# Análise de Dados Ambulatoriais do DATASUS

**UNIVERSIDADE PRESBITERIANA MACKENZIE**
**ESTRUTURA DE DADOS 2**
**Cristiane Camilo Hernandez**

Bruno Rabelo Torchio de Oliveira RA:10239373
Caio Teixeira Torres RA:10417478
Guilherme Dias Ferreira Pereira RA:10417684
João Victor de Paula Silva RA:10418556

## Sumário:

# Introdução:

Este projeto foi desenvolvido como parte das atividades da disciplina de Estruturas de Dados 2, com o objetivo de aplicar os conceitos aprendidos durante o semestre de maneira prática e extensionista. A proposta visa beneficiar a sociedade ao alinhar-se com os Objetivos de Desenvolvimento Sustentável (ODS) da ONU, especificamente o **ODS 3: Saúde e Bem-Estar**.

O projeto utilizará **Estruturas de Dados** como **Árvore Binária de Busca (ABB)** e **Árvore Binária de Busca Balanceada (AVL)** para filtrar e analisar registros ambulatoriais obtidos do DATASUS, com foco nos dados do estado de São Paulo. A comparação entre a eficiência das duas estruturas será feita por meio de testes de desempenho, gráficos e outras formas de análise.

A relevância deste projeto está em sua contribuição para a **conscientização sobre a importância da saúde pública**, além de proporcionar uma forma mais eficiente de acesso e análise dos dados relacionados a atendimentos médicos no Sistema Único de Saúde (SUS). A organização e análise dos dados possibilita uma maior facilidade no acesso à informação, colaborando para uma gestão de saúde mais eficaz e transparente.

# Escolha do dataset

Escolhemos um **dataset** disponível na plataforma **DATASUS**, que contém informações detalhadas sobre a produção ambulatorial do Sistema Único de Saúde (SUS) a partir de 2008. Este conjunto de dados inclui resultados de exames realizados nos serviços de saúde pública, como **colesterol**, **hemoglobina** e **triglicerídeos**, oferecendo uma visão abrangente sobre a saúde da população atendida.

Durante a pesquisa inicial, encontramos algumas dificuldades para localizar um dataset adequado. No entanto, após uma busca mais aprofundada, decidimos optar por um **arquivo CSV** referente à produção ambulatorial do estado de **São Paulo**, que estava mais facilmente acessível e alinhado com os objetivos do projeto.

O **dataset** selecionado está diretamente alinhado com o **ODS 3: Saúde e Bem-Estar**, que visa garantir uma vida saudável e promover o bem-estar para todos, em todas as idades. A análise dos exames e resultados contidos neste dataset permitirá a identificação de padrões relacionados a doenças comuns, como doenças cardiovasculares e diabetes, ao monitorar fatores como níveis de colesterol e triglicerídeos. Isso contribuirá para uma melhor compreensão do estado de saúde da população e possibilitará a formulação de políticas públicas mais eficazes, baseadas em dados concretos.

Além disso, ao analisar a produção ambulatorial no SUS, podemos contribuir para a **redução das desigualdades no acesso à saúde**, outro dos objetivos do **ODS 10**. A utilização dessa base de dados para filtrar e organizar informações pode ajudar na identificação de áreas ou grupos com maior necessidade de atenção, melhorando o direcionamento dos recursos e serviços de saúde.

BRASIL. Ministério da Saúde. **TabNet Win32 3.0: Produção Ambulatorial SUS - Estado de São Paulo (a partir de Jan/2008)**. Disponível em:TabNet Win32 3.0: Produção Ambulatorial SUS - Estado de São Paulo - (A partir de Jan/2008) Acesso em: 25 nov. 2024

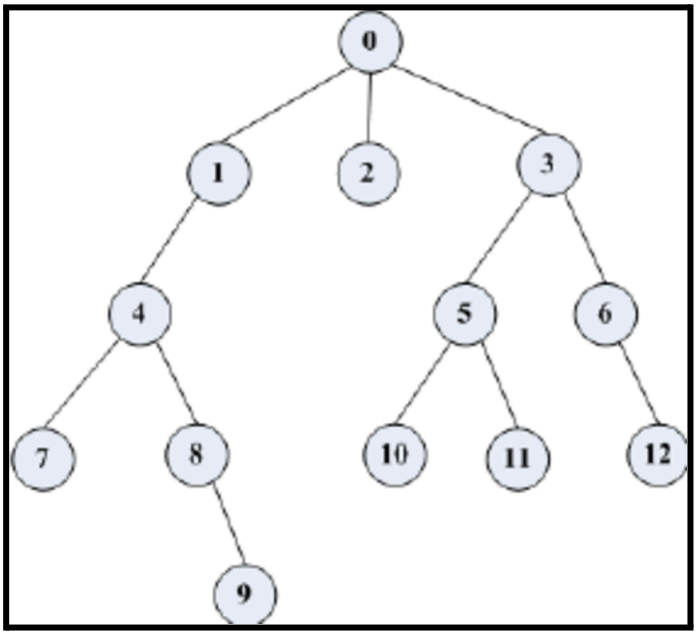
# Modelagem dos Dados em Árvores

**Árvore Binária de Busca(ABB):**

Cada nó na árvore armazena um valor proveniente do CSV,o nome da linha que o dado pertence e o próprio dado, organizando-se de acordo com as regras da ABB. Os nós são dispostos da seguinte maneira:

- Os valores **menores** que o nó raiz ou nó pai são inseridos na **subárvore esquerda**.
- Os valores **maiores** são inseridos na **subárvore direita**.

É importante destacar que a ABB **não realiza balanceamento automático**. Veja a seguir:

