PCC-534

Análise de Algoritmos e Estruturas de Dados

Prof.: André Pimenta Freire

PROJETO FINAL

"Análise e Visualização de Dados de Consumo de Água via IoT Utilizando Árvores AVL em Python"

Aluno: João Antonio Resende Paviani

1 - Introdução

O mundo moderno está cada vez mais imerso em dados. Com o advento das tecnologias de Internet das Coisas (IoT), a capacidade de coletar, transmitir e analisar dados em tempo real tornou-se não apenas viável, mas também indispensável em vários setores. Um dos domínios que mais se beneficiam dessa revolução tecnológica é o da gestão de recursos naturais, particularmente o consumo de água. Este projeto se insere nesse contexto, focando no monitoramento do consumo de água em um ambiente universitário. O monitoramento é especialmente relevante neste ambiente, onde o uso eficiente de recursos é crucial tanto para a sustentabilidade quanto para a gestão eficaz das instalações.

1.1 - Telemetria: Uma Visão Geral

Telemetria é o processo de coleta e transmissão de dados de dispositivos localizados em locais remotos para um sistema central para monitoramento e análise. Originário do grego "tele", que significa remoto, e "metron", que significa medida, a telemetria é fundamental em diversas áreas que vão desde o monitoramento ambiental e medicina até a indústria e transporte.

1.2 - Aplicações e Utilidade

Na era atual da Internet das Coisas (IoT), a telemetria ganha ainda mais relevância, permitindo a coleta de dados em tempo real de uma ampla gama de fontes. No contexto deste projeto, a telemetria é usada para monitorar o consumo de água em diferentes setores de uma instituição universitária. Sensores instalados em hidrômetros coletam dados sobre o volume de água usado e transmitem essas informações via LoRaWAN para um gateway centralizado. O protocolo LoRaWAN é especialmente relevante para aplicações de telemetria, dada a sua eficiência em termos de energia e capacidade de transmitir dados por longas distâncias. Após a coleta, os dados são então armazenados e processados em um banco de dados MySQL, permitindo uma análise mais aprofundada.

Um dos principais desafios na telemetria é a gestão eficaz e eficiente dos dados coletados. Isso envolve não apenas o armazenamento seguro dos dados, mas também a capacidade de processá-los e analisá-los de forma que possam ser usados para tomar decisões informadas. Neste projeto, técnicas avançadas de estruturas de dados e algoritmos são empregadas para garantir que o sistema não seja apenas robusto, mas também escalável e eficiente.

1.3 - Objetivos específicos

O objetivo primordial deste projeto acadêmico é simular um sistema IoT para coleta e armazenamento de dados de consumo de água em vários setores de uma universidade, de forma semelhante ao projeto que está sendo desenvolvido na UFLA. Além disso, visa demonstrar o uso eficiente de estruturas de dados avançadas, como árvore AVL, para o gerenciamento e a análise desses dados.. A meta é otimizar a performance do sistema, garantindo uma recuperação de dados rápida e eficaz, e oferecer insights valiosos que possam ser aplicados para tornar o consumo de água mais eficiente no projeto real em produção.

1.4 - Contribuição do Projeto

Este projeto se destaca não apenas por sua implementação técnica, mas também por seu impacto social e ambiental. Ao focar no monitoramento do consumo de água em um ambiente universitário, ele aborda uma questão crítica de sustentabilidade que tem implicações muito além do campus. Utilizando tecnologias de vanguarda como Internet das Coisas (IoT) e LoRaWAN, junto com algoritmos e estruturas de dados avançados, o projeto oferece um modelo robusto, escalável e eficiente para o gerenciamento de recursos hídricos. Além disso, o ambiente simulado em um contexto acadêmico permite uma investigação mais profunda sem interferir em sistemas críticos, tornando este projeto não apenas inovador, mas também eticamente responsável.

1.5 - Estrutura do Documento

O restante deste documento está organizado da seguinte forma: A Seção 2 abordará o "Cenário", detalhando o fluxo de dados desde a coleta até a visualização. A Seção 3, intitulada "Estrutura de dados", será dedicada a explicar as estruturas de dados e algoritmos utilizados, incluindo uma análise comparativa para justificar as escolhas feitas. A Seção 4 apresentará o "Código e Implementação", fornecendo um mergulho geral no código e suas funcionalidades. A Seção 5 concluirá com uma "Análise de Resultados", onde os dados coletados serão avaliados em relação aos objetivos do projeto, seguida pela “Conclusão”. Finalmente, na seção 6 serão apresentadas as principais referências que foram consultadas no decorrer do trabalho.

2 - Cenário

O projeto real em produção na Universidade Federal de Lavras (UFLA) representa uma implementação completa de um sistema de monitoramento de consumo de água e energia. Neste sistema, hidrômetros e sensores de energia são instalados em diferentes setores da universidade para coletar dados essenciais. Estes dispositivos transmitem as informações coletadas via uma rede LoRaWAN até um gateway centralizado. O gateway, posicionado estrategicamente para otimizar a coleta de dados, é responsável por receber essas informações e encaminhá-las para um servidor através do protocolo MQTT.

No servidor, um conjunto de algoritmos foi desenvolvido para processar e refinar os dados. Um deles, denominado processa_dados, retira os dados brutos armazenados no servidor e realiza diversas operações para torná-los mais úteis e analisáveis. Este algoritmo filtra, ordena e estrutura os dados antes de devolvê-los ao servidor, onde são armazenados em um banco de dados MySQL para futuras análises. Este banco de dados funciona como o repositório central de todas as informações coletadas, permitindo uma recuperação eficiente e análises detalhadas.

Para a apresentação e visualização desses dados, é utilizado o Metabase, uma plataforma de Business Intelligence que permite a criação de dashboards interativos e relatórios analíticos. Com isso, os gestores e tomadores de decisão têm acesso a informações atualizadas e insights valiosos sobre o consumo de água e energia na instituição.

