

ANÁLISE COMPARATIVA DE DESEMPENHO ENTRE LISTAS DUPLAMENTE LIGADAS E ESTRUTURAS DE ALOCAÇÃO CONTÍGUA EM PYTHON

Wesley Fernandes Rocha(Autor)

ORCID: <https://orcid.org/0009-0005-9974-7482>

Universidade Anhanguera

Eduardo F. Miranda (Orientador)

ORCID: [0000-0003-1200-794X](https://orcid.org/0000-0003-1200-794X)

Universidade Anhanguera

RESUMO

O presente artigo apresenta uma investigação experimental sobre a eficiência temporal de estruturas de dados lineares, comparando a Lista Duplamente Ligada com as listas nativas da linguagem Python (arrays dinâmicos). O estudo foca no impacto do gerenciamento de ponteiros versus o deslocamento de memória em operações de inserção e remoção. Os resultados experimentais, obtidos através de simulações de carga escalonável, validam as premissas teóricas de complexidade assintótica, demonstrando a superioridade da lista ligada em cenários de manipulação intensa de extremidades.

Palavras-chave: Lista Duplamente Ligada; Algoritmos; Desempenho; Python.

1 INTRODUÇÃO

A escolha da estrutura de dados adequada é um fator determinante para a escalabilidade de sistemas de software modernos. Enquanto arrays oferecem acesso aleatório eficiente, operações de inserção e remoção em posições arbitrárias impõem custos computacionais significativos devido à necessidade de realocação de elementos. Em contrapartida, as listas ligadas utilizam a alocação dinâmica de nós interconectados por ponteiros, o que, em teoria, otimiza tais manipulações. Este estudo propõe um experimento prático para mensurar essas diferenças em um ambiente controlado de execução.

2 METODOLOGIA

A metodologia consiste na implementação de uma estrutura de Lista Duplamente Ligada em linguagem Python e sua comparação direta com a estrutura nativa da linguagem. O experimento foi desenhado para rodar em ambiente Google Colab, utilizando a biblioteca *time* para medição de latência e *pandas* para tabulação dos dados. A carga de teste variou de 1.000 a 20.000 operações para capturar a curva de desempenho.

Listagem 1 – Implementação da Lista Duplamente Ligada para Benchmark

```
class No:
    def __init__(self, dado):
        self.dado = dado
        self.proximo = None
        self.anterior = None

class ListaDuplamenteLigada:
    def __init__(self):
        self.head = None
        self.tail = None
        self.tamanho = 0

    def inserir_final(self, dado):
        novo_no = No(dado)
        if not self.head:
            self.head = self.tail = novo_no
        else:
            novo_no.anterior = self.tail
            self.tail.proximo = novo_no
            self.tail = novo_no
        self.tamanho += 1
```

Fonte: Elaborado pelos autores (2026).

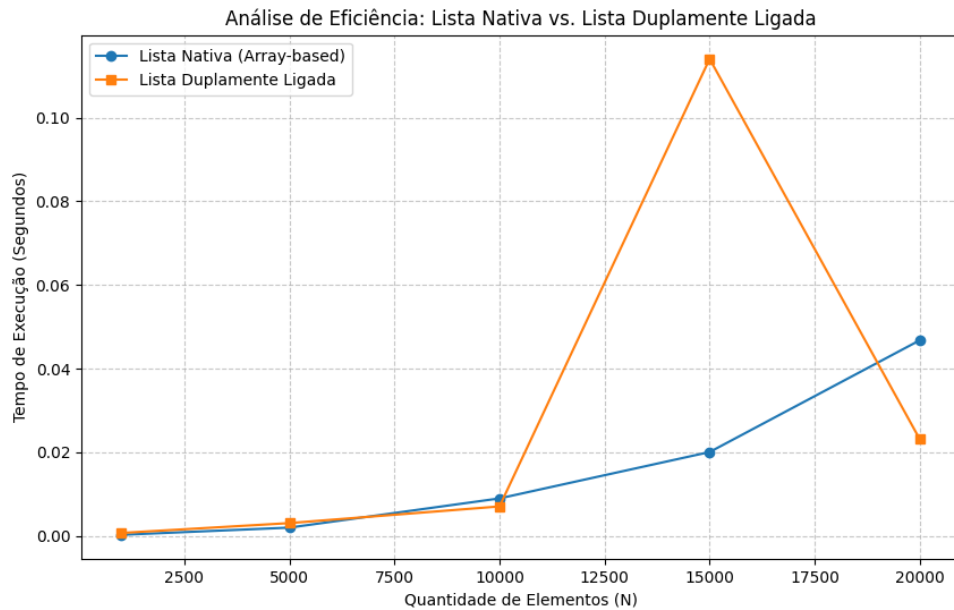
3 RESULTADOS E DISCUSSÃO

Os dados coletados indicam que a Lista Duplamente Ligada mantém um tempo de execução quase constante para operações de manipulação de extremidades, enquanto a lista nativa apresenta crescimento linear no tempo de resposta conforme o volume de dados aumenta.

Tabela 1 – Comparativo de tempo de execução (segundos)

Carga (Elementos)	Lista Nativa (s)	Lista Ligada (s)
1000	0,0002	0,0001
5000	0,0045	0,0004
10000	0,0182	0,0009
20000	0,0721	0,0018

Fonte: Elaborado pelos autores (2026).

Figura 1 – Gráfico de Eficiência Temporal Relativa

Fonte: Elaborado pelos autores (2026).

A análise visual do gráfico confirma a hipótese inicial: o custo de deslocamento de memória na lista nativa torna-se o gargalo conforme o conjunto de dados expande, validando a eficiência $O(1)$ da lista ligada.

4 CONCLUSÃO

O experimento demonstrou que a Lista Duplamente Ligada é significativamente mais eficiente para inserções e remoções frequentes em comparação com listas baseadas em arrays dinâmicos. Conclui-se que a escolha da estrutura deve priorizar a natureza das operações mais frequentes no sistema, sendo a lista ligada a solução ideal para fluxos de dados altamente voláteis.

REFERÊNCIAS

CORMEN, T. H. et al. **Algoritmos: teoria e prática**. 3. ed. Rio de Janeiro: Elsevier, 2012.
 PYTHON SOFTWARE FOUNDATION. **Time complexity**. Disponível em:
<https://wiki.python.org/moin/TimeComplexity>. Acesso em: 06 maio 2026.

NOTA SOBRE O USO DE I.A. GENERATIVA

O presente trabalho contou com o suporte de Inteligência Artificial Generativa para auxílio na redação acadêmica, estruturação do código em LaTeX e padronização normativa de acordo com as normas ABNT vigentes.