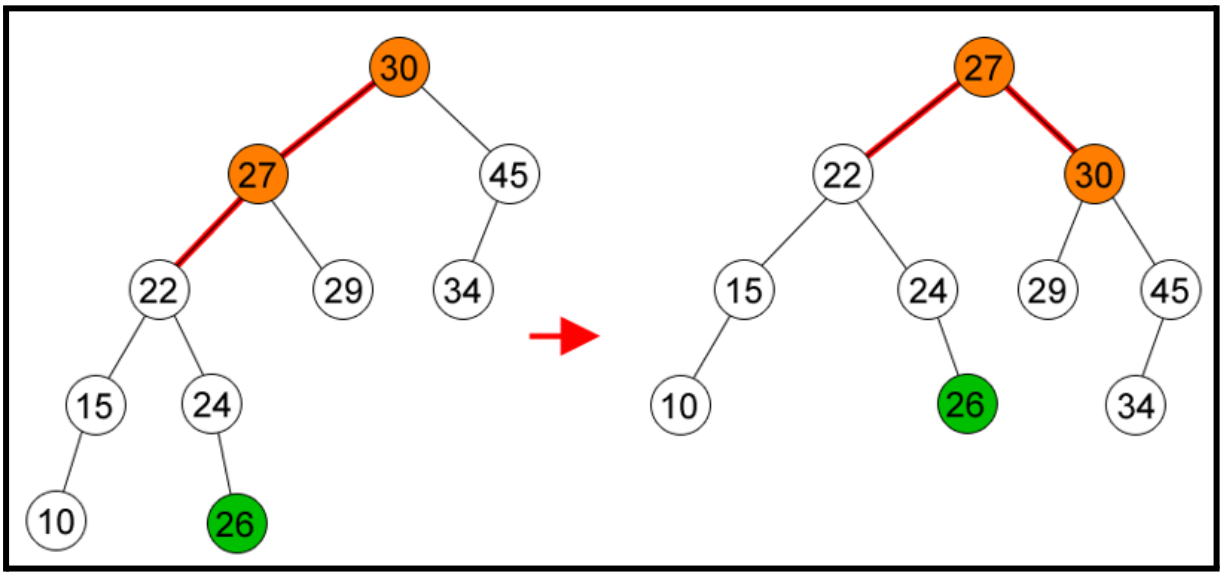



**Árvore Binária de Busca Balanceada(AVL):**

Assim como na ABB, cada nó na AVL armazena um valor proveniente do CSV, organizando-se com base nas mesmas regras:

- Os valores **menores** que o nó raiz ou nó pai são inseridos na **subárvore esquerda**.
- Os valores **maiores** são inseridos na **subárvore direita**.

No entanto, a AVL se diferencia por **garantir o balanceamento automático da árvore**. Após cada inserção ou remoção, o fator de balanceamento (diferença entre as alturas das subárvores esquerda e direita) é avaliado. Se a diferença ultrapassar o limite de ±1, a AVL realiza **rotações simples ou duplas** para corrigir o desbalanceamento. Veja a seguir:




## Implementação das Operações sobre a Árvore

**Árvore Binária de Busca(ABB):**
Nesta árvore fizemos operações de **inserção**, **remoção** e **operações para percorrer a árvore**, que serão explicadas a seguir:

- **Inserção(**inserir(E valor)**):**

Compara o valor a ser inserido com o valor do nó atual. Se o valor for menor, a inserção ocorre na subárvore esquerda, caso contrário, na subárvore direita. A operação recursiva segue até encontrar um nó nulo para a inserção do novo valor. Além disso, adicionamos variáveis para contar o tempo de execução para podermos analisar.

```
public void inserir(E valor) {
```

```
        long inicio = System.nanoTime(); // Início da medição
de tempo
        raiz = inserir(valor, raiz);
        long fim = System.nanoTime(); // Fim da medição de
tempo
        comparacaoArvores.adicionarTempoExecucaoABB(fim -
inicio);
    }

    private Node<E> inserir(E valor, Node<E> node) {
        comparacaoArvores.incrementarOperacoesABB(); //
Incrementa operações
        if (node == null) {
            return new Node<>(valor); // Cria um novo nó se a
posição estiver vazia
        }
        if (valor.compareTo(node.getValue()) < 0) {
            node.setFilhoEsquerdo(inserir(valor,
node.getFilhoEsquerdo())); // Insere à esquerda
        } else {
            node.setFilhoDireito(inserir(valor,
node.getFilhoDireito())); // Insere à direita
        }
        return node; // Retorna o nó (para reconstruir a
árvore)
    }
```

- **Remoção (**eliminar(E valor)**)**:

A remoção de um nó é feita da seguinte forma: se o nó a ser removido tiver dois filhos, ele será substituído pelo maior valor de sua subárvore esquerda. Caso tenha um filho ou nenhum, o nó será simplesmente removido e substituído pelo filho (caso haja).Além disso, adicionamos variáveis para contar o tempo de execução para podermos analisar.

```
    private Node<E> eliminar(E valor, Node<E> node) {
        if (node == null) {
            return null;
        }
        comparacaoArvores.incrementarOperacoesABB();
        if (valor.compareTo(node.getValue()) < 0) {
            node.setFilhoEsquerdo(eliminar(valor,
node.getFilhoEsquerdo()));
        } else if (valor.compareTo(node.getValue()) > 0) {
```

```
            node.setFilhoDireito(eliminar(valor,
node.getFilhoDireito()));
        } else {
            if (node.getFilhoEsquerdo() == null) {
                return node.getFilhoDireito();
            } else if (node.getFilhoDireito() == null) {
                return node.getFilhoEsquerdo();
            } else {
                Node<E> maior =
getMax(node.getFilhoEsquerdo());
                node.setValue(maior.getValue());

node.setFilhoEsquerdo(eliminar(maior.getValue(),
node.getFilhoEsquerdo()));
            }
        }
        return node;
    }
```

- **Operações para percorrer a árvore**(emNivel()), (emOrdem()), (posOrdem()), (preOrdem()):

Essas são estratégias que utilizam de busca em largura e/ou recursividade para percorrer toda a árvore e exibir os nós, cada uma de sua respectiva maneira, porém a método mais comum e utilizado é o emOrdem que mostrará todos os nós em sua respectiva ordem, como o nome já sugere.Além disso, adicionamos variáveis para contar o tempo de execução para podermos analisar.

```
   public void emOrdem() {
        long inicio = System.nanoTime();
        emOrdem(raiz);
        long fim = System.nanoTime();
        comparacaoArvores.adicionarTempoExecucaoABB(fim -
inicio);
    }

    private void emOrdem(Node<E> node) {
        if (node != null) {
            emOrdem(node.getFilhoEsquerdo());
            System.out.print(node.getValue() + " ");
            comparacaoArvores.incrementarOperacoesABB();
            emOrdem(node.getFilhoDireito());
        }
    }
```

```
    // Percorre a árvore em pré-ordem
    public void preOrdem() {
        long inicio = System.nanoTime();
        preOrdem(raiz);
        long fim = System.nanoTime();
        comparacaoArvores.adicionarTempoExecucaoABB(fim -
inicio);
    }

    private void preOrdem(Node<E> node) {
        if (node != null) {
            System.out.print(node.getValue() + " ");
            comparacaoArvores.incrementarOperacoesABB();
            preOrdem(node.getFilhoEsquerdo());
            preOrdem(node.getFilhoDireito());
        }
    }

    // Percorre a árvore em pós-ordem
    public void posOrdem() {
        long inicio = System.nanoTime();
        posOrdem(raiz);
        long fim = System.nanoTime();
        comparacaoArvores.adicionarTempoExecucaoABB(fim -
inicio);
    }

    private void posOrdem(Node<E> node) {
        if (node != null) {
            posOrdem(node.getFilhoEsquerdo());
            posOrdem(node.getFilhoDireito());
            System.out.print(node.getValue() + " ");
            comparacaoArvores.incrementarOperacoesABB();
        }
    }

    // Percorre a árvore em nível
    public void emNivel() {
        long inicio = System.nanoTime();
        LinkedList<Node<E>> fila = new LinkedList<>();
        fila.add(raiz);
```

```
        while (!fila.isEmpty()) {
            Node<E> atual = fila.removeFirst();
            if (atual != null) {
                comparacaoArvores.incrementarOperacoesABB();
                System.out.print(atual.getValue() + " ");
                fila.add(atual.getFilhoEsquerdo());
                fila.add(atual.getFilhoDireito());
            }
        }
        long fim = System.nanoTime();
        comparacaoArvores.adicionarTempoExecucaoABB(fim -
inicio);
    }
```