Esse cenário apresenta desafios específicos, incluindo a necessidade de garantir a segurança dos dados, a escalabilidade do sistema e a eficiência na recuperação e análise dos dados coletados. É nesse ambiente complexo e multifacetado que este projeto acadêmico se insere, buscando não apenas replicar, mas também otimizar e expandir as capacidades do sistema atual.

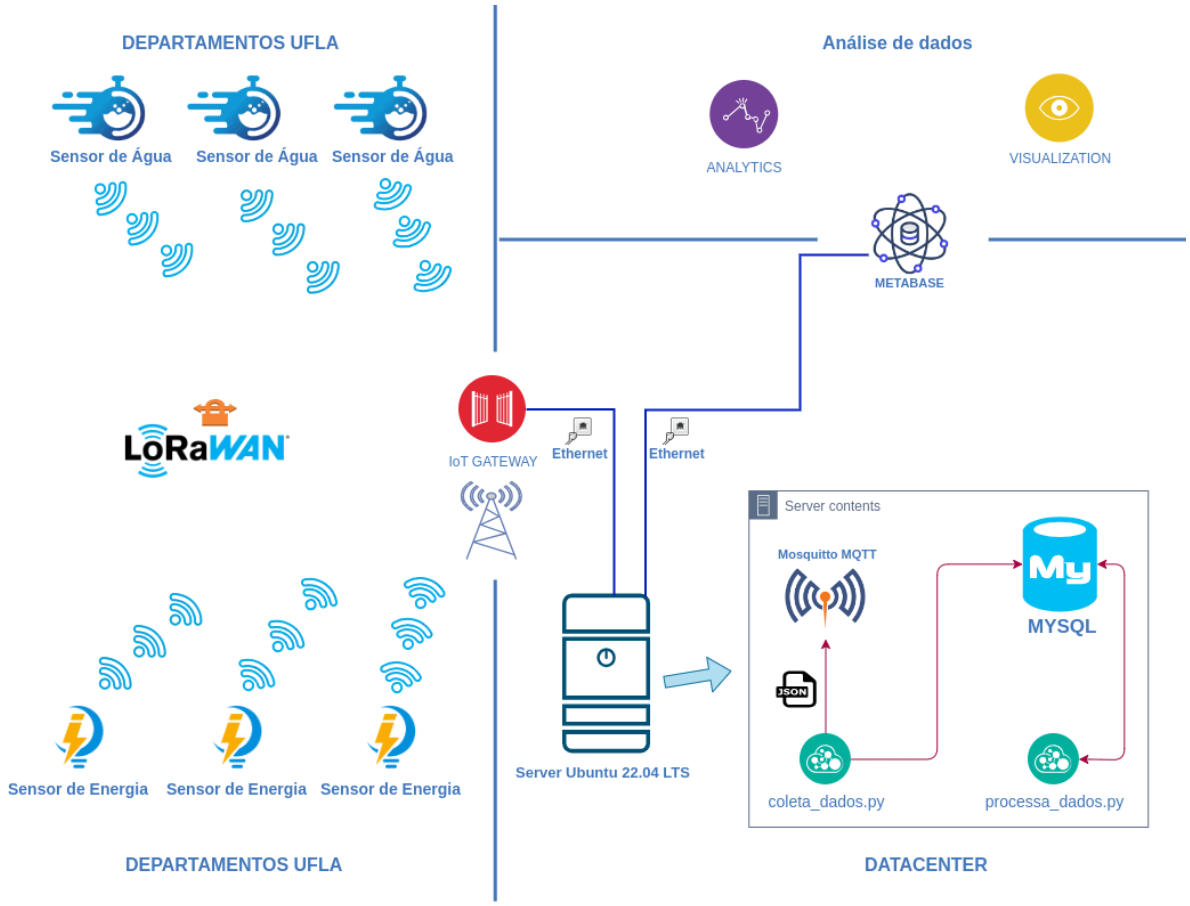


Figura 1: *Arquitetura do Sistema de Monitoramento de Consumo de Água e Energia na Universidade Federal de Lavras (UFLA)*

2.1 - Cenário Proposto

Devido à natureza sensível e crítica de sistemas em produção, experimentações acadêmicas e otimizações de algoritmos devem ser realizadas em um ambiente controlado. Por este motivo, foi criado um projeto acadêmico que simula o sistema real. O foco desse projeto acadêmico é a Análise de Algoritmos e Estrutura de Dados (AAED), e o objetivo é estudar e avaliar a eficiência dos algoritmos e estruturas de dados empregados na coleta, armazenamento e análise dos dados de consumo de água.

O cenário deste projeto é uma simulação de um ambiente universitário com múltiplos setores onde dados de consumo de água são coletados por hidrômetros. Estes dispositivos transmitem os dados através da tecnologia LoRaWAN para um gateway centralizado. Em nosso projeto acadêmico, essa transmissão é simulada através da criação de arquivos JSON que contêm informações como a data e hora da coleta (timestamp), endereço do

dispositivo (devAddr), volume de água coletado (valorAgua), e o setor universitário ao qual pertence (setor).
Neste ambiente simulado, uma pasta contendo vários arquivos JSON atua como um substituto para o que, em um ambiente de produção, seria o output do gateway MQTT. Estes arquivos JSON simulam os dados transmitidos pelos dispositivos IoT através de LoRaWAN. O algoritmo-chave desenvolvido especificamente para este projeto é encarregado de ler esses arquivos diretamente desta pasta. Após a leitura, o algoritmo extrai as informações relevantes e as armazena em uma Árvore AVL, que emula o banco de dados MySQL do sistema real.

Uma vez que os dados são armazenados na Árvore AVL, o sistema realiza uma travessia in-order para organizar as informações em ordem cronológica, baseando-se no campo "timestamp". O resultado final é um arquivo CSV que contém os dados ordenados, facilitando futuras análises e visualizações.

