 9

3.1 Tipo abstratos de dados....................................................................... 10

3.2 Árvores Binárias de Pesquisa.............................................................. 12

4. Caminhamento em Árvore............................................................................... 13

5. **CONCLUSÃO**................................................................................................... 16

**REFERÊNCIAS**.................................................................................................... 17

1. **INTRODUÇÃO**

Em ciência da computação, estrutura de dados é um modo de armazenamento, organização e manipulação de dados em um computador, para que a partir de funções e procedimentos possa utilizar de tais para elaboração de um programa. Para manipulação de dados, utiliza-se dos algoritmos e da lógica de programação, dispondo estes dados de forma organizada e coerente, caracterizado assim uma estrutura de dados. Essa formação inclui eficiência para buscas e padrões específicos de acesso, necessidades especiais para manejo de grandes volumes ou a simplicidade de implementação e uso. Alguns tipos evidentes de uso de ED são através da orientação a objeto, criação de listas, arvores binarias, pilhas, vetores e fila. Uma boa estrutura de dados caracteriza por manter consultas rápidas e dados organizados de forma hierárquica, o que reduz bastante a quantidade de comparações necessárias para se encontrar algum valor, como por exemplo a estrutura de árvore, sendo essa a estudada nesse artigo. Dessa forma ED’s, os TD’s e TAD’s são definidos indiretamente pelas operações e usos, e propriedades destas operações e usos depende do custo computacional e o espaço que pode representar e ocupar na memória.

**2. DEFINIÇÃO DE ÁRVORES**

Árvores genéricas Peritos em produtividade dizem que as mudanças se originam em pensamentos “não lineares”. Uma das estruturas de dados não lineares mais importantes da computação será estudada: as árvores. Estruturas do tipo árvore são, na verdade, uma ruptura na organização de dados, pois permitem a implementação de uma gama de algoritmos muito mais rapidamente do que usando estruturas de dados lineares, como listas. Árvores também oferecem uma forma natural de organizar os dados e, consequentemente, tornaram-se estruturas ubíquas em sistemas de arquivos, interfaces gráficas com o usuário, bancos de dados, sites da Web e outros sistemas computacionais. Não é muito claro o que os peritos querem afirmar com pensamento “não linear”, mas quando se diz que árvores são não lineares, a referência é feita a um relacionamento organizacional que é mais rico do que simplesmente “antes” e “depois” entre objetos de uma sequência. Os relacionamentos em uma árvore são hierárquicos, com alguns objetos estando “acima” e outros “abaixo”. Na verdade, a principal terminologia das estruturas de árvores vem das árvores genealógicas, e os termos “pai”, “filho”, “ancestral” e “descendente” são os mais usados para descrever os relacionamentos.

Uma árvore é um tipo abstrato de dados que armazena elementos de maneira hierárquica. Com exceção do elemento do topo, cada elemento da árvore tem um elemento pai e zero ou mais elementos filhos. Uma árvore é normalmente desenhada colocando-se os elementos dentro de elipses ou retângulos e conectando pais e filhos com linhas retas. Normalmente, o elemento topo é chamado de raiz da árvore, mas é desenhado como sendo o elemento mais alto, com todos os demais conectados abaixo, sendo exatamente como ao contrário de uma árvore na vida real.