- **Operações para pegar o nó específicos**(getMaiorValorABB()), getElemento(E valor)), buscarElemento(E valor, Node<E> node)):

Esses métodos percorrem a árvore para achar um nó em específico, os dois relacionados a elemento completam a execução do maior valor

```
    public Node<E> getMaiorValorABB() {
        if (raiz == null) {
            return null; // Retorna null se a árvore estiver
vazia
        }

        Node<E> atual = raiz;
        while (atual.getFilhoDireito() != null) {
            atual = atual.getFilhoDireito();
            incrementarOperacoes(); // Incrementa operações
para estatísticas
        }

        return atual; // Retorna o maior nó
    }
    public Node<E> getElemento(E valor) {
        return buscarElemento(valor, raiz);
    }

    private Node<E> buscarElemento(E valor, Node<E> node) {
        if (node == null) {
            return null; // Retorna null se o nó não for
```

```
encontrado
        }

        if (valor.compareTo(node.getValue()) < 0) {
            return buscarElemento(valor,
node.getFilhoEsquerdo()); // Busca à esquerda
        } else if (valor.compareTo(node.getValue()) > 0) {
            return buscarElemento(valor,
node.getFilhoDireito()); // Busca à direita
        } else {
            return node; // Retorna o nó se o valor for
encontrado
        }
    }
```

**Árvore Binária de Busca(AVL):**

A árvore AVL é uma variação da árvore binária de busca (ABB), com o diferencial de manter um balanceamento entre as subárvores. Para isso, a árvore utiliza rotações sempre que a diferença de altura entre as subárvores de um nó for maior que 1 ou menor que -1. Vamos explicar as operações da árvore AVL, que incluem inserção, remoção, busca e balanceamento:

- **Inserção (inserir(E valor)):**

A inserção na árvore AVL é similar à inserção na árvore binária de busca, mas após a inserção do nó, é necessário realizar o balanceamento da árvore para garantir que ela continue balanceada. O balanceamento é feito através de rotações, que serão explicadas mais à frente.

```
public void inserir(E valor) {
    long inicio = System.nanoTime(); // Início da medição de
tempo
    raiz = inserir(valor, raiz);
    long fim = System.nanoTime(); // Fim da medição de tempo
    tempoExecucao += (fim - inicio); // Adiciona o tempo de
execução
}

private Node<E> inserir(E valor, Node<E> node) {
    contadorInsercao++; // Incrementa a contagem de inserção
    if (node == null) {
```

```
            return new Node<>(valor); // Cria um novo nó
        }
        if (valor.compareTo(node.getValue()) < 0) {
            node.setFilhoEsquerdo(inserir(valor,
node.getFilhoEsquerdo())); // Insere à esquerda
        } else if (valor.compareTo(node.getValue()) > 0) {
            node.setFilhoDireito(inserir(valor,
node.getFilhoDireito())); // Insere à direita
        }

        // Atualiza a altura do nó após a inserção
        node.setAltura(1 +
Math.max(getAltura(node.getFilhoEsquerdo()),
getAltura(node.getFilhoDireito())));

        // Realiza o balanceamento da árvore
        return balancear(node);
    }
```

- **Balanceamento:**

Após cada inserção, a árvore AVL pode precisar ser balanceada. Isso é feito com base no fator de balanceamento, que é a diferença de altura entre a subárvore esquerda e a subárvore direita de um nó. Se o fator de balanceamento for maior que 1 ou menor que -1, o nó será balanceado com rotações.

```
    private Node<E> balancear(Node<E> node) {
        int balanceamento = getFatorBalanceamento(node);

        if (balanceamento > 1 &&
getFatorBalanceamento(node.getFilhoEsquerdo()) >= 0) {
            return rotacaoDireita(node); // Caso de rotação à
direita
        }

        if (balanceamento < -1 &&
getFatorBalanceamento(node.getFilhoDireito()) <= 0) {
            return rotacaoEsquerda(node); // Caso de rotação à
esquerda
        }

        if (balanceamento > 1 &&
getFatorBalanceamento(node.getFilhoEsquerdo()) < 0) {
```

```
node.setFilhoEsquerdo(rotacaoEsquerda(node.getFilhoEsquerdo()
));
            return rotacaoDireita(node); // Caso de rotação
dupla: esquerda-direita
        }

        if (balanceamento < -1 &&
getFatorBalanceamento(node.getFilhoDireito()) > 0) {

node.setFilhoDireito(rotacaoDireita(node.getFilhoDireito()));
            return rotacaoEsquerda(node); // Caso de rotação
dupla: direita-esquerda
        }

        return node; // Caso a árvore já esteja balanceada
    }
```

- **Rotações:**

As rotações são operações fundamentais para o balanceamento da árvore AVL. Elas garantem que a árvore permaneça balanceada após inserções e remoções.

```
    // Rotações
    private Node<E> rotacaoDireita(Node<E> y) {
        Node<E> x = y.getFilhoEsquerdo();
        Node<E> T2 = x.getFilhoDireito();

        x.setFilhoDireito(y);
        y.setFilhoEsquerdo(T2);

        // Atualiza as alturas dos nós
        y.setAltura(Math.max(getAltura(y.getFilhoEsquerdo()),
getAltura(y.getFilhoDireito())) + 1);
        x.setAltura(Math.max(getAltura(x.getFilhoEsquerdo()),
getAltura(x.getFilhoDireito())) + 1);

        return x; // Retorna a nova raiz
    }

    private Node<E> rotacaoEsquerda(Node<E> x) {
        Node<E> y = x.getFilhoDireito();
        Node<E> T2 = y.getFilhoEsquerdo();

        y.setFilhoEsquerdo(x);
```

```
    x.setFilhoDireito(T2);

    // Atualiza as alturas dos nós
    x.setAltura(Math.max(getAltura(x.getFilhoEsquerdo()),
getAltura(x.getFilhoDireito())) + 1);
    y.setAltura(Math.max(getAltura(y.getFilhoEsquerdo()),
getAltura(y.getFilhoDireito())) + 1);

    return y; // Retorna a nova raiz
}
```