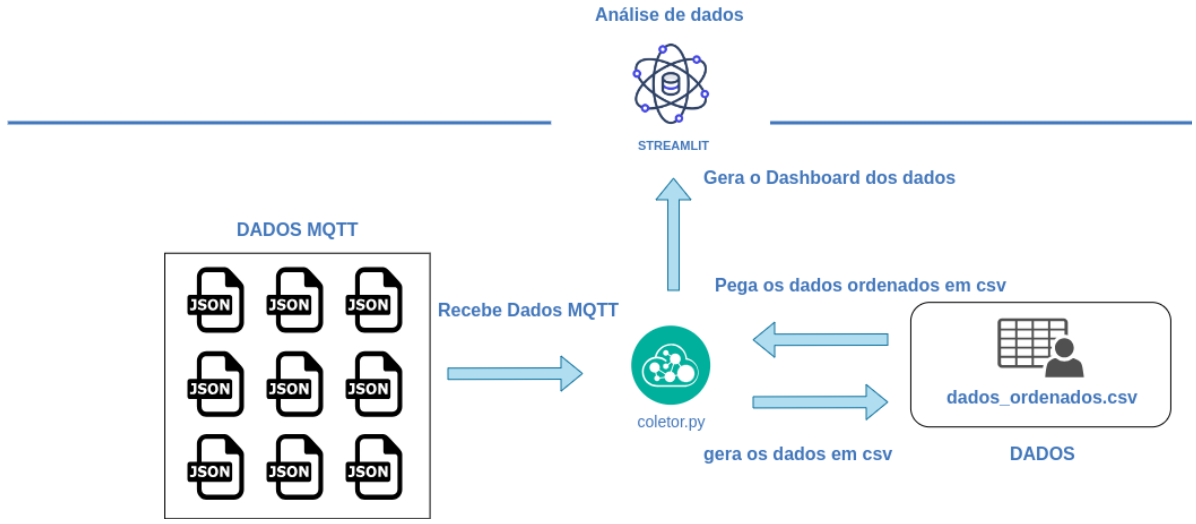


Figura 2: *Cenário do ambiente simulado com o coletor de dados.*

3 - Estrutura de Dados

Este capítulo explora a estrutura de dados escolhida, a Árvore AVL, para o armazenamento e a organização eficientes dos dados de consumo de água. Detalharemos seu funcionamento, justificativas para sua escolha e como ela contribui para a otimização do sistema

3.1 - Estrutura de dados utilizada

Árvore AVL: Utilizada para armazenar as informações "devAddr", "timestamp", "setor" e "valorAgua" dos arquivos JSON. Os dados são automaticamente ordenados por "timestamp", facilitando análises temporais e agregações.
Listas: Utilizadas para o armazenamento temporário e manipulação dos dados lidos dos arquivos JSON antes de serem inseridos na Árvore AVL.

3.2 - Tecnologias Utilizadas

Python: Utilizado como a linguagem de programação principal para scripting e automação do projeto, incluindo a leitura de arquivos JSON, o gerenciamento de dados na árvore AVL e a criação de visualizações.
JSON: Empregado para simular mensagens recebidas dos dispositivos IoT em um ambiente de produção. Os arquivos JSON contêm dados fictícios que representam as leituras de consumo de água.
Streamlit: Usado para a visualização dos dados coletados e processados. Permite a criação de dashboards interativos para análise dos dados de consumo de água.

3.3 - Escolha da Estrutura de Dados

Este subcapítulo busca elucidar as decisões tomadas em relação à seleção de estruturas de dados e algoritmos utilizados no projeto. As escolhas foram feitas com base em critérios como eficiência, eficácia e adequação ao problema em questão.

3.3.1 - Árvore AVL

Uma árvore AVL é um tipo especial de árvore binária de pesquisa que mantém seu equilíbrio durante as operações de inserção e remoção. Isso significa que a árvore é projetada para minimizar o número de etapas necessárias para encontrar um elemento, otimizando assim o tempo de pesquisa para se aproximar de O(log n), onde n é o número total de nós.
Para manter esse estado equilibrado, a árvore AVL realiza ajustes estruturais, conhecidos como rotações, sempre que um novo nó é inserido ou removido. Essas rotações ajudam a redistribuir os nós de tal forma que a altura da árvore seja mantida dentro de um limite aceitável, garantindo assim a eficiência das operações de pesquisa. As tarefas de localizar, adicionar e excluir nós têm uma complexidade de tempo que tende a O(log n), tornando a Árvore AVL uma estrutura de dados eficiente para aplicações que exigem buscas rápidas e frequentes.