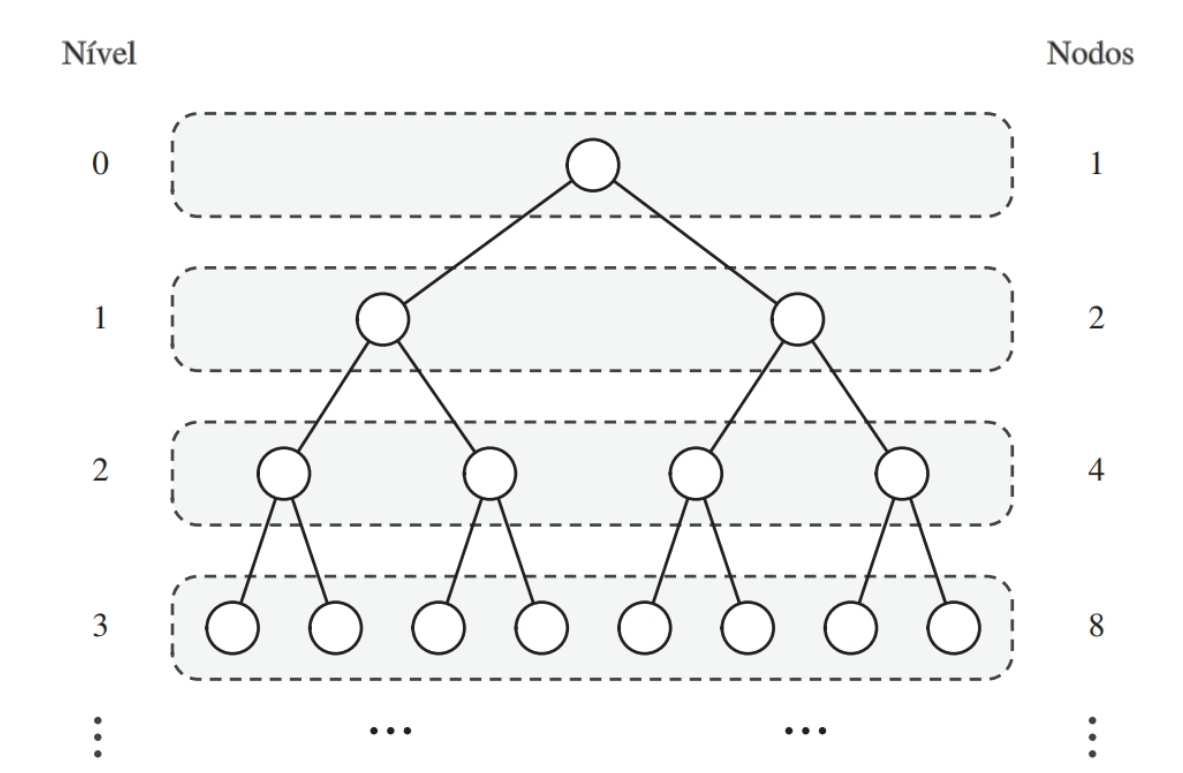
Formalmente definimos uma árvore T como um conjunto de diferentes nós que armazenam os elementos em relação a pai e filho com as seguintes propriedades:

* Se T está vazio, ele tem um nó especial chamado de raiz de T, que não pode ser pai.
* Cada nó v de T diferente da raiz tem um único nó pai, que é w; todo nó que tem pai w é filho de w.
* Dois nós que são filhos do mesmo pai são irmãos. Um nó v é externo se v não tem filhos, e uma nó v é interno se v tem um ou mais filhos. Nós internos também são conhecidos como folhas.

Observando essas situações, pode-se perceber que uma árvore pode ser vazia, o que significa que ela não terá nó. Esta convenção permite que se defina uma árvore recursivamente, de maneira que uma árvore T ou vazia, ou consiste em um nó r que é chamado de raiz de T, que é um conjunto possivelmente vazio de árvores cujas raízes são filhas de r.

Propriedades de árvores binárias

As árvores binárias tem várias propriedades interessantes quanto as relações entre sua altura e seu número de nós. Denota-se o conjunto de nós de mesma profundidade d de uma árvore T como sendo o nível d de T. Em uma árvore binária, o nível 0 tem no máximo um nó (a raiz), o nível 1 tem no máximo 2 (os filhos da raiz), o nível 2 tem no máximo 4, e assim por diante. Generalizando, pode-se dizer que o nível d tem no máximo 2d nós.



Pode-se observar que o número máximo de nós nos níveis de uma árvore binária cresce de forma exponencial à medida que se desce na árvore. A partir desta observação, podem-se derivar as seguintes propriedades relacionando a altura de uma árvore binária T com o número de nós.

