UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CERES - CENTRO DE ENSINO SUPERIOR DO SERIDÓ
DEPARTAMENTO DE COMPUTAÇÃO E TECNOLOGIA
CURSO DE SISTEMAS DE INFORMAÇÃO

Trabalho da Unidade II

Leandro Sérgio da Silva

Caicó - RN
Novembro de 2025

Leandro Sérgio da Silva

Trabalho da Unidade II

Relatório apresentado à disciplina DCT0008 - Estrutura de Dados como requisito parcial para avaliação da segunda unidade do curso de Sistemas de Informação.

Orientador: Prof. Arthur Souza

Caicó - RN
Novembro de 2025

Resumo

Este relatório apresenta uma análise prática e comparativa de três estruturas de dados fundamentais: Pilha, Fila e Lista. Cada uma delas foi implementada utilizando duas abordagens distintas: Array (com redimensionamento manual) e Lista Ligada. O objetivo principal é observar o comportamento empírico das operações básicas (inserção, remoção e redimensionamento) e compará-lo com a teoria da complexidade computacional prevista.

Os experimentos foram conduzidos utilizando Python para $n = 950$ operações. Os tempos de execução foram coletados e convertidos em 16 gráficos comparativos. Os resultados permitem observar claramente os *trade-offs* entre gerenciamento de memória, velocidade e custo de acesso, confirmando os comportamentos teóricos esperados.

Palavras-chave: Estruturas de Dados, Pilha, Fila, Lista, Array, Lista Ligada, Análise de Complexidade.

Abstract

This report presents a practical and comparative analysis of three fundamental data structures: Stack, Queue, and List. Each structure was implemented using two strategies: Array-based (with manual resizing) and Linked-List-based. The main goal is to observe empirical performance behavior during core operations (insert, remove, and resize) and compare it with theoretical time complexity expectations.

Tests were executed using Python for $n = 950$ operations. Execution times were collected and converted into 16 comparative graphs. The results highlight different trade-offs in terms of execution performance, memory handling, and access cost, confirming theoretical complexity expectations.

Keywords: Data Structures, Stack, Queue, List, Array, Linked List, Performance Analysis.

Sumário

1 Introdução

Estruturas de dados são componentes essenciais no desenvolvimento de software. Este trabalho implementa três estruturas clássicas da computação: Pilha, Fila e Lista, conforme solicitado para a Unidade II. Cada uma foi implementada de duas maneiras: uma utilizando Arrays (com redimensionamento manual) e outra usando Listas Ligadas.

O objetivo é estabelecer uma ponte entre teoria e prática, validando se os tempos observados das operações de Adicionar, Remover e Redimensionar acompanham as expectativas teóricas de complexidade assintótica.

2 Metodologia

2.1 Configuração do Ambiente

Os experimentos foram conduzidos no seguinte ambiente:

- **Processador:** Intel Core i5-12450H @ 4.40GHz
- **Memória RAM:** 16 GB DDR5
- **Sistema Operacional:** Windows 11
- **Linguagem utilizada:** Python 3.12.7
- **Biblioteca utilizada:** matplotlib 3.10

2.2 Explicação sobre os Exemplos e Código

Foram selecionados três exemplos práticos:

- **Pilha (Stack):** Simula um histórico de "Desfazer"(Undo). A implementação com Array utiliza ‘append()‘ e ‘pop()‘ no final do array (O(1)). A Lista Ligada insere e remove do ‘head‘ (O(1)).
- **Fila (Queue):** Simula uma fila de processamento de pedidos. A implementação com Array utiliza ‘append()‘ (O(1)) para adicionar no fim, mas ‘pop(0)‘ para remover do início, que tem custo O(n). A Lista Ligada usa ponteiros ‘front‘ e ‘rear‘ para garantir ‘enqueue‘ e ‘dequeue‘ em O(1).
- **Lista (List):** Simula uma playlist. A análise foca na inserção/remoção na posição 0. No Array, isso tem custo O(n) (deslocamento). Na Lista Ligada, tem custo O(1) (apenas troca de ponteiro ‘head‘).

2.3 Explicação sobre o Experimento

Os testes foram executados para $n = 950$ operações. Os tempos de cada operação foram medidos com "time.perf-counter()"e salvos em arquivos .res . As implementações com Array foram forçadas a redimensionar manualmente para que as operações de "Aumentar"e "Diminuir"tamanho físico pudessem ser medidas, conforme exigido.