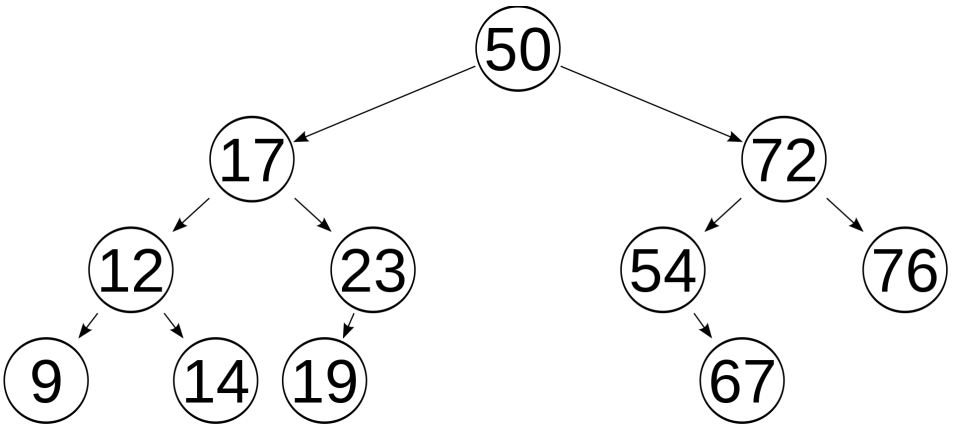


Figura 3: *Exemplo de uma árvore AVL.*

3.3.2 - Árvore AVL vs Outras Estruturas

Este subcapítulo examina as razões pelas quais a Árvore AVL foi selecionada em detrimento de outras estruturas de dados potencialmente aplicáveis, como tabelas hash, listas ligadas ou mesmo árvores binárias de busca simples (ABB). O principal fator que levou à escolha da Árvore AVL foi sua eficácia em manter os dados ordenados, o que é crucial para as análises temporais e agregações necessárias neste projeto. Além disso, a Árvore AVL oferece a vantagem de auto-balanceamento, o que garante um tempo de busca eficiente, independentemente da quantidade de dados armazenados. Este balanceamento é particularmente útil em cenários onde inserções e remoções ocorrem frequentemente, mantendo o desempenho do sistema otimizado.

3.4 - Análise de Complexidade

Este subcapítulo tem como objetivo examinar a complexidade computacional das principais operações realizadas pelo sistema. Isso inclui a inserção e busca na Árvore AVL

3.4.1 - Inserção na Árvore AVL

A inserção em uma Árvore AVL envolve duas etapas principais: a inserção do novo nó e o balanceamento subsequente da árvore para manter suas propriedades AVL. A complexidade de tempo para inserção é *O(log n)* onde *n* é o número de nós na árvore. Isso se deve ao fato de que a altura da árvore é mantida logarítmica devido ao auto-balanceamento. A eficiência na inserção foi um fator crucial para a escolha desta estrutura de dados, especialmente considerando que o sistema precisa armazenar e recuperar registros em tempo real.

3.4.2 - Busca na Árvore AVL

A busca na Árvore AVL possui uma complexidade de tempo de *O(log n)* onde *n* é o número de nós na árvore. Tal eficiência é crucial para o sistema, pois permite consultas rápidas e recuperação de dados em tempo real, um aspecto fundamental para o monitoramento e a análise do consumo de água.

3.4.3 - Complexidade Espacial

A complexidade espacial da Árvore AVL é *O(n)*, onde *n* é o número total de nós. Cada nó na árvore armazena uma chave, bem como informações adicionais como o fator de equilíbrio, e possui referências para seus nós filhos (esquerdo e direito). Portanto, a memória usada é proporcional ao número de elementos armazenados, tornando a estrutura eficiente também em termos de espaço.

3.4.4 - Resumo da Análise

A análise de complexidade demonstra que a Árvore AVL é uma estrutura de dados altamente eficiente para este projeto. Ela oferece operações de busca e inserção eficientes com complexidade de tempo O(log n), além de manter um uso eficiente de espaço com complexidade O(n). Esta eficiência faz da Árvore AVL uma escolha ideal para um sistema que necessita de rápido acesso aos dados, bem como de uma organização ordenada desses dados. Portanto, a estrutura se alinha bem com as exigências de eficiência em tempo e espaço do projeto.

4 - Código e implementação

Este capítulo oferece um olhar detalhado sobre o código Python que simula o monitoramento de consumo de água. Vamos explorar as funcionalidades principais e como o código atende aos objetivos do projeto.

4.1 - Visão Geral do código

Este tópico fornece uma visão geral da estrutura do código Python que simula o monitoramento de consumo de água.

O código é escrito em Python e faz uso da Árvore AVL para armazenar e organizar os dados de consumo de água de forma eficiente. A implementação também inclui a visualização dos dados por meio do Streamlit, permitindo uma análise intuitiva.

O código é dividido em três partes principais:
Leitura dos dados: Esta parte do código lê os dados de consumo de água de arquivos JSON.
Implementação da Árvore AVL e Armazenamento dos Dados: Esta parte do código armazena os dados de consumo de água em uma árvore AVL.
Visualização dos dados: Esta parte do código visualiza os dados de consumo de água por meio do Streamlit.

4.2 - Pré-requisitos e configurações iniciais

Para instalar o projeto, siga estas etapas:

1. Faça o download do código fonte do GitHub através do link ***https://github.com/Paviani/PROJETO_AAED***.
2. Certifique-se de que você tem o Python 3.11 instalado. Caso contrário, você pode baixá-lo ***https://www.python.org/downloads/.***
3. Instale as dependências necessárias usando o comando `pip install -r requirements.txt`.
4. Execute o projeto usando o comando `streamlit run coletor.py`.

4.3 - Leitura dos dados

Este tópico discute como o código lê os dados de consumo de água de arquivos JSON.

- Formato dos dados
- Funções de leitura de dados

4.3.1 - formato de dados

Os dados de consumo de água são armazenados em arquivos JSON. Cada arquivo JSON contém um conjunto de registros de consumo de água. Cada registro é representado por um objeto JSON com os seguintes campos:

timestamp: O timestamp do registro de consumo de água.
devAddr: O endereço do dispositivo que realizou a leitura do consumo de água.
setor: O setor onde o dispositivo está localizado.
valorAgua: O valor do consumo de água.

4.3.2 - Funções de leitura de dados

O código fornece duas funções para ler os dados de consumo de água de arquivos JSON:

ler_arquivo_json(arquivo_json): Essa função lê um arquivo JSON e retorna uma lista de registros de consumo de água.

ler_arquivos_json(diretorio): Essa função lê todos os arquivos JSON em um diretório e retorna uma lista de registros de consumo de água.

```
# Função para ler um arquivo JSON
def ler_arquivo_json(nome_arquivo):
    with open(nome_arquivo, 'r') as f:
        return json.load(f)
```

Figura 5: Função para ler um arquivo JSON.