Altura e Profundidade

A profundidade de v é o número de ancestrais de v excluindo o próprio v. O nó que armazena Internacional tem profundidade 2.

A profundidade de um nó v também pode ser definida recursivamente como segue:

* Se v é a raiz, então a profundidade de v é 0.
* Em qualquer outro caso, a profundidade de v é um mais a profundidade do pai de v.

Altura

A altura de um nó v em árvore T também é definida recursivamente:

* Se v é um nó externo, então a altura de v é 0;
* Em qualquer outro caso, a altura de v é um mais a altura máxima dos filhos de v;
* A altura de uma árvore não vazia T é a altura da raiz de T;
* A altura de uma árvore não vazia T é igual à profundidade máxima dos nós externos de T.

**3. TIPOS DE DADOS**

Em qualquer linguagem de programação, é importante classificar constantes, variáveis e valores de gerado para alguma função de acordo com seu tipo de dados. O tipo de dados é o que caracteriza o conjunto de valores a que uma constante pertence. O tipo de dado pode ser elementar (simples) ou estruturado(complexo), sendo o elementar caracterizado por um domínio de valores indivisíveis, como por exemplo os definidos por tipos numerais, texto e símbolo. Um tipo estruturado define uma colocação homogênea de valores ou um agregado de valores de diferentes tipos, como exemplo se tem as “listas”.

3.1 Tipos abstratos de dados

Em qualquer linguagem de programação, um tipo abstrato de dados é visto como um modelo matemático que se aplica junto a um modelo de dados e um conjunto de procedimentos que atuam especificamente sobre esses dados. Em abstração mais baixa esse procedimento é implementado por subprogramas conhecidos como operações, métodos ou serviços. Todo procedimento a ser realizado sobre os dados encapsulados pelo TAD só poderá ser executado por intermédio dos procedimentos definidos matematicamente, sendo esta restrição uma característica operacional útil nesse tipo de estrutura. Um programa baseado em TAD deverá conter algoritmos e estruturas de dados que implementam, em termos da linguagem de programação adotada, os procedimentos e os modelos de dados dos TADs utilizados pelo programa. Assim, a implementação de cada TAD pode ocupar porções bem definidas do programa: uma para a definição das estruturas de dados e outra para a definição do conjunto de algoritmos.

O TAD para a árvore binária, significa armazenar elementos em posições como de uma lista, na qual são definidas em relação as posições de seus vizinhos. Essas posições da árvore são os nós, e o posicionamento pela vizinhança satisfaz as relações pai-filho, que definem uma árvore válida. Assim, os termos “posição” e “nó” são usados com o mesmo sentido no caso de árvores.

As vantagens disso, são a utilização dos métodos de acesso do TAD árvore que retornam e associam as posições, sendo métodos como:

* root( ): Retorna a raiz da árvore;
* parent(v): Retorna o nó pai de v
* children(v): Retorna uma coleção iterável contendo os filhos do nó v.

Tais métodos facilitam a programação com árvores, tornando mais legível, podendo também usá-los nas condições de comandos if e de laços while. Outros tipos de métodos são os genéricos que uma árvore deveria suportar, que não estão necessariamente relacionados com sua estrutura, incluindo os seguintes:

* size( ): Retorna o número de nós na árvore;
* isEmpty( ): Testa se a árvore tem ou não tem algum nó;
* iterator( ): Retorna um iterado de todos os elementos armazenados nos nós da árvore;
* positions( ): Retorna uma coleção iterável com todos os nós dá árvore;
* replace(v,e): Retorna o elemento armazenado em v e o substitui por e.

Em níveis de abstração mais baixos, se há foco no nível de design, onde a estrutura passiva deve ser representada por um modelo de dados como sequência e árvore binária, e as operações devem ser especificadas através de procedimentos de forma que a representação não dependa de uma linguagem de programação. Em um nível de abstração ainda mais baixo, denominado nível de implementação, deve-se tomar como base o design do TAD e estabelecer representações concretas para os elementos de sua estrutura em termos de uma linguagem de programação específica. Os procedimentos serão implementados por meio dos algoritmos apropriados às operações raiz, esquerda, direita etc.

