# Implementação e análise algorítmica das estruturas de dados do tipo árvore

**Marcos P. S. Barreto**[∓]

[1] [∓] Universidade Federal do Piauí(UFPI) - Picos, Piauí

***Resumo.*** *Este trabalho tem como objetivo a resolução de problemas com a utilização das estruturas de dados árvores. Das oito questões propostas foram resolvidas quatro, onde cada umas das questões utilizou-se árvores. As variações de árvores para os problemas resolvidos foram a árvore binária de busca, AVL, rubro negra e 2-3.*

## 1. Introdução

Árvores são um conjunto de elementos que podem estar vazios ou particionado em três ou mais subconjuntos, o primeiro subconjunto contém um único elemento, chamado raiz da árvore. Os outros subconjuntos também são árvores, entretanto, chamadas de subárvores ou filhos. Dependendo do tipo de árvore esses filhos localizam-se a esquerda, direita ou meio do seu nó pai [Tenenbaum et al. 2004]. Cada elemento em uma árvore, independente do seu tipo, é chamado de nó, ou node.

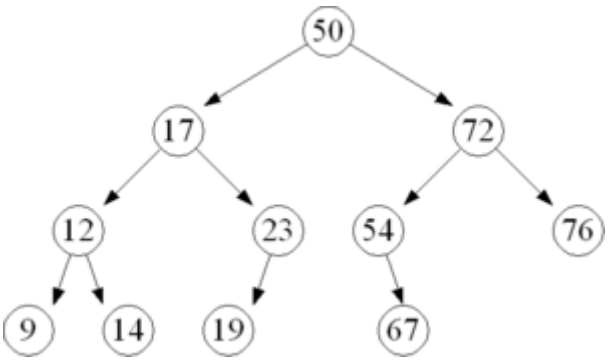


**Figura 1. Exemplo de árvore do tipo binária**

Essas estruturas são comumente utilizados no âmbito da computação devido a sua eficiência e eficacia. Ao lidar com grandes conjuntos de dados a aplicação de árvores nesse contexto resolve a maior parte dos problemas relacionados a busca, inserção e remoção pelo fato de sua estrutura natural otimizar todas essas funcionalidades. Entretanto, dependendo do tipo de árvore adotada, a implementação pode-se tornar complexa, dificultando a sua utilização e consequentemente implicando em um maior esforço para sua concepção. Embora não vejamos explicitamente a aplicação desse tipo de estrutura elas estão presentes em nossos cotidiano de forma encapsulada e abstraída. Alguns exemplos de sua utilização:

- Sistema de arquivos NTFS do Windows;
- Sistema de arquivos HFS do Mac;
- Sistemas de arquivos ReiserFS, XFS, Ext3FS, JFS do Linux;
- Bancos de dados ORACLE, DB2, INGRES, SQL e PostgreSQL.

Neste trabalho será abordado a resolução de problemas através de implementações em linguagem C utilizando-se quatro tipos de árvores:

- Árvore de busca binária;
- Árvore AVL;
- Árvore Rubro-Negra;
- Árvore 2-3.

## 2. Seções Específicas

### 2.1. Problema 1 utilizando árvore binária de busca

O primeiro problema consiste em implementar um programa em C que gere 1000 números aleatórios e os insira em uma árvore binária de busca, Posteriormente deve ser impresso na tela o nível da folha de maior profundidade e o nível da folha de menor profundidade. O processo deve ser repetido 30 vezes, onde no seu termino deve-se mostrar quantas das 30 vezes a diferença entre a profundidade máxima e mínima foram de 0, 1, 2 , 3 e assim por diante. E por fim o programa deve contabilizar os tempos de inserção e busca na árvore.

#### 2.1.1. Árvore binária de busca

Uma árvore binária de busca é uma estrutura de dados baseada em nós, onde todos os nós da subárvore esquerda possuem um valor numérico inferior ao nó raiz e todos os nós da subárvore direita possuem um valor superior ao nó raiz.