- **Remoção (eliminar(E valor)):**

A remoção em uma árvore AVL é semelhante à de uma árvore binária de busca. Após a remoção de um nó, é necessário verificar o balanceamento da árvore e realizar as rotações se necessário.

```
public boolean eliminar(E valor) {
    if (isEmpty()) return false;
    raiz = eliminar(valor, raiz);
    return true;
}

private Node<E> eliminar(E valor, Node<E> node) {
    contadorRemocao++; // Incrementa a contagem de remoção
    if (node == null) {
        return null;
    }
    if (valor.compareTo(node.getValue()) < 0) {
        node.setFilhoEsquerdo(eliminar(valor,
node.getFilhoEsquerdo()));
    } else if (valor.compareTo(node.getValue()) > 0) {
        node.setFilhoDireito(eliminar(valor,
node.getFilhoDireito()));
    } else {
        // Caso o nó tenha 0 ou 1 filho, é removido
diretamente
        if (node.getFilhoEsquerdo() == null) {
            return node.getFilhoDireito();
        } else if (node.getFilhoDireito() == null) {
            return node.getFilhoEsquerdo();
        } else {
            // Caso o nó tenha dois filhos, encontra o menor
nó à direita
```

```
            Node<E> maior = getMin(node.getFilhoDireito());
            node.setValue(maior.getValue());
            node.setFilhoDireito(eliminar(maior.getValue(),
node.getFilhoDireito()));
        }
    }
    // Atualiza a altura e balanceia a árvore após remoção
    node.setAltura(1 +
Math.max(getAltura(node.getFilhoEsquerdo()),
getAltura(node.getFilhoDireito())));
    return balancear(node);
}
```

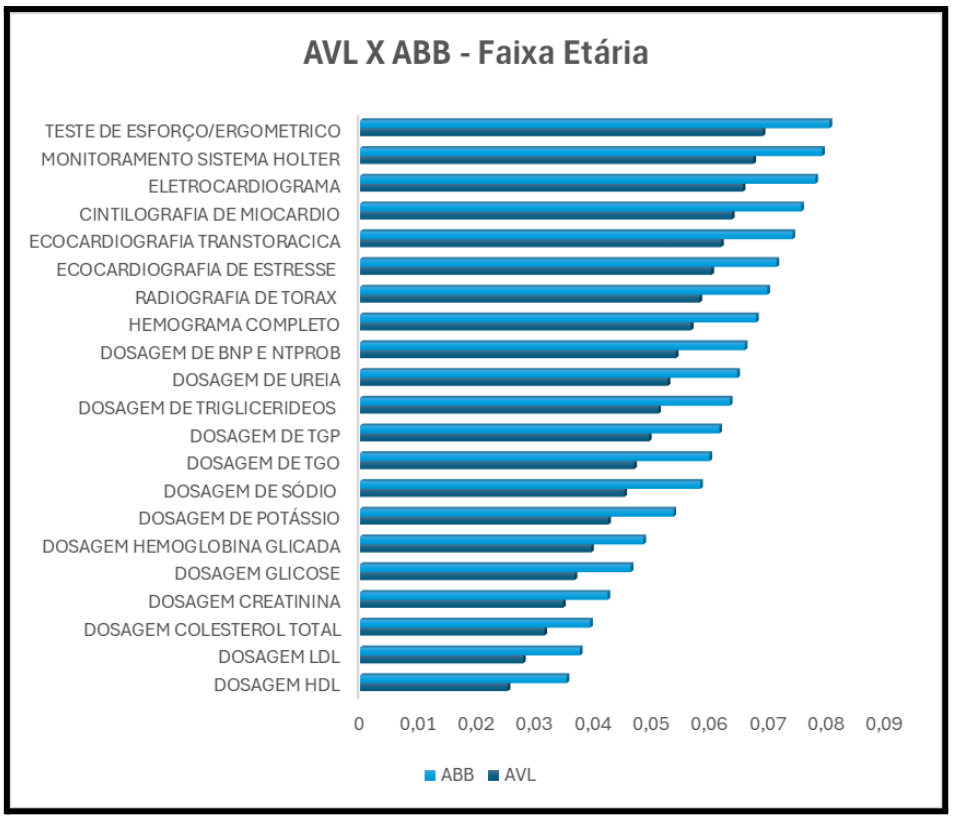
- **Busca (buscar(E valor)):**

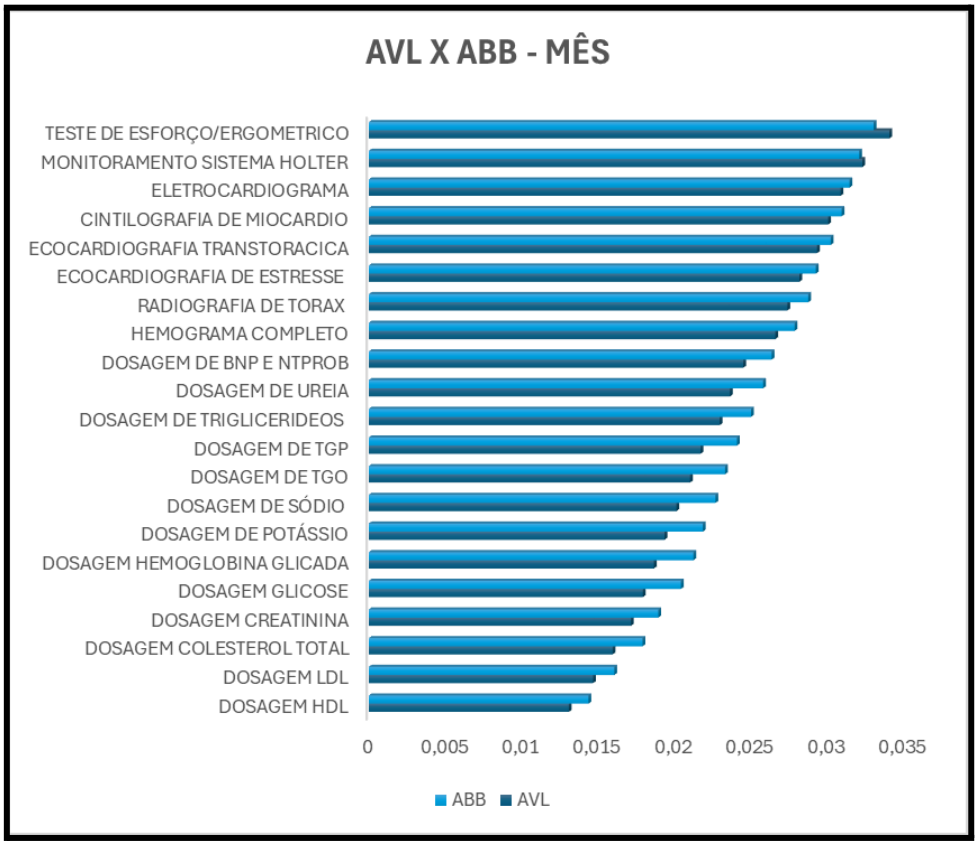
A busca na árvore AVL segue o mesmo processo de uma árvore binária de busca, mas é realizada de forma eficiente devido ao balanceamento da árvore.

```
public boolean buscar(E valor) {
    return buscar(raiz, valor);
}

private boolean buscar(Node<E> node, E valor) {
    contadorBusca++;
    if (node == null) {
        return false;
    }
    if (valor.compareTo(node.getValue()) == 0) {
        return true;
    } else if (valor.compareTo(node.getValue()) < 0) {
        return buscar(node.getFilhoEsquerdo(), valor);
    } else {
        return buscar(node.getFilhoDireito(), valor);
    }
}
```

# Comparação de Desempenho entre ABB e AVL




Conforme observado nos gráficos apresentados, a AVL manteve desempenho superior durante todo o teste, tanto em relação à faixa etária quanto na maior parte do teste mensal.