Uma razão importante para programar em termos de TAD é o fato de que os elementos da estrutura passiva do TAD são acessíveis somente através dos elementos da estrutura ativa. O acesso a qualquer elemento da estrutura de dados pode ocorrer apenas via os algoritmos correspondentes às operações inserir, eliminar e consultar. Essa restrição conduz a uma forma eficiente de programação defensiva, protegendo os dados encapsulados no TAD contra manipulações inesperadas por parte de outros algoritmos. Uma segunda razão, também importante, para programar em termos de TAD é o fato de que seu uso permite introduzir alterações nas estruturas definidas no nível de implementação visando o aumento de eficiência livre da preocupação de gerar erros no restante do programa. Tais erros não ocorrem porque a única conexão entre o TAD e o restante do programa é aquela constituída pela interface imutável dos algoritmos que implementam a estrutura ativa do TAD.

Por fim, pode-se dizer que um TAD bem construído pode tornar-se uma porção de código confiável e genérica, permitindo e aconselhando seu reuso em outros programas. Dessa forma, aumenta-se a produtividade na construção de programas e, sobretudo, garante-se a qualidade dos produtos gerados.

Método Height1

O Método height1 Max da classe java.lang.Math. Infelizmente não é muito eficiente. Uma vez que height1 chama o algoritmo depth(v) sobre cada nó externo v de T, o tempo de execução de height1 é dado por O(n+Σ(1+d)), onde n é o número de nós de T, d, é a profundidade do nó v e E é o conjunto de nós externos de T. No pior caso, o somatório Σ(1 + d) é proporcional a n². Logo, o algoritmo height1 executa em tempo O(n²).

Método Height2

O algoritmo height2 é mais eficiente que o height1. O algoritmo é recursivo e, se for chamado inicialmente sobre T, será́ eventualmente chamado sobre cada um dos nós de T. Logo, pode-se determinar o tempo de execução deste método somando, sobre todos os nós, o tempo gasto em cada nó (na parte não recursiva). Processar cada nó em children(v) consome tempo O(cv), onde cv denota o número de filhos do nó v. Assim, o laço while tem c, repetição, e cada repetição do laço consome tempo O(1) mais o tempo das chamadas recursivas sobre os filhos de v. Logo, o algoritmo height2 consome tempo O(1 + cv) em cada nó v, e seu tempo de execução é O(Σ v(1 + cv)).

3.2 Árvores Binárias de Pesquisa

Pode-se usar uma árvore binária de pesquisa T do conjunto S para determinar se um certo valor y se encontra em S percorrendo para baixo a árvore T começando pela raiz. Em cada nó interno v, compara-se o valor pesquisado y com o elemento x(v) armazenado em v. Se y **≤** x(v), então a pesquisa continua na subárvore da esquerda de v. Se y **=** x(v), então a pesquisa terminou com sucesso. Se y **≥** x(v), então a pesquisa continua na subárvore direita. Finalmente, se foi encontrado um nó externo, então a pesquisa terminou sem sucesso. Em outras palavras, uma árvore de pesquisa binária pode ser entendida como uma árvore binária de decisão, onde a questão formulada em cada nó interno diz respeito ao fato de o elemento armazenado naquele nó ser menor, igual ou maior que o elemento pesquisado. Na verdade, é exatamente esta correspondência com uma árvore de decisão binária que motiva a restrição de que árvores de pesquisa binária devem ser próprias (com nós externos “de armazenamento”).

Observa-se que o tempo de execução de pesquisa em uma árvore binária de pesquisa T é proporcional à altura de T.. Assim, as árvores de pesquisa binária são mais eficientes quando tem altura pequena.

Seja S um conjunto cujos elementos tem uma relação de ordem. Por exemplo, S pode ser um conjunto de inteiros. Uma árvore binária de pesquisa para S é uma árvore binária própria T tal que.