4.4 - Implementação da Árvore AVL e Armazenamento dos Dados

A Árvore AVL (Adelson-Velsky e Landis) é uma estrutura de dados de árvore binária de busca balanceada que foi implementada no script Python para organizar e armazenar dados relacionados ao consumo de água. A seguir, detalhamos como a Árvore AVL foi implementada e utilizada no script.

4.4.1 - Classe

```
# Classe para representar os nós da árvore AVL
class No:
    def __init__(self, timestamp, dado):
        self.esquerda = None
        self.direita = None
        self.timestamp = timestamp
        self.dado = dado
        self.altura = 1
```

Figura 6: Classe para representar os nós da árvore AVL.

A classe `No` é a estrutura básica que representa cada nó na Árvore AVL. Segue abaixo os atributos da classe:

- **esquerda:** Referência ao nó filho à esquerda.
- **direita:** Referência ao nó filho à direita.
- **timestamp:** O timestamp do dado armazenado, usado como chave.
- **dado:** O dado armazenado no nó, que pode ser um objeto complexo.
- **altura:** A altura do nó na árvore, usada para balanceamento.

O construtor (`__init__`) inicializa esses atributos quando um novo objeto `No` é criado.

4.4.2 - Inserção de Nós na Árvore

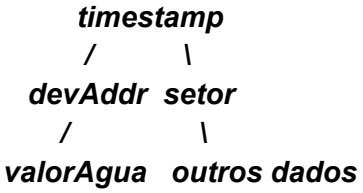
```
# Função para inserir um novo nó na árvore AVL
def inserir_no(raiz, timestamp, dado):
    if raiz is None:
        return No(timestamp, dado)
    else:
        if timestamp < raiz.timestamp:
            raiz.esquerda = inserir_no(raiz.esquerda, timestamp, dado)
        else:
            raiz.direita = inserir_no(raiz.direita, timestamp, dado)

    raiz.altura = 1 + max(altura(raiz.esquerda), altura(raiz.direita))
    return balancear(raiz)
```

Figura 7: Função para inserir um novo nó na árvore AVL

Para cada arquivo JSON lido, o script extrai o campo 'timestamp' e o utiliza como chave para a inserção do nó na árvore. A função inserir_no é responsável por esta tarefa e também chama a função balancear para assegurar que a árvore permaneça balanceada após cada inserção.

Neste contexto, cada nó da árvore representa um registro de consumo de água. O campo timestamp serve como a chave única para cada nó, permitindo uma organização eficiente e uma recuperação rápida dos dados. Os campos ***devAddr***, ***setor*** e ***valorAgua*** são armazenados como parte do dado em cada nó, mas não são utilizados para a organização da árvore em si, como na ilustração abaixo:



Cada nó da árvore AVL contém os seguintes campos:

timestamp: O timestamp do registro de consumo de água.
devAddr: O endereço do dispositivo que realizou a leitura do consumo de água.
setor: O setor onde o dispositivo está localizado.
valorAgua: O valor do consumo de água.
outros dados: Outros dados que podem ser armazenados no registro de consumo de água, como fator de espalhamento, o nível do sinal, a pressão da água, etc.

Essa estrutura de dados é adequada para este projeto porque permite uma organização eficiente dos dados de consumo de água. O campo timestamp é usado para organizar os dados em ordem crescente, o que facilita a recuperação dos dados.

4.4.3 - Cálculo da Altura do Nó

```
# Função para obter a altura de um nó
def altura(no):
    if no is None:
        return 0
    return no.altura
```

Figura 8: Função para obter a altura de um nó.

Esta função, altura, retorna a altura do nó passado como argumento. Se o nó for None (ou seja, um nó vazio), a função retorna 0. Caso contrário, ela retorna o valor do atributo altura do nó.

4.3.4 - Balanceamento da Árvore

```
# Função para balancear um nó na árvore AVL
def balancear(no):
    if no is None:
        return no

    balanceamento = altura(no.esquerda) - altura(no.direita)

    # Rotação Simples à Direita
    if balanceamento > 1:
        # Rotação Esquerda-Direita
        if altura(no.esquerda.esquerda) < altura(no.esquerda.direita):
            no.esquerda = rotacao_esquerda(no.esquerda)
        return rotacao_direita(no)

    # Rotação Simples à Esquerda
    if balanceamento < -1:
        # Rotação Direita-Esquerda
        if altura(no.direita.direita) < altura(no.direita.esquerda):
            no.direita = rotacao_direita(no.direita)
        return rotacao_esquerda(no)

    return no
```

Figura 9: Função para balancear um nó na árvore AVL

.

A função balancear recebe um nó como argumento e calcula o fator de balanceamento desse nó. O fator de balanceamento é calculado como a diferença entre a altura da subárvore esquerda e da subárvore direita.

Se o fator de balanceamento estiver fora do intervalo [-1, 1], isso indica que a árvore está desbalanceada e são necessárias rotações para trazer a árvore de volta ao equilíbrio.

Esta função é chamada sempre que um novo nó é inserido ou removido, garantindo que a árvore permaneça balanceada durante todas as operações, o que é crucial para manter a eficiência da estrutura de dados.

4.3.5 - Rotações para Balanceamento

```
# Função para realizar a rotação simples à direita
def rotacao_direita(y):
    x = y.esquerda
    T = x.direita

    x.direita = y
    y.esquerda = T

    y.altura = 1 + max(altura(y.esquerda), altura(y.direita))
    x.altura = 1 + max(altura(x.esquerda), altura(x.direita))

    return x

# Função para realizar a rotação simples à esquerda
def rotacao_esquerda(x):
    y = x.direita
    T = y.esquerda

    y.esquerda = x
    x.direita = T

    x.altura = 1 + max(altura(x.esquerda), altura(x.direita))
    y.altura = 1 + max(altura(y.esquerda), altura(y.direita))

    return y
```

Figura 10: Função para realizar a rotação simples à direita.