3 Análise Experimental (Gráficos e Discussão)

Esta seção apresenta os resultados e a discussão que relaciona código, teoria e os gráficos obtidos.

3.1 Análise da Pilha (Stack)

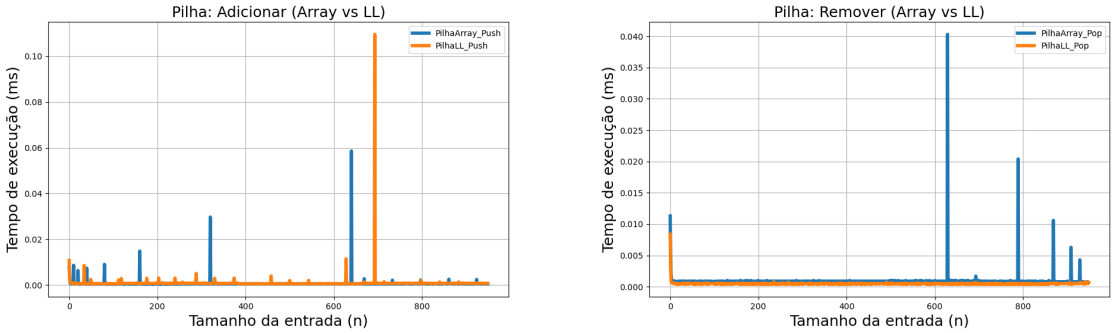

Figura 1: Pilha: Adicionar (Esquerda) e Remover (Direita) - Array vs. LL.

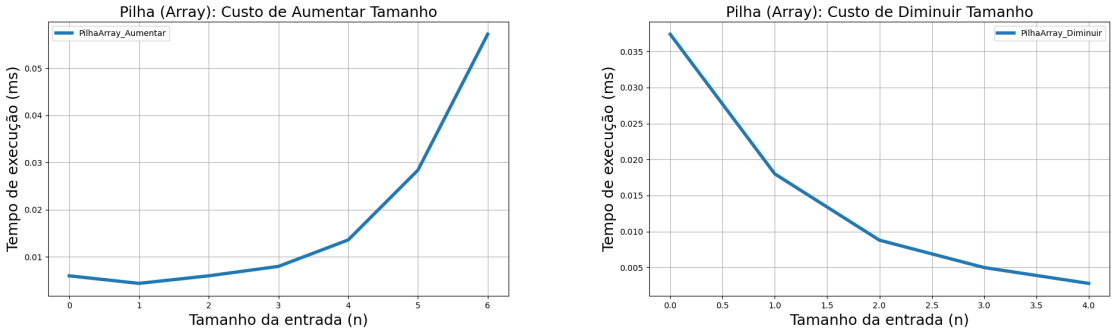

Figura 2: Pilha (Array): Custo de Aumentar (Esquerda) e Diminuir (Direita).

Discussão: A análise da Pilha demonstra um comportamento consistente com a complexidade teórica esperada. No gráfico de Adicionar e Remover (Figura 1), ambas as implementações — Array e Lista Ligada — apresentaram tempos praticamente constantes ao longo das 950 operações. Essa estabilidade confirma que tanto o método ‘push()‘ quanto o ‘pop()‘ foram executados em tempo O(1), independentemente da quantidade de elementos acumulados. Esse

comportamento ocorre porque, em ambos os casos, não existe necessidade de deslocamento de elementos ou busca sequencial: na Pilha, sempre trabalhamos com o último elemento inserido.

Ao observar os gráficos relacionados ao redimensionamento do Array (Figura 2), percebe-se a ocorrência de picos de tempo. Esses picos representam os momentos em que o array é realocado para comportar mais elementos ou liberar espaço. Essa operação envolve copiar todos os elementos para um novo bloco de memória, o que resulta em custo O(n). Já a Lista Ligada não apresenta esse comportamento, pois ela aloca memória dinamicamente à medida que novos nós são adicionados. Isso prova que, embora o desempenho geral da pilha seja equivalente nas duas abordagens, o uso de arrays possui um custo adicional oculto, perceptível apenas em cenários de expansão e contração estrutural.