Esse resultado deve-se à eficiência do balanceamento rigoroso da AVL, que oferece melhor desempenho ao lidar com grandes volumes de dados desordenados.

Embora as diferenças entre AVL e ABB possam parecer pequenas no início, os dados da planilha mostram que essas discrepâncias aumentam significativamente conforme o volume de dados cresce, evidenciando a vantagem da AVL em cenários mais complexos e volumosos. Sendo assim a AVL se mostra muito superior em relação a lidar com grandes quantidades de dados.

# Testes e Resultados

É importante mencionar que no início de ambos os códigos o caminho para o csv está sendo especificado, porém pode haver algum erro em achar o caminho devido a diferenças de nome de pastas e a compactação pro arquivo zip, por isso, caso haja um erro em achar o arquivo CSV, vá na barra lateral esquerda, clique com o botão direito no CSV a ser analisado, copie o caminho e insira na linha especificada na imagem abaixo.

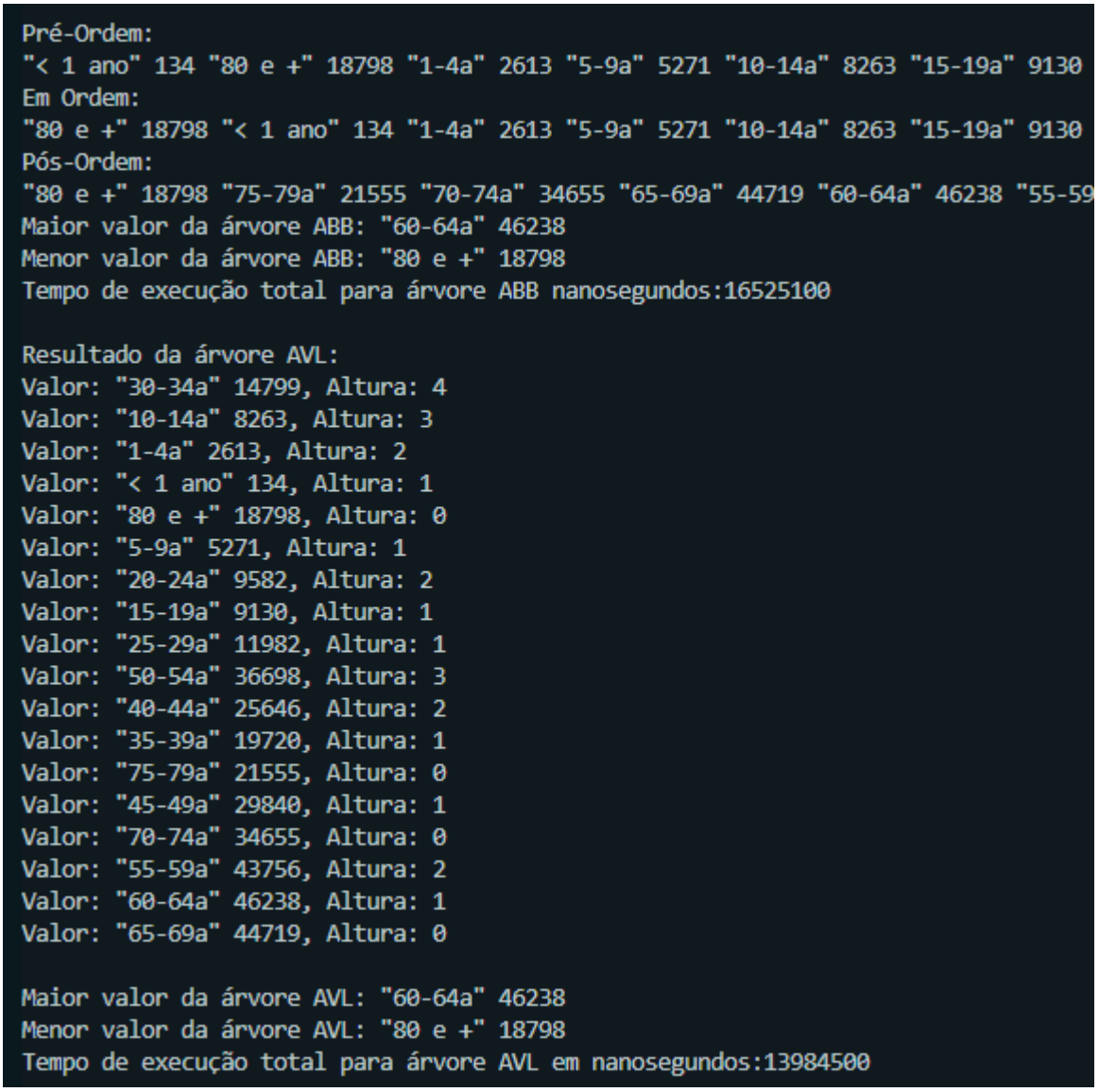
```
String arquivoCSV = "..\\Projeto ED 2 2 versão 7º FINAL MESMO\\DadosMes.csv"; // Nome do arquivo CSV
System.out.println("Analisando o arquivo: " + arquivoCSV);
```

Na primeira seção do código, onde analisamos e respondemos as perguntas referente a faixas etárias não existe nenhuma entrada pelo usuário, como queríamos fazer uma análise direta de todas as informações optamos por pegar automaticamente todos os dados e agrupar cada árvore por exame feito, obtendo assim os dados de todas as faixas etárias e pegando os grupos com maior e menor número de exames feitos

```
Analisando o arquivo: Projeto ED 2 2 versão 2\Projeto ED 2 2\DadosIdade.csv
Colunas encontradas no arquivo:
1: "0202010279 DOSAGEM DE COLESTEROL HDL"
2: "0202010287 DOSAGEM DE COLESTEROL LDL"
3: "0202010295 DOSAGEM DE COLESTEROL TOTAL"
4: "0202010317 DOSAGEM DE CREATININA"
5: "0202010473 DOSAGEM DE GLICOSE"
6: "0202010503 DOSAGEM DE HEMOGLOBINA GLICOSILADA"
7: "0202010600 DOSAGEM DE POTASSIO"
8: "0202010635 DOSAGEM DE SODIO"
9: "0202010643 DOSAGEM DE TRANSAMINASE GLUTAMICO-OXALACETICA (TGO)"
10: "0202010651 DOSAGEM DE TRANSAMINASE GLUTAMICO-PIRUVICA (TGP)"
11: "0202010678 DOSAGEM DE TRIGLICERIDEOS"
12: "0202010694 DOSAGEM DE UREIA"
13: "0202010791 DOSAGEM DE PEPT?DEOS NATRIUR?TICOS TIPO B (BNP E NT-PROBNP)"
14: "0202020380 HEMOGRAMA COMPLETO"
15: "0204030153 RADIOGRAFIA DE TORAX (PA E PERFIL)"
16: "0205010016 ECOCARDIOGRAFIA DE ESTRESSE"
17: "0205010032 ECOCARDIOGRAFIA TRANSTORACICA"
18: "0208010033 CINTILOGRAFIA DE MIOCARDIO P/ AVALIACAO DA PERFUSAO EM SITUACAO DE REPOUSO (MINIMO 3 PROJECOES)"
19: "0211020036 ELETROCARDIOGRAMA"
20: "0211020044 MONITORAMENTO PELO SISTEMA HOLTER 24 HS (3 CANAIS)"
21: "0211020060 TESTE DE ESFORCO / TESTE ERGOMETRICO"

Analisando a coluna: "0202010279 DOSAGEM DE COLESTEROL HDL"
```