* cada nó interno v de T armazena um elemento de S denotado x(v);
* para cada nó interno v de T, os elementos armazenados na subárvore es- queda de v são menores ou iguais a x(v), e os elementos armazenados na subárvore direita de v sejam maiores ou iguais a x(v);
* Os nós externos de T não armazenam elementos.

Um caminhamento interfixado sobre uma árvore binária de pesquisa T visita os elementos em uma sequência não decrescente.

**4. CAMINHAMENTO EM ÁRVORES**

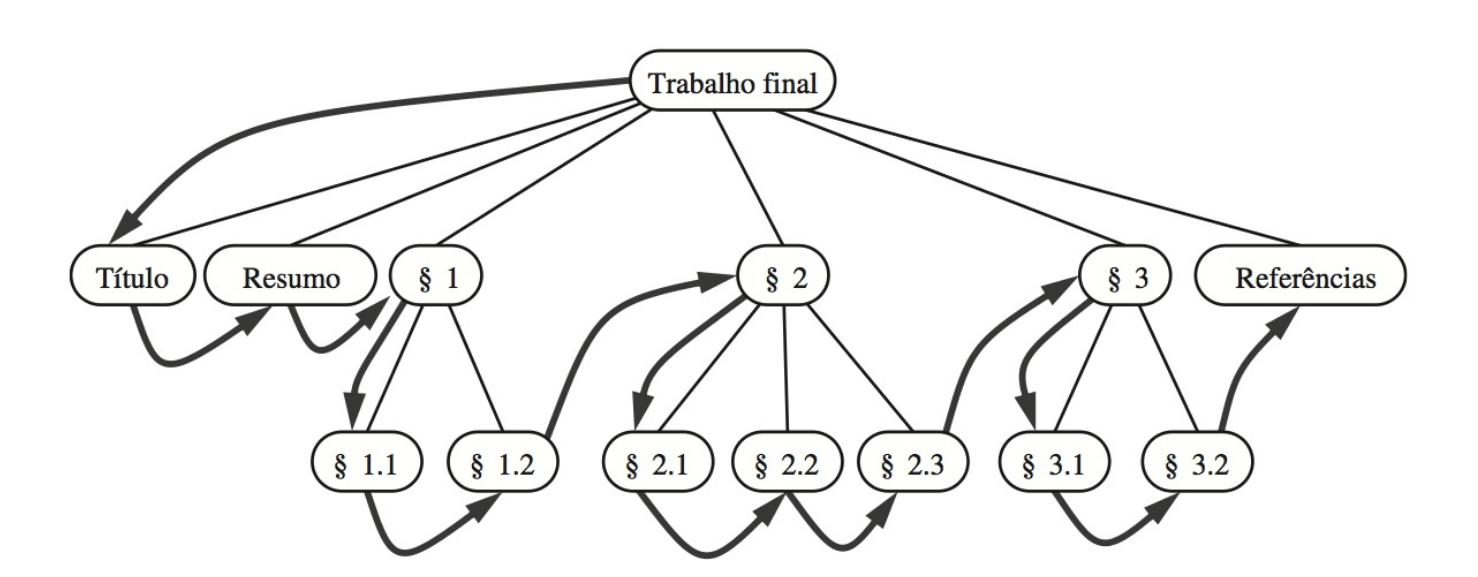
Caminhamento é o ato de percorrer todos os nós de uma árvore de forma sistemática sendo cada nó visitado uma única vez, um caminhamento completo sobre uma árvore produz uma sequência linear de nós, de modo que cada nó da árvore passa a ter um nó seguinte ou um nó anterior, ou ambos, para uma forma dada de caminhamento.

Existem três tipos de caminhamentos mais frequentes, são eles o caminhamento Pós-fixado (LRN), o caminhamento pré-fixado-fixado (NLR) e o caminhamento interfixado ou Central (LNR). Onde L = Left (esquerda), R = Right (direita) e N = Node.