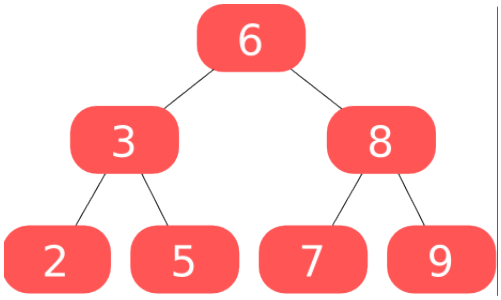


**Figura 2. Árvore binária de busca**

Cada nó da árvore contém informações específicas para seu funcionamento, podemos definir um nó a partir de uma estrutura em C com as seguintes informações:

- Um ponteiro para o filho da direita;
- Um ponteiro para o filho da esquerda;
- Um valor X para definir a localização do nó(esquerda ou direita).

### 2.1.2. Implementação da solução

Recebe como parâmetros dois ponteiros do tipo arvore, o primeiro refere-se a raiz da árvore e o segundo ao novo nó a ser inserido. A lógica desta função consiste em comparar o valor do nó a ser inserido com o nó atual presente na estrutura, caso o novo valor seja menor faz-se uma chamada recursiva para adicionar o no a esquerda caso contrário para a direita. Após inserir é contada as repetições de cada número sorteado na inserção, se não houver repetições desse número o mesmo é adicionado em uma lista de ocorrências. O programa também possui a funcionalidade de calcular a altura e profundidade de um determinado nó. Por fim a função recalcula recursivamente as alturas dos nós afetados. O programa também registra os níveis das folhas de maior e menor profundidade, o tempo para realizar um busca, o tempo para realizar as inserções e a quantidade dos valores referentes as profundidades máximas e mínimas.

## 2.2. Problema 2 utilizando árvore AVL e Rubro-Negra

Este problema consiste em resolver o mesmo problema abordado anteriormente, entretanto, com tipos de árvores diferentes, a AVL e a rubro-negra

### 2.2.1. Árvore AVL

A AVL (Adelson-Velskii e Landis) é uma árvore balanceada, isto é, nas inserções e exclusões, procura-se executar uma rotina de balanceamento tal que as alturas das sub-árvores esquerda e direita tenham alturas bem próximas, com no máximo uma unidade de diferença. Em uma AVL para todo nó temos uma altura *hd*(altura da sub-árvore direita) e uma *he*(altura da sub-árvore esquerda), a diferença entre *hd* e *he* resulta no fator de balanceamento do nó,apresentando os valores 0 , 1 ou -1 se estiver balanceada caso contrário o nó está desbalanceado.

Na figura abaixo podemos visualizar uma árvore completamente balanceada, ou seja, o fator de balanceamento de cada nó é 1 , 0 ou -1.

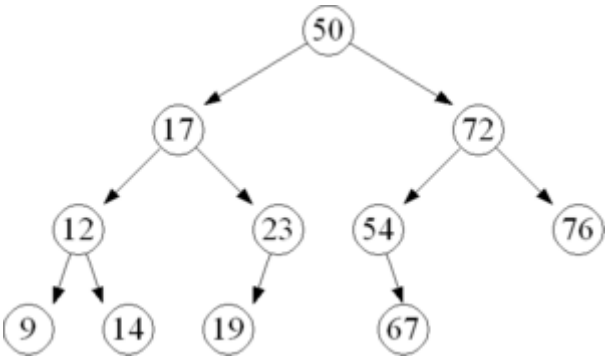


**Figura 3. Árvore balanceada**

### 2.2.2. Balanceamento da árvore AVL

O balanceamento da AVL se dá por meio de rotação, ou seja, o nó desbalanceado é rotacionado, obedecendo um certo padrão. Temos as seguintes rotações:

- *Rotação Simples a Direita* - O nó desbalanceado pende para o lado esquerdo;
- *Rotação Simples a Esquerda* - O nó desbalanceado pende para o lado direito ;
- *Rotação Dupla a Direita* - O nó desbalanceado pende para a sub-árvore esquerda do lado direito. É composta de uma rotação simples à direita, seguida de uma rotação simples à esquerda;
- *Rotação Dupla a Esquerda* - O nó desbalanceado pende para a sub-árvore direita do lado esquerdo. É composta de uma rotação simples à esquerda, seguida de uma rotação simples à direita;.