4.3.5.1 - Rotação Simples à Direita

A função rotacao_direita ajusta o balanceamento da árvore, girando nós para a direita. É usada quando a subárvore esquerda é mais pesada.

4.3.5.2 - Rotação Simples à Esquerda

A função rotacao_esquerda faz o oposto, girando nós para a esquerda. É aplicada quando a subárvore direita é mais pesada.

Ambas as rotações atualizam as alturas dos nós envolvidos e são essenciais para manter a árvore AVL balanceada.

4.3.5.3 - Rotações Duplas para Balanceamento

Além das rotações simples à direita e à esquerda, foram implementadas rotações duplas para lidar com casos específicos de desbalanceamento. Essas rotações são chamadas de rotação dupla esquerda-direita (LR Rotation) e rotação dupla direita-esquerda (RL Rotation).

Rotação Dupla Esquerda-Direita (LR Rotation)

```
# Função para realizar a rotação dupla esquerda-direita (LR Rotation)
def rotacao_esquerda_direita(y):
    y.esquerda = rotacao_esquerda(y.esquerda)
    return rotacao_direita(y)
```

Figura 11: Função para realizar a rotação dupla esquerda-direita (LR Rotation).

Esta rotação é aplicada quando um nó se torna desequilibrado devido a uma inserção na subárvore direita do filho esquerdo. Primeiro, realiza-se uma rotação simples à esquerda no filho esquerdo, seguida por uma rotação simples à direita no nó desbalanceado.

Rotação Dupla Direita-Esquerda (RL Rotation)

```
# Função para realizar a rotação dupla direita-esquerda (RL Rotation)
def rotacao_direita_esquerda(x):
    x.direita = rotacao_direita(x.direita)
    return rotacao_esquerda(x)
```

Figura 12: Função para realizar a rotação dupla direita-esquerda (RL Rotation).

A rotação dupla direita-esquerda é aplicada em uma situação inversa, onde o desequilíbrio ocorre devido a uma inserção na subárvore esquerda do filho direito. Primeiramente, executa-se uma rotação simples à direita no filho direito, seguida por uma rotação simples à esquerda no nó desbalanceado.

4.3.6 - Travessia In-Order

```
# Função para realizar a travessia "in-order" na árvore e armazenar os nós em uma lista
def travessia_inorder(raiz, resultado):
    if raiz:
        travessia_inorder(raiz.esquerda, resultado)
        resultado.append(raiz.dado)
        travessia_inorder(raiz.direita, resultado)
```

Figura 13: Função para realizar a travessia "in-order" na árvore e armazenar os nós em uma lista.

Uma vez que todos os dados são inseridos, uma travessia in-order é realizada para coletar os dados em ordem crescente de 'timestamp'. A função travessia_inorder é utilizada para esta finalidade.

4.3.7 - Leitura e Gravação de Dados

4.3.7.1 - Leitura de Dados de Arquivo JSON

```
# Função para ler um arquivo JSON
def ler_arquivo_json(nome_arquivo):
    with open(nome_arquivo, 'r') as f:
        return json.load(f)
```

Figura 14:Função para ler um arquivo JSON.

A função ler_arquivo_json é responsável por ler um arquivo JSON e retornar seus dados como um objeto Python. A função usa a biblioteca json do Python para fazer a leitura e a conversão dos dados.

4.3.7.2 - Gravação de Dados em Arquivo CSV

```
# Função para gerar um arquivo CSV a partir dos dados ordenados
def gerar_csv(dados_ordenados):
    cabecalho = ['devAddr', 'timestamp', 'setor', 'valorAgua']
    with open(os.path.join(caminho_pasta_dados, 'dados_ordenados.csv'),
'w', newline='') as arquivo_csv:
        escritor = csv.writer(arquivo_csv)
        escritor.writerow(cabecalho)
        for dado in dados_ordenados:
                escritor.writerow([dado['devAddr'], dado['timestamp'],
dado['setor'], dado['valorAgua']])
```

Figura 15:Função para gerar um arquivo CSV a partir dos dados ordenados.

Depois de realizar a travessia in-order e ordenar os dados, a função gerar_csv é utilizada para criar um arquivo CSV que contém os dados ordenados. O cabeçalho do CSV inclui os campos 'devAddr', 'timestamp', 'setor', e 'valorAgua'.

4.3.8 - Inicialização da Árvore

```
# Inicializando a árvore AVL
raiz = None
```

Figura 16:Inicializando a árvore AVL.

A árvore é inicializada como uma estrutura vazia, com a variável raiz definida como None. Esta variável atuará como o nó raiz da árvore.

4.3.9 - Execução da Travessia In-Order e Armazenamento dos Dados

```
# Realizando a travessia in-order e armazenando os dados em uma lista
dados_ordenados = []
travessia_inorder(raiz, dados_ordenados)
```

Figura 17:Realizando a travessia in-order e armazenando os dados em uma lista.

Este trecho de código chama a função travessia_inorder, que percorre a árvore em ordem e armazena os dados ordenados na lista dados_ordenados. Esta lista é posteriormente usada para exportar os dados para um arquivo CSV.

4.3.10 - Geração de Arquivo CSV ordenado

```
# Função para gerar um arquivo CSV a partir dos dados ordenados
def gerar_csv(dados_ordenados, nome_arquivo):
    cabecalho = ['devAddr', 'timestamp', 'setor', 'valorAgua']
       with  open(os.path.join(caminho_pasta_dados,  nome_arquivo),  'w',
newline='') as arquivo_csv:
        escritor = csv.writer(arquivo_csv)
        escritor.writerow(cabecalho)
        for dado in dados_ordenados:
                escritor.writerow([dado['devAddr'], dado['timestamp'],
dado['setor'], dado['valorAgua']])
```

Figura 18:Função para gerar um arquivo CSV a partir dos dados ordenados.