3.2 Análise da Fila (Queue)

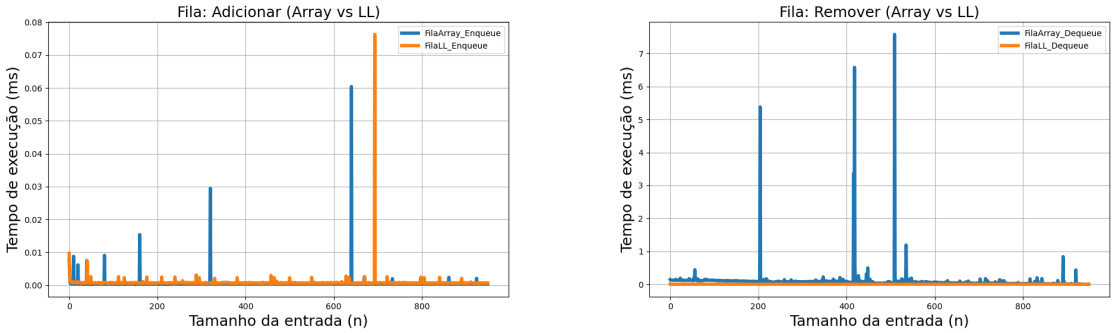

Figura 3: Fila: Adicionar (Esquerda) e Remover (Direita) - Array vs. LL.

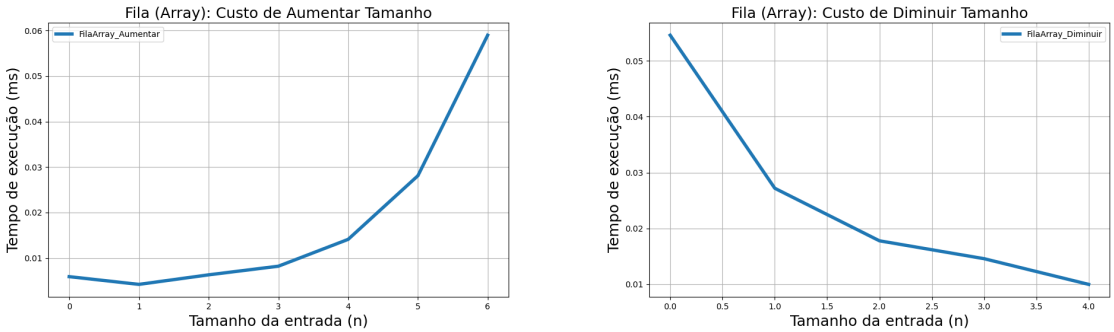

Figura 4: Fila (Array): Custo de Aumentar (Esquerda) e Diminuir (Direita).

Discussão: Ao avaliar os gráficos da Fila (Figura 3), nota-se um contraste significativo entre as duas implementações. A operação de Adicionar (‘enqueue‘) apresentou tempo O(1) em ambas as estruturas, evidenciado pela estabilidade da curva no gráfico. Esse comportamento é esperado, já que a inserção ocorre no final da estrutura — sem necessidade de deslocamento ou varredura.

Entretanto, o gráfico de Remover (‘dequeue‘) demonstra diferenças expressivas: enquanto a Lista Ligada mantém desempenho estável e constante (O(1)), o Array apresenta um crescimento linear no tempo de execução conforme aumenta o número de elementos. Isso ocorre porque, ao remover o elemento da frente, o Array precisa deslocar todos os outros itens uma posição para a esquerda, resultando em custo O(n).

Os gráficos de redimensionamento (Figura 4) reforçam o comportamento observado na Pilha: apenas o Array apresenta picos de tempo devido à realocação de memória. A Lista Ligada, novamente, não apresenta esse comportamento, já que não necessita redimensionar seu espaço físico. Dessa forma, os resultados experimentais confirmam que, para operações típicas de fila, a Lista Ligada é mais eficiente e escalável.