Como podemos ver, o programa inicia mostrando os dados contidos no CSV e já parte para analisar coluna a coluna




As linhas e colunas contêm os dados dos exames e a faixa etária analisada, respectivamente, o programa lê cada linha e verificar em qual coluna está, para então salvar todos os dados da mesma coluna em uma ABB e uma AVL, daí então ele imprime as informações referentes aquele exame e as compara no tempo de execução, além de testar se ambas estão corretamente implementadas, pegando o maior e o menor número de exames.(Dados deste print referentes a coluna “Dosagem de Colesterol HDL”)

Já o nosso código para pegar os exames agrupados por Mês/Ano recebe uma entrada pedindo um ano para ser analisado, a escolha de fazer isso se dá pela quantidade maior de dados contidos nesse CSV. A análise fica mais simples e mais direta quando pegamos ano a ano, agrupar os dados fica mais fácil e ainda podemos contar quantos exames foram realizados no ano.

```
Analisando o arquivo: Projeto ED 2 2 versão 2\Projeto ED 2 2\DadosMes.csv
Digite o ano que deseja analisar: 2023

Analisando o procedimento: 0202010279 DOSAGEM DE COLESTEROL HDL

Resultado da árvore ABB:
```

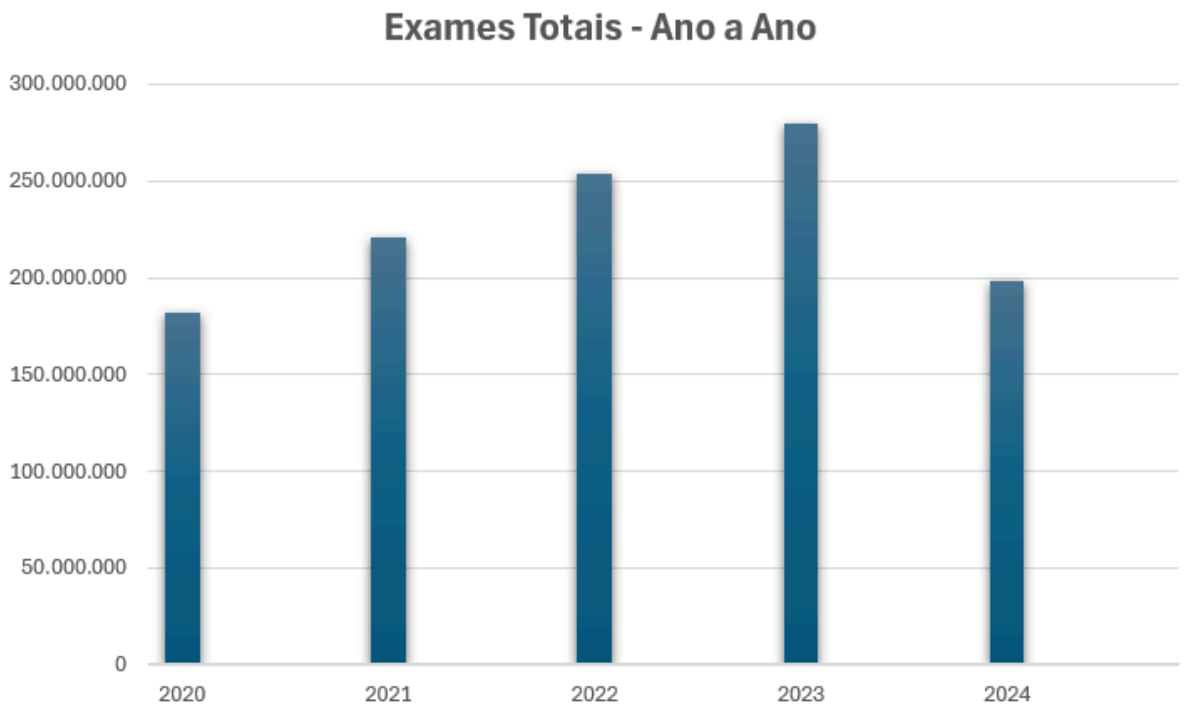
Como usamos dois CSVs diferentes, este segundo está separado de forma diferente, a coluna agora representa o mês e ano analisado, enquanto a primeira instância da linha é o exame sendo realizado, então diferentemente do primeiro código, ele lê a linha do exame e pega todos os dados referentes às colunas do ano sendo analisado e as salva nas árvores, para então comparar elas e imprimir as informações salvar na tela, no mesmo padrão já mostrado.