Esta função recebe a lista dados_ordenados e o nome do arquivo CSV como argumentos. Ela cria um novo arquivo CSV e escreve o cabeçalho seguido dos registros de dados, um por linha.

4.3.7 - Visualização com Streamlit

```
def load_data():
    return pd.read_csv("dados/dados_ordenados.csv")

# Carregar os dados pela primeira vez
df = load_data()


# Botão para recarregar dados
if st.button('Recarregar Dados'):
    df = load_data()

# Título da página
st.title("Análise de Consumo de Água")

# Mostrar os dados brutos em uma tabela
st.write("Dados brutos:")
st.write(df)

# Criar um gráfico de barras para o consumo de água por setor
st.write("Gráfico de Consumo de Água por Setor:")
fig1, ax1 = plt.subplots()
df.groupby("setor")["valorAgua"].sum().plot(kind='bar', ax=ax1)
ax1.set_ylabel("Consumo de Água")
st.pyplot(fig1)

# Criar um gráfico de pizza para o consumo de água por setor
```

```
st.write("Gráfico de Pizza de Consumo de Água por Setor:")
fig2, ax2 = plt.subplots()
df.groupby("setor")["valorAgua"].sum().plot(kind='pie',
autopct='%1.1f%%', startangle=90, ax=ax2)
ax2.axis('equal')  # Manter a proporção de aspecto igual garante que o
gráfico de pizza seja desenhado como um círculo.
st.pyplot(fig2)
```

Figura 19:Estrutura de visualização com Streamlit

O script utiliza a biblioteca Streamlit para fornecer uma interface de usuário onde os dados e gráficos relacionados ao consumo de água são exibidos.

- Carrega os dados do arquivo CSV para um DataFrame do Pandas.
- Fornece um botão para recarregar os dados.
- Exibe os dados brutos em uma tabela.
- Gera um gráfico de barras e um gráfico de pizza para representar o consumo de água por setor.

5 - Análise de Resultados

Este capítulo tem como objetivo avaliar o desempenho do código e os resultados alcançados em relação aos objetivos iniciais do projeto.

5.1 - Desempenho do Código

Neste segmento, avaliamos indicadores chave de desempenho, como tempo de execução e eficácia na organização dos dados. A figura a seguir demonstra o tempo médio de execução do código:

```
(venv) PS C:\Users\jarpa\Documents\DESENVOLVIMENTO\AAED\MESTRADO\PROJETO> python coletor.py
2023-10-13 14:16:46.094
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run coletor.py [ARGUMENTS]
Tempo de execução de 0.6754422187805176
(venv) PS C:\Users\jarpa\Documents\DESENVOLVIMENTO\AAED\MESTRADO\PROJETO> python coletor.py
2023-10-13 14:16:53.404
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run coletor.py [ARGUMENTS]
Tempo de execução de 0.6928670406341553
(venv) PS C:\Users\jarpa\Documents\DESENVOLVIMENTO\AAED\MESTRADO\PROJETO> python coletor.py
2023-10-13 14:16:57.367
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run coletor.py [ARGUMENTS]
Tempo de execução de 0.6752829551696777
```

Figura 20:Testes de tempo de execução.

O tempo de execução médio é de 0.6812 segundos. Considerando que o algoritmo tem uma complexidade de tempo ***O(log n)*** para as operações de inserção e busca na Árvore AVL, este resultado é considerado adequado para o volume de dados processados.

5.2 - Resultados Obtidos

Os resultados obtidos demonstram que o projeto atingiu seus objetivos iniciais. A estrutura de dados Árvore AVL utilizada permitiu uma organização eficiente dos dados de consumo de água, facilitando análises rápidas e precisas. A interface do Streamlit proporcionou uma visualização intuitiva dos dados, permitindo uma fácil interpretação dos padrões de consumo de água.

5.2.1 - Dados Brutos Ordenados

A tabela abaixo apresenta um conjunto de dados brutos que foram ordenados pelo código. Este arquivo CSV é um resultado direto das operações de inserção e balanceamento executadas na árvore AVL.

devAddr	timestamp	setor	valorAgua
devAddr14	2023-10-10 06:41:21	DMV	114.0
devAddr20	2023-10-10 12:28:30	DMV	119.0
devAddr10	2023-10-10 14:20:46	DCS	110.0
devAddr13	2023-10-10 21:32:06	BIBLIOTECA	113.0
devAddr19	2023-10-11 03:28:30	DCC	119.0
devAddr16	2023-10-11 08:02:57	DNU	116.0
devAddr3	2023-10-11 12:31:05	RU	180.0
devAddr6	2023-10-11 15:41:42	DCS	107.1
devAddr8	2023-10-11 17:31:30	RU	152.3
devAddr4	2023-10-11 23:00:33	DBI	110.7
devAddr9	2023-10-12 00:08:01	RU	107.6
devAddr12	2023-10-12 01:42:46	RU	112.0
devAddr2	2023-10-12 02:29:05	DNU	95.2
devAddr5	2023-10-12 05:01:23	DMV	130.4
devAddr15	2023-10-12 09:24:56	DCS	115.0
devAddr7	2023-10-12 09:48:51	DCS	128.4
devAddr17	2023-10-12 17:41:10	DMV	117.0
devAddr1	2023-10-12 20:36:20	DGTI	120.5
devAddr18	2023-10-12 21:28:42	DBI	118.0
devAddr11	2023-10-12 21:38:32	BIBLIOTECA	111.0

Figura 21: Dados Brutos ordenados

A ordenação dos dados por 'timestamp' permite uma análise mais direta e eficiente, tornando mais fácil identificar padrões ou anomalias no consumo de água ao longo do tempo.