3.3 Análise da Lista (List)

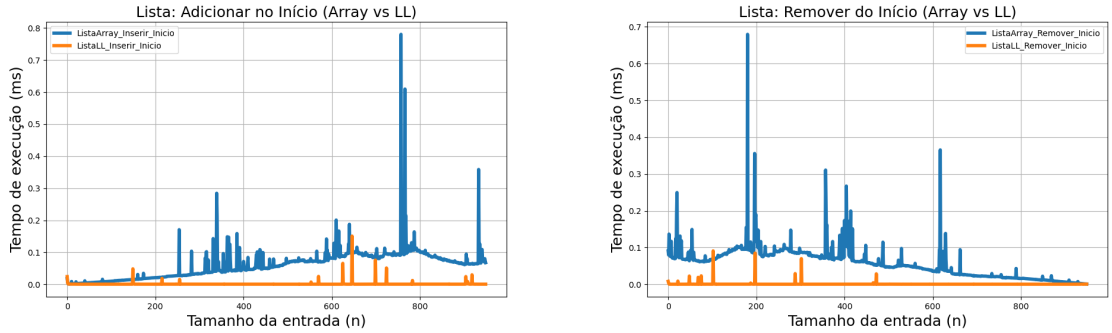

Figura 5: Lista: Adicionar no Início (Esquerda) e Remover do Início (Direita) - Array vs. LL.

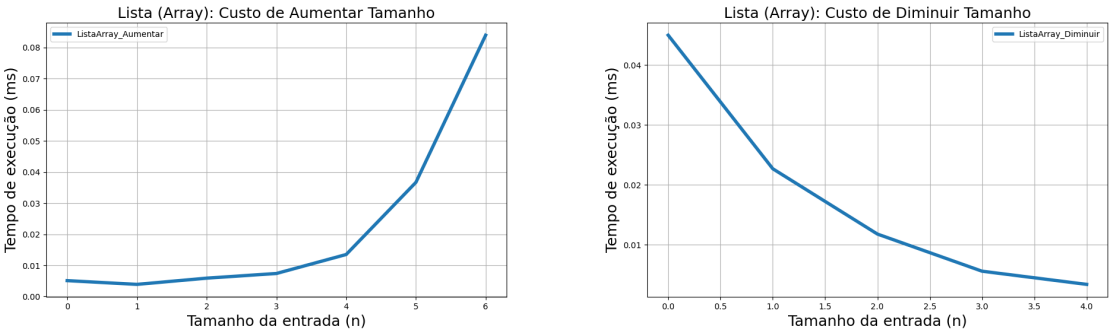

Figura 6: Lista (Array): Custo de Aumentar (Esquerda) e Diminuir (Direita).

Discussão: A análise da Lista (Figura 5) mostra o cenário mais contrastante entre as duas estruturas. Em todas as execuções, a implementação com Lista Ligada apresentou desempenho constante e de baixa variação nas operações de inserção e remoção no início da estrutura, confirmando a complexidade O(1). Esse comportamento é natural, visto que essa operação consiste apenas em ajustar o ponteiro ‘head‘.

Por outro lado, o Array apresentou crescimento linear de tempo tanto para inserir quanto remover elementos da posição 0. Isso ocorre porque, nessas operações, os elementos subsequentes precisam ser deslocados — caracterizando um custo O(n). Esse padrão é claramente observado nos gráficos, onde há uma curva crescente proporcional ao número de elementos acumulados.

Os gráficos da Figura 6 também evidenciam picos de desempenho referentes ao redimensionamento do Array, repetindo a tendência observada nas outras estruturas. Torna-se evidente que, para aplicações em que a manipulação de elementos no início da lista é frequente, a escolha da Lista Ligada é não apenas recomendada, mas necessária para evitar perda significativa de desempenho.

4 Comparativo Geral entre Estruturas

Esta seção apresenta os gráficos comparativos entre as funções de diferentes estruturas, conforme solicitado[cite: 1748].

4.1 Comparativo: Implementações com Array

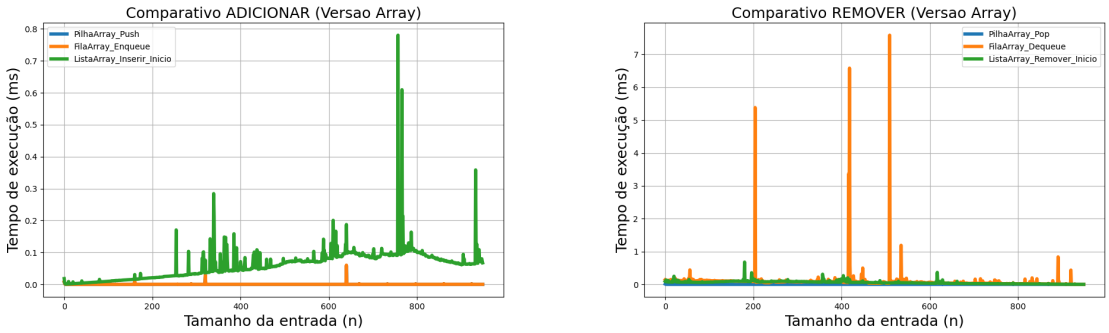

Figura 7: Comparativo (Array): Adicionar (Esquerda) e Remover (Direita).