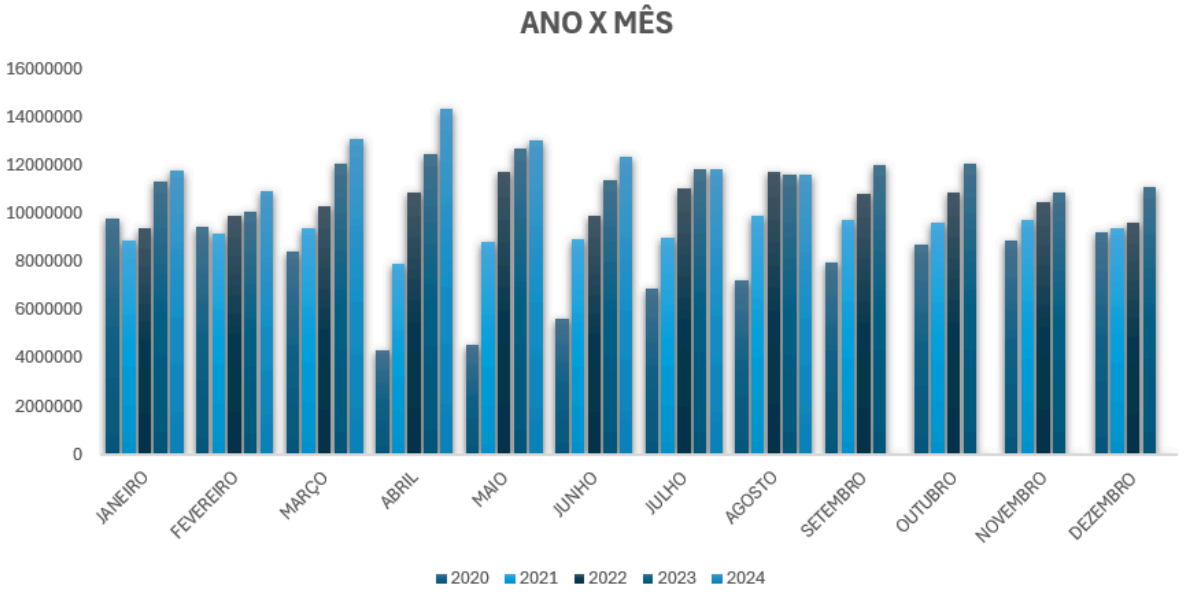
```
Pré-Ordem:
Jan/2023 671078 Fev/2023 566108 Jun/2023 657020 Nov/2023 616529 Dez/20
Em Ordem:
Fev/2023 566108 Nov/2023 616529 Dez/2023 639367 Jun/2023 657020 Ago/20
Pós-Ordem:
Dez/2023 639367 Nov/2023 616529 Ago/2023 665469 Jun/2023 657020 Fev/20
Resultado da árvore AVL:
Valor: Jan/2023 671078, Altura: 3
Valor: Nov/2023 616529, Altura: 2
Valor: Fev/2023 566108, Altura: 1
Valor: Jun/2023 657020, Altura: 1
Valor: Dez/2023 639367, Altura: 0
Valor: Ago/2023 665469, Altura: 0
Valor: Mar/2023 708697, Altura: 2
Valor: Abr/2023 698784, Altura: 1
Valor: Out/2023 696750, Altura: 0
Valor: Jul/2023 700672, Altura: 0
Valor: Mai/2023 761030, Altura: 1
Valor: Set/2023 710901, Altura: 0

Maior valor da árvore ABB: Mai/2023 761030
Menor valor da árvore ABB: Fev/2023 566108
Tempo de execução total para árvore ABB nanosegundos:16371800

Maior valor da árvore AVL: Mai/2023 761030
Menor valor da árvore AVL: Fev/2023 566108
Tempo de execução total para árvore AVL em nanosegundos:20713700
```

# Gráficos e Análises de Resultados




Na utilização do Dataset nós tivemos acesso a dados desde janeiro de 2020 a julho de 2024, esses dados foram filtrados por mês e separados em grupos pela faixa etária, nele analisamos a quantidade de exames por grupo e por mês, obtendo os resultados para as perguntas exploratórias a seguir:

***Com base nos dados de saúde pública, quais faixas etárias registraram o maior número de exames realizados nos últimos anos? Quais fatores podem estar contribuindo para esse padrão observado?***

Observando os dados, constatamos que as faixas etárias de 60 a 64 anos e de 65 a 69 anos se destacam significativamente no número de exames realizados. Podemos inferir que indivíduos nesses grupos etários mais avançados realizam exames com maior regularidade devido a problemas de saúde relacionados ao envelhecimento, especialmente doenças cardiovasculares, que são o foco principal dos exames analisados. Além disso, é plausível afirmar que a frequência das visitas médicas aumenta nessa fase da vida devido à necessidade de monitoramento contínuo de condições crônicas e ao seguimento de recomendações médicas para prevenção e detecção precoce de possíveis complicações de saúde.

***Com base nos dados de saúde pública, quais faixas etárias registraram o menor número de exames realizados nos últimos anos? Quais fatores podem estar contribuindo para esse padrão observado?***

Analisando os dados do SUS, observa-se que a faixa etária com o menor número de exames realizados é a de pessoas com mais de 80 anos. Esse fenômeno pode ser atribuído a diversos fatores. Primeiramente, indivíduos nessa faixa etária podem enfrentar limitações físicas ou cognitivas que dificultam o acesso aos serviços de saúde, como problemas de mobilidade ou dependência de cuidadores. Além disso, há a possibilidade de que médicos e familiares optem por um manejo mais conservador, evitando exames e intervenções que possam não trazer benefícios significativos devido à fragilidade associada à idade avançada. Questões como menor expectativa de vida, comorbidades complexas e prioridades voltadas para a qualidade de vida em vez de procedimentos diagnósticos também podem influenciar na menor frequência de exames realizados nessa população.

***Os números obtidos após e durante a pandemia são mais altos e/ou preocupantes?***

Observando os dados, podemos ver que desde 2020 os exames já apresentam números muito altos, apresentando quase 200 milhões de exames, e esses números continuaram aumentando com o passar dos anos, batendo quase 300 milhões no ano de 2023. Ou seja, com o passar da pandemia os números foram aumentando cada vez mais, deixando a situação um tanto quanto alarmante. Analisando os dados que temos até o periodo de julho deste ano(2024), temos mais uma alarmante quantidade de 200 milhões de exames mas considerando que já foi passado mais de metade do ano, podemos esperar um número menor ou ao menos

semelhante ao ano de 2023, mostrando uma estabilidade nos casos e uma queda nos exames.

***Quais períodos do ano apresentam maior ou menor número de exames ? quais podem ser as causas ?***

Na análise, podemos ver um padrão que segue por todos os anos, os meses do final e do começo do ano costumam ter uma maioria de exames, meses como dezembro e janeiro possuem mais do que o resto do ano, e novembro e fevereiro seguem logo atrás enquanto abril é o mês com menor número de casos em todos os anos. Podemos assimilar isso a diversos fatores, um deles seria a época, no começo e no final do ano acontecem diversos eventos e comemorações, e nesses eventos muitas pessoas podem viajar se encontrar com outras e fazer coisas que normalmente não fazem no resto do ano, gerando doenças que os fariam realizar algum exame. E também podemos assumir que o clima pode gerar isso, com as mudanças das estações e a mudança de clima, muitas pessoas ficam doentes e precisam ser levadas a hospitais o que poderia aumentar bastante os casos.

# Conclusões

Após a análise do desempenho das duas árvores (ABB e AVL), concluímos algumas tendências em relação a exames médicos, como por exemplo, que pessoas de aproximadamente 60/70 anos costumam realizar mais exames que as outras faixas etárias. Assim como conseguimos ver o quanto a pandemia afetou a população num todo, com números consistentemente maiores nos anos de 2021 para 2023. Estudos como esses, mesmo que simples, nos permitiram entender melhor algumas características do povo brasileiro em relação à saúde.

Comparar as árvores lado a lado durante o processo de programação desse projeto nos deu uma visão melhor e mais prática da importância do estudo nessa área. Mesmo que pequenas, ambas as árvores possuem pontos fortes e fracos e quando implementadas bem e nos lugares corretos podem construir sistemas para armazenagem de dados muito eficientes. Também conseguimos entender melhor aquilo que aprendemos em sala de aula de uma maneira mais direta, o que ajudou e melhorou nosso entendimento da linguagem Java e de estruturas de dados em geral.