### 2.2.3. Implementação da solução

Ao inserir um novo valor na árvore uma função calcula a altura dos nós afim de verificar se houve desbalanceamento, se sim utiliza-se as técnicas de rotação simples a esquerda(*LL*), rotação simples a direita(*RR*) ou as duas em conjunto para rotações duplas. Antes de realizar a rotação o programa analisa o padrão de desbalanceamento para decidir qual rotação aplicar. Por fim a função recalcula recursivamente as alturas dos nós afetados. O programa também registra os níveis das folhas de maior e menor profundidade, o tempo para realizar um busca, o tempo para realizar as inserções e a quantidade dos valores referentes as profundidades máximas e mínimas.

## 2.3. Árvore Rubro-Negra

Uma árvore rubro-negra é uma árvore binária de busca onde cada nó possui um atributo de cor, vermelho ou preto. Cada nó nesse tipo de árvore possui os seguintes atributos:

- *Cor* - podendo ser vermelha ou preta, é utilizada como critério de balanceamento;
- *Valor* - Inteiro a ser inserido no nó;
- *Ponteiro para esquerda* - Filho da esquerda;
- *Ponteiro para direita* - Filho da direita;

Em relação as árvores binárias, a rubro-negra apresenta algumas condições a mais para sua implementação:

- Um nó é vermelho ou preto;
- A raiz sempre é preta;
- Todo nó folha (NULL) é preto;
- Se um nó é vermelho, então os seus filhos são pretos;
- Não existem nós vermelhos consecutivos;
- Para cada nó, todos os caminhos desse nó para os nós folhas descendentes contém o mesmo número de nós pretos.

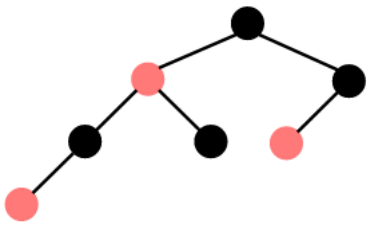

**Figura 4. Rubro-Negra**

### 2.3.1. Implementação da solução

Ao ser inserido um novo valor na árvore o programa verifica em qual sub-árvore o valor será inserido e se as condições da rubro-negra foram preservadas após a inserção. Depois da inserção o programa realiza uma analise das cores dos nós e seus filhos, para verificar a integridade da rubro-negra. Dependendo do resultado dessa analise o programa rotaciona o nó para direita, esquerda ou simplesmente troca as cores dele e seus filhos. Por fim a função recalcula recursivamente as alturas dos nós afetados. O programa também registra os níveis das folhas de maior e menor profundidade, o tempo para realizar um busca, o tempo para realizar as inserções e a quantidade dos valores referentes as profundidades máximas e mínimas.

## 2.4. Problema 3 utilizando árvore binária de busca

O problema consiste na implementação de um algoritmo que converta um dicionario inglês/português em português/inglês. O resultado dessa conversão deve ser armazenado em uma árvore de busca binária onde cada nó contém uma palavra em português.

### 2.4.1. Implementação da solução

Ao ler cada linha do arquivo de texto contendo o dicionário inglês/português o programa separa as palavras em português e inglês por seus delimitadores. Para cada palavra em português é criado um novo nó na árvore levando em conta a ordem alfabética, nas estrutura de cada nó há uma lista com as equivalentes em inglês da palavra inserido. Um critério importante é não repetir as palavras em português, caso haja mais de um significado no arquivo em inglês, deve-se adicionar na lista de equivalências.

## 2.5. Problema 4 utilizando árvore 2-3

O problema 4 trata-se de implementar a mesma solução do problema 3, porém utilizando uma árvore 2-3.