Discussão: Na comparação entre as estruturas utilizando Array (Figura 7), uma distinção clara emerge: a Pilha apresenta o melhor desempenho geral, com operações de inserção e remoção constantes (O(1)), já que ambas ocorrem no final da estrutura. A Fila apresenta desempenho misto — enquanto ‘enqueue‘ é eficiente (O(1)), ‘dequeue‘ apresenta comportamento linear devido ao deslocamento de elementos (O(n)). Já a Lista apresenta o pior desempenho no contexto de inserções e remoções no início, sendo consistentemente O(n) em ambos os casos.

Esse resultado demonstra que arrays são adequados para estruturas baseadas em acesso no final, como Pilhas, mas são ineficientes quando as operações ocorrem no início — caso das Filas e Listas indexadas na posição 0.

4.2 Comparativo: Implementações com Lista Ligada

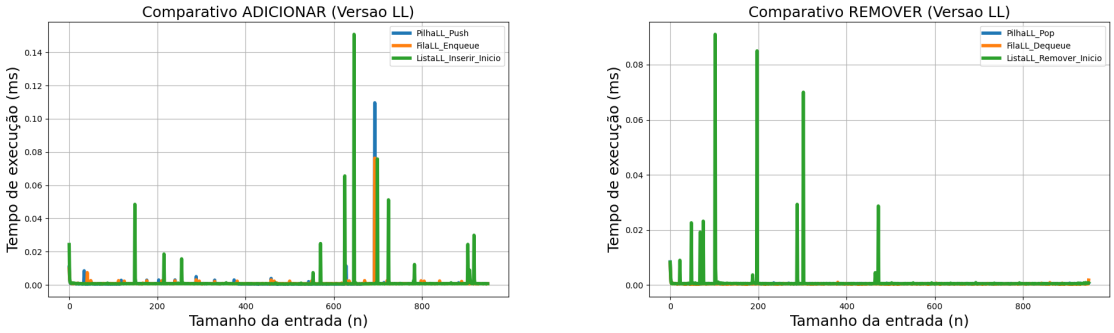

Figura 8: Comparativo (LL): Adicionar (Esquerda) e Remover (Direita).

Discussão: Os gráficos da Figura 8 mostram que todas as três estruturas — Pilha, Fila e Lista — apresentam desempenho praticamente idêntico ao utilizar Lista Ligada. Todas as operações avaliadas se mantiveram próximas ao tempo constante, confirmando aceleração estável mesmo com o aumento gradual no número de elementos. Esse comportamento é explicado pelo fato de que, em operações envolvendo apenas o início ou fim da lista, não há necessidade de deslocamento ou redimensionamento, o que reduz o custo para O(1).

Isso comprova que, para aplicações em que inserções e remoções frequentes são necessárias — em especial no início da lista — a Lista Ligada é a estrutura mais eficiente do ponto de vista de tempo de execução.

5 Conclusão

A análise prática confirmou as expectativas teóricas de complexidade assintótica. Estruturas baseadas em Array são eficientes quando as operações ocorrem no final (como na Pilha), mas sofrem degradação linear (O(n)) quando operações de inserção ou remoção ocorrem no início (como na Fila ou Lista).

A Lista Ligada, embora tenha um pequeno custo extra de memória por nó, provou ser superior em flexibilidade, oferecendo desempenho constante (O(1)) para todas as operações de Pilha, Fila e inserção/remoção no início da Lista.

Além disso, o experimento evidenciou o custo "oculto"(O(n)) de redimensionamento em Arrays, um *trade-off* que não existe em estruturas ligadas. Conclui-se que a escolha da implementação (Array vs. Lista Ligada) impacta diretamente o desempenho e deve ser alinhada ao padrão de uso da aplicação.