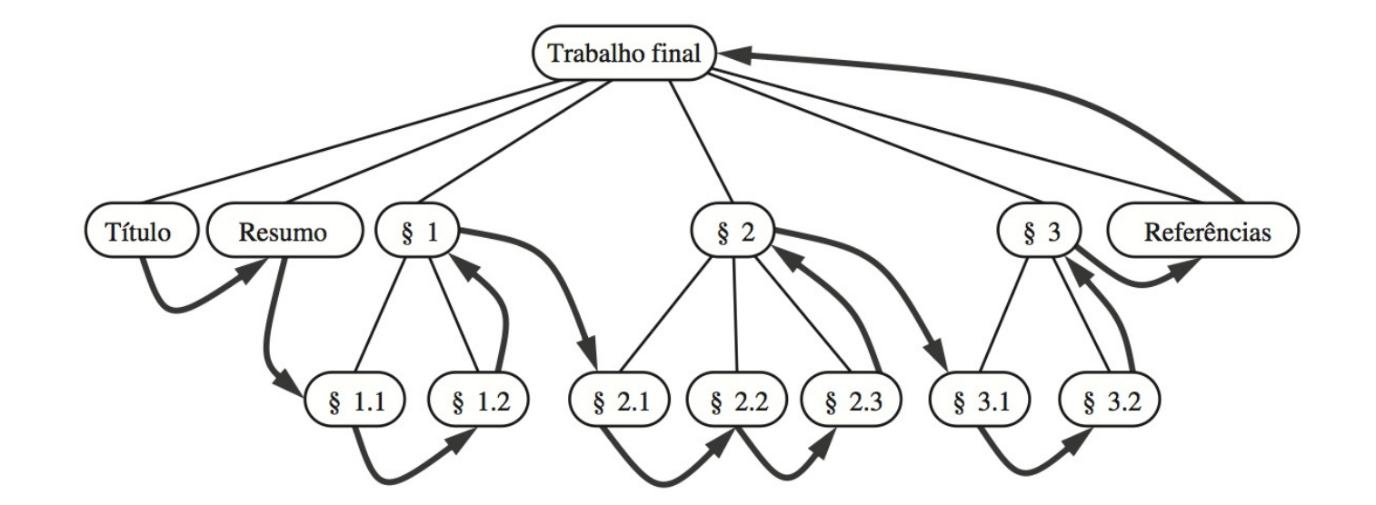
* Caminhamento Pré-fixado (NLR)

O caminhamento de uma árvore T é uma forma sistemática de acessar ou “visitar” todos os nós de T.

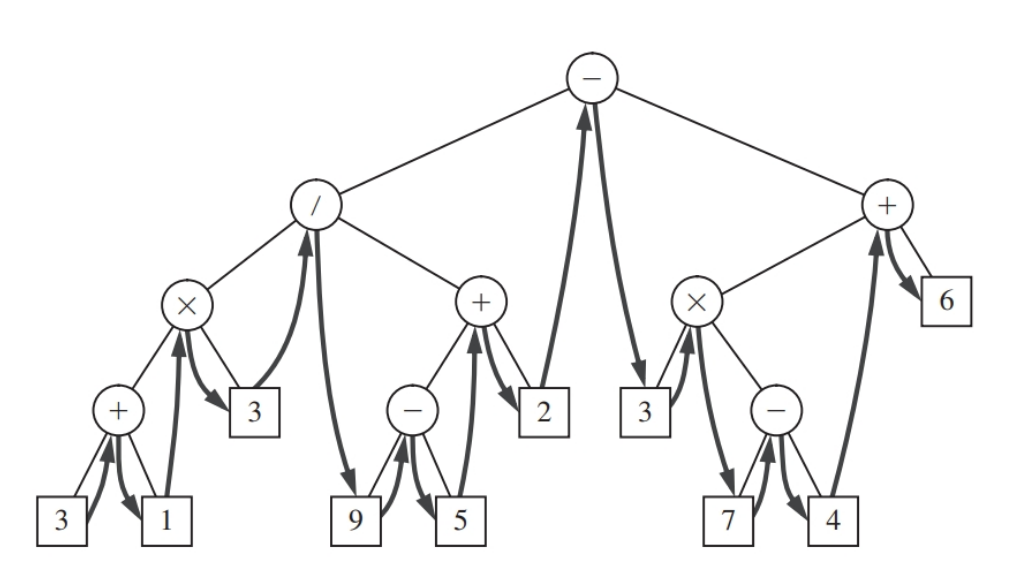
Em um caminhamento prefixado de uma árvore T, a raiz de T é visitada primeiro e, então, as subárvores, cujas raízes são seus filhos, são percorridas recursivamente. Se a árvore está ordenada, então as subárvores são percorridas de acordo com a ordem dos filhos. A ação especifica associada com a “visita” de um nó v depende da