## 2.6. Árvore 2-3

Árvores 2-3 são naturalmente balanceadas onde os seus nós podem possuir ate três filhos, cada um com no máximo duas informações. Os atributos do nó de uma árvore 2-3 são basicamente os seguintes:

- Número de informações do nó;
- Informação um;
- Informação dois;
- Ponteiro para o nó da esquerda;
- Ponteiro para o nó do meio;
- Ponteiro para o nó da direita.

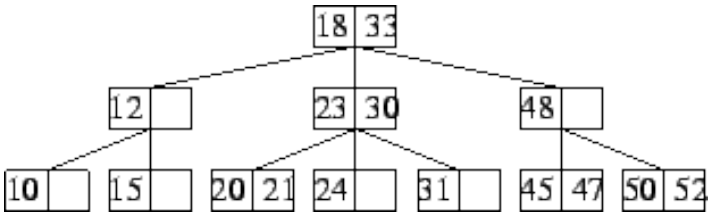


**Figura 5. Árvore 2-3**

### 2.6.1. Implementação da solução

As funções de tratamento de entrada e leitura do dicionário inglês/português é a mesma da implementação 2.4.1, a diferença se dá na inserção de uma nova palavra, onde cada nó pode possuir até duas palavras em português e duas listas de equivalentes em inglês. Ao ser inserido um novo nó o programa, através de comparações, verifica em qual sub-árvore a palavra será inserida, a esquerda, meio ou direita. Antes de inserir o programa verifica se há espaço no nó selecionado, se houver a palavra é inserida normalmente caso contrário uma função ”quebra”o nó que inicialmente estava cheio, cria dois novos nós e reestabelece as ligações. O nó central que foi quebrado deve ser inserido em seu nó pai se tiver espaço, caso contrário a função quebrar nó é chamada recursivamente.

## 3. Resultados

A tabela dos resultados foi gerada a partir da média dos tempos de inserção e busca nas árvores binária, AVL e rubro-negra. O processo de inserção de 1000, 10000 e 100000 elementos foi realizado 30 vezes em cada uma das árvores, além da inserção também foi feita, nas mesmas condições, uma busca fixa pelo valor 50. A máquina utilizada para a realização dos testes era relativamente simples, contando apenas com um processador dual core de ate 1.8Ghz e 4GB de memória RAM.

| Análise de desempenho | | | |
|---|---|---|---|
| Árvore | Elementos | Inserção(ms) | Busca(ms) |
| Árvore Binária | 1.000 | 0.0 | 0.0 |
| Árvore Binária | 10.000 | 3.0 | 0.0 |
| Árvore Binária | 20.000 | 7.0 | 0.0 |
| Árvore AVL | 1.000 | 18.06 | 0.0 |
| Árvore AVL | 10.000 | 2021.43 | 0.0 |
| Árvore AVL | 20.000 | 6284.43 | 0.0 |
| Árvore Rubro Negra | 1.000 | 0.0 | 0.0 |
| Árvore Rubro Negra | 10.000 | 139.06 | 0.0 |
| Árvore Rubro Negra | 20.000 | 514.79 | 0.0 |

## 4. Conclusão

O entendimento a respeito do funcionamento das estruturas de dados árvores é extremamente importante e inerente ao ramo da computação. Compreender a lógica e implementação dessas estruturas permite uma maior habilidade ao programador de resolver problemas complexos de forma eficiente e elegante. Além disso, várias soluções computacionais são implementados utilizando-se árvores. A realização deste trabalho permitiu um maior entendimento a sobre a implementação e regras utilizadas em cada tipo de árvore, além disso, por meio das análises técnicas dos algoritmos observou-se o desempenho dos estruturas em casos de testes diferentes, obtendo como resultado a a árvore mais eficiente e eficaz.

## Referências

Tenenbaum, A. M., Langsam, Y., and Augenstein, M. J. (2004). *Estruturas de dados usando C*. Pearson Makron Books.