Neste projeto de extensão, lidamos com muitos desafios novos. Desde achar um arquivo csv útil, aplicar tal csv em java e integrá-lo nas árvores ABB e AVL, assim como analisar o desempenho de ambas as árvores utilizando o excel. Este trabalho nos ensinou muito e estamos muito empolgados em utilizar esses novos

entendimentos em outros projetos ou até mesmo em nossos estágios a fim de melhorar a vida daqueles ao nosso redor.

# Reflexão Final

## Bruno Rabelo Torchio de Oliveira:

Este projeto nos ofereceu muitos desafios mas também muitas oportunidades de aprendizagem. Manipular arquivos csv é uma habilidade que raramente usei após aprender os básicos da técnica em python. Poder aprender sua implementação em java e entender melhor a mecânica de como usá-las na prática foi uma experiência muito interessante.

Ter que encontrar um csv útil para o nosso trabalho também foi um pequeno desafio interessante que me ensinou mais do que eu esperava sobre achar esse tipo de informação.

Também pude refletir sobre a importância da ODS de promover o acesso à saúde, e o quanto a tecnologia é importante para esse processo. A saúde é um aspecto do dia a dia que raramente se dá atenção pois sempre focamos no quanto conseguimos ser produtivos e “úteis” tanto que acabamos afetando negativamente não só nós mesmos, como aqueles ao nosso redor.

Enfim, espero poder utilizar esses ensinamentos na prática no futuro. E também desejo que eu possa usar aquilo que aprendi para melhorar a vida e a saúde dos outros. Nem que seja de pequenas formas.

## Caio Teixeira Torres:

Trabalhar com temas relacionados aos Objetivos de Desenvolvimento Sustentável (ODS) é sempre uma experiência enriquecedora. Refletir sobre como a tecnologia pode ser empregada para melhorar a vida das pessoas, seja por meio da análise de dados, inteligência artificial ou estruturas de dados, é fascinante. A tecnologia sempre se apresenta como uma ferramenta poderosa para oferecer soluções inovadoras a problemas antigos.

No contexto deste trabalho, foi essencial reconhecer o papel fundamental das estruturas de dados na análise realizada. Utilizar e comparar as árvores AVL e ABB com base em seus desempenhos mostrou-se extremamente esclarecedor para futuras análises e estudos comparativos. Embora desafiador, trabalhar com essas duas estruturas proporcionou um aprendizado significativo, destacando a importância de enfrentar desafios como forma de crescimento. Aprendi muito sobre

manipulação de dados, sobre as estruturas de árvores, e principalmente, tive meu primeiro contato em como manipular um CSV em java.

Os conhecimentos adquiridos ao longo deste processo, tanto no domínio da linguagem Java quanto em pesquisa e na construção deste documento, certamente contribuirão para meu desenvolvimento acadêmico e profissional.

Espero no futuro poder voltar a trabalhar com temas da ODS para conseguir ajudar o mundo com a tecnologia e conhecimento.

## Guilherme Dias Ferreira Pereira:

A proposta da ODS de promover o acesso à saúde é, sem dúvida, um dos objetivos mais essenciais a serem alcançados. Melhorar a qualidade de vida é uma prioridade, e a utilização da tecnologia para enriquecer as nossas vidas e melhorar em abundância os aspectos da nossa sociedade, principalmente a saúde, é um objetivo que eu acredito que seja essencial para a nossa evolução como sociedade.

Ao pensar em soluções ou melhorias para qualquer cenário, é impossível ignorar o papel transformador da tecnologia. Desenvolver um projeto que nos desafia a criar formas de analisar e comparar dados não é apenas intelectualmente estimulante, mas também proporciona uma compreensão profunda da importância de sistemas eficazes para conceber as melhores soluções possíveis. Mesmo tendo experiência com programação e já tendo estudado o funcionamento dessas árvores, aplicar elas com essa finalidade me deixou ao mesmo tempo aflito, pois a quantidade de dados deixava a organização mais complexa do que eu estava acostumado, mas ao mesmo tempo intrigado, por que era uma maneira nova de organização que eu não tinha tanta prática, por isso eu digo que foi complicado mas que valeu a pena pelas horas de estudo, e pelo conhecimento na área da saúde, que é o setor que eu considero mais importante na nossa sociedade.

Considerando os conhecimentos obtidos neste projeto, me sinto mais capaz de utilizar árvores binárias como forma de analisar dados, e com mais conhecimento sobre saúde, me sinto ansioso para poder usar esse conhecimento no futuro e espero novamente utilizar para ajudar com o acesso à saúde.

## João Victor de Paula Silva:

Os Objetivos de Desenvolvimento Sustentável propostos pela ONU, quando implementados de forma eficaz, têm o potencial de melhorar significativamente a qualidade de vida global. Portanto, conscientizar as pessoas sobre esses objetivos é

essencial, e o propósito deste trabalho é exatamente esse: alertar e informar, com foco especial no 3º objetivo.

Em relação ao desenvolvimento do trabalho, foi minha primeira experiência com a análise de um arquivo CSV em Java. Embora eu já tenha familiaridade com a linguagem, integrar Estruturas de Dados com CSV se revelou um grande desafio. O processo de tratar esses dados, juntamente com o gerenciamento dos métodos das classes ABB e AVL, foi um aprendizado valioso que ampliou meus conhecimentos e habilidades.

Ademais, buscar dados do governo e filtrá-los no DATASUS me abriu um novo leque de oportunidades, em futuros projetos pessoais planejo utilizar mais dados desse tipo, para passar maior credibilidade além de incrementar um pouco na dificuldade.

# Referências


Link de acesso para os datasets:
https://tabnet.saude.sp.gov.br/deftohtm.exe?tabnet/SIA_MS_2008.DEF

ORACLE. *JDK 23 Documentation - Home*. Disponível em:
JDK 23 Documentation - Home. Acesso em: 15 nov. 2024.

YOUTUBE. *Árvore Binária de Busca*. Disponível em:
https://youtu.be/XZ0MEDhb4oE?si=29JIsHiR9sAhwgx6. Acesso em: 16 nov. 2024.

YOUTUBE. *Árvore AVL*. Disponível em:
https://youtu.be/3zmjQlJhBLM?si=Zxn9yNlb6Tyxhc-P. Acesso em: 18 nov. 2024.

YOUTUBE. *Java: Leitura e escrita de arquivos(.csv)*. Disponível em:
https://youtu.be/cHFQCT7_1VM?si=wvWy73sxvQxLdmFH. Acesso em 17 nov 2024.

ONU. *Objetivos de Desenvolvimento Sustentável*. Disponível em:
Objetivos de Desenvolvimento Sustentável | As Nações Unidas no Brasil Acesso em 14 nov.2024

GEEKSFORGEEKS. *Pattern Compile.* Disponível em:
https://www.geeksforgeeks.org/pattern-compilestring-method-in-java-with-examples/ Acesso em 24 nov. 2024.