aplicação do caminhamento, e pode envolver qualquer coisa, desde incremento de um contador até́ um cálculo complexo para v.

* Caminhamento Pós-fixado (LRN)

Outro tipo importante de caminhamento em árvores é o caminhamento pós-fixado. Esse algoritmo pode ser entendido como o oposto do caminhamento prefixado, porque primeiro percorre recursivamente as subárvores enraizadas nos filhos da raiz, e depois visita a raiz. É similar ao caminhamento prefixado, entretanto, pois usando-o para resolver um determinado problema, especializa-se a ação associada com a “visitação” de um nó v. Ainda, da mesma forma que o caminhamento prefixado, se a árvore for ordenada, as chamadas recursivas nos filhos de um nó v são feitas de acordo com sua ordem especifica.

* Caminhamento Interfixado (LNR)

 Um método de caminhamento adicional para uma árvore binária é o caminhamento interfixado. Nesse método, visita-se o nó entre os caminhamentos recursivos das subárvores direita e esquerda. O caminhamento interfixado sobre uma árvore binária T pode ser informalmente considerado como a visita aos nós de T “da esquerda para a direita”. De fato, para cada nó v, o caminhamento interfixado visita v após todos os nós da subárvore esquerda de v e antes de visitar todos os nós da subárvore direita de v.

Cada vez que é necessário percorrer uma árvore e executar uma operação (visitar) sobre cada nó, é necessário reescrever

**5. CONCLUSÃO**

Uma árvore binária é uma estrutura de dados mais geral que uma lista encadeada.  Ela e uma estrutura de dado mais formalizada que atende maiores necessidades sendo um conjunto de registro que satisfaz certas condições.  Os registros serão chamados nós*,* células ou nodos s. Cada nó tem um endereço.  Em um jeito simples de se dizer cada nó tem apenas três campos:  um número inteiro e dois ponteiros para nós.  Se tem esse nome por ser uma estrutura de dados útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo, que de forma analógica, vai se desenrolando e formando uma árvore. Tanto a estrutura de árvores binarias quando os tipos de abstrações de dados são estratégias de programação disponível para várias linguagens de programação, sendo útil e otimizando o programa de acordo com a necessidade.

**REFERÊNCIAS**

FEOFILOFF, Paulo. **Árvores binárias.** 2018. Instituto de matemática e estatística universidade de São Paulo. Disponível em: <https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html>. Acesso: 29 de maio de 2021.

GOODRICH, Michael T; TAMASSIA, Roberto. Estruturas de Dados e Algoritmos em Java, 5ª Edição. Porto Alegre: Bookman, 2013.

INSTITUO FEDERAL - SANTA CATARINA, CAMPUS SÃO JOSE **.PRG29003: Introdução a árvores binárias**. 2020 Disponível em: <https://wiki.sj.ifsc.edu.br/index.php/PRG29003:_Introdu%C3%A7%C3%A3o_a_%C3%A1rvores_bin%C3%A1rias>. Acesso: 29 de maio de 2021

NETO, Alberto Costa. **Árvores Binárias**. Universidade Federal de Sergipe – Curso sistemas de informação. 2010. Disponível em: <http://albertocn.sytes.net/2010-2/ed1/aulas/arvores_binarias.htm>. Acesso: 29 de maio de 2021