5.3 - Análise Gráfica dos Dados

Para uma compreensão mais profunda dos dados coletados e organizados, foram gerados dois tipos de gráficos: um gráfico de barras e um gráfico de pizza. Ambos os gráficos foram criados com base nos dados brutos ordenados apresentados na seção anterior.

5.3.1 - Gráfico de Barras

O gráfico de barras exibe o consumo total de água por setor. Este tipo de visualização facilita a identificação de quais setores têm maior ou menor consumo de água, o que é crucial para análises futuras e tomadas de decisão.

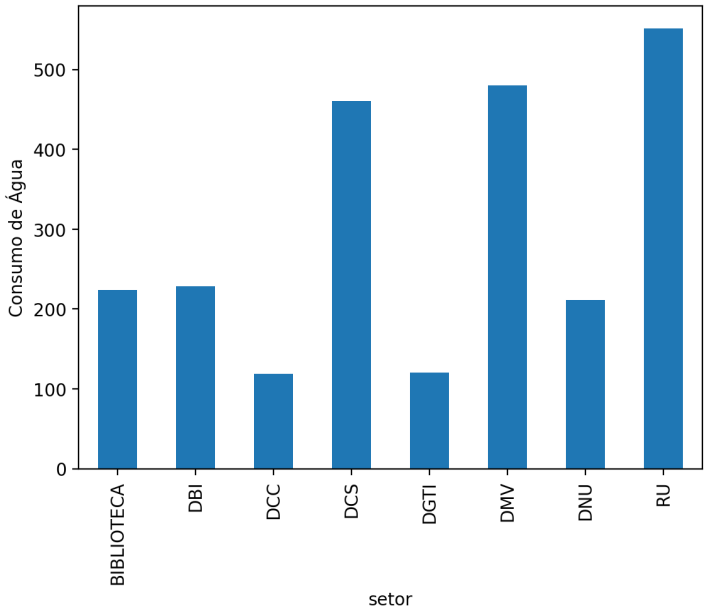


Figura 22: Gráfico de Barras

5.3.1 - Gráfico de Pizza

O gráfico de pizza apresenta a proporção do consumo de água por setor em relação ao consumo total. Este gráfico é útil para entender a distribuição percentual do consumo de água entre os diferentes setores.

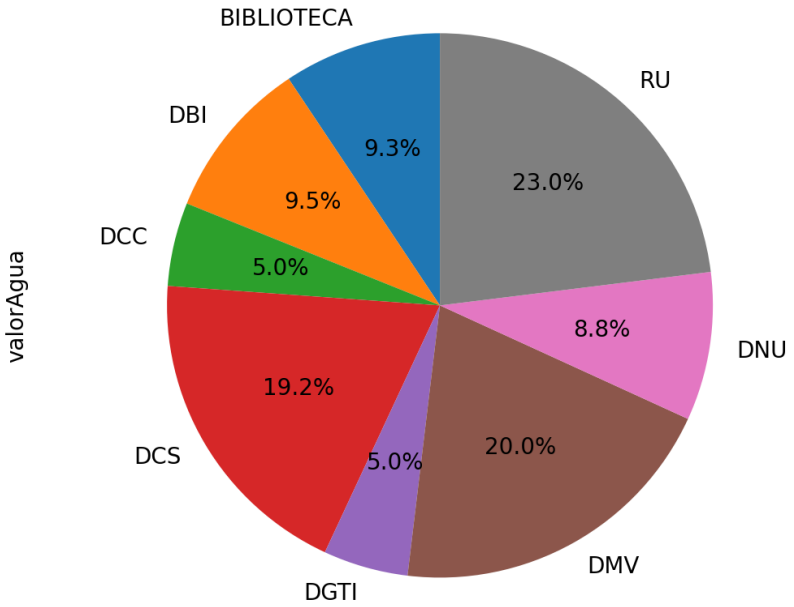


Figura 23: Gráfico de Pizza

Ambos os gráficos fornecem insights valiosos sobre o consumo de água e podem ser utilizados para diversas análises, desde a identificação de padrões de consumo até a detecção de possíveis anomalias ou ineficiências em setores específicos.

5.4 - Conclusão

O projeto apresentado demonstrou a aplicação prática de estruturas de dados avançadas, especificamente a Árvore AVL, no contexto do monitoramento de consumo de água. A escolha dessa estrutura de dados mostrou-se eficaz tanto em termos de organização dos dados quanto em eficiência computacional, com um tempo médio de execução de aproximadamente 0.681 segundos.

A capacidade de armazenar, organizar e recuperar grandes volumes de dados em tempo hábil é crucial em qualquer sistema de monitoramento. Nesse sentido, a implementação da Árvore AVL permitiu não apenas armazenar os dados de forma eficiente, mas também facilitou operações de busca e análise, como demonstrado pelos gráficos gerados.

Em resumo, este projeto não apenas atingiu seus objetivos iniciais mas também destacou a importância da escolha apropriada de estruturas de dados em aplicações do mundo real. O sucesso do projeto reforça o valor de combinar conhecimentos teóricos e práticos em estruturas de dados para resolver problemas complexos de forma eficiente.

6 - Referências:

Udemy. Estrutura de Dados e Algoritmos em Python: Guia Completo. Udemy. Disponível em: https://www.udemy.com/course/estrutura-de-dados-e-algoritmos-python-guia-completo/.

Python Software Foundation. Data Structures. Disponível em: https://docs.python.org/3/tutorial/datastructures.html.

GeeksforGeeks. AVL Tree (Adelson-Velsky and Landis Tree). Disponível em: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/.

Streamlit. Streamlit Documentation. Disponível em: https://docs.streamlit.io/.

The Pandas Development Team. Pandas Documentation. Disponível em: https://pandas.pydata.org/pandas-docs/stable/.

Matplotlib Development Team. Matplotlib Documentation. Disponível em: https://matplotlib.org/stable/contents.html.