João Felipe Veras Dantas Rocha / DRE:117202753
Lucas dos Santos Melo / DRE:123312281
Rafael Augusto Santos / DRE:124155331
Rayan Lucas de Paula Carvalhal / DRE:124180742
Link para o GitHub dos arquivos com os códigos, assim como o arquivo de texto: https://github.com/jfvdrocha/BD-2025.1-hierarquia

Hierarquia de Classes ED Lineares

Professor: Milton Ramos Ramirez

Rio de Janeiro

2025

Comentários sobre o Desenvolvimento

Estruturas de Dados Utilizadas:
Nesta tarefa, implementamos uma hierarquia de classes para estruturas de dados lineares, incluindo `Array`, `SinglyLinkedList`, `DoublyLinkedList`, `CircularSinglyLinkedList`, `Pilha` (stack), e `Queue` (fila). Cada classe foi desenvolvida seguindo os requisitos do professor, com métodos para inserção, remoção, consulta e ordenação. A classe abstrata `LinearDataStructure` serviu como base para garantir a consistência entre as implementações. Além disso, resolvemos problemas específicos, como a fila do bandejão e o armazenamento de figuras geométricas, utilizando essas estruturas.

Divisão de Módulos:
O código foi organizado em módulos separados para cada estrutura de dados (`array_class.py`, `singly_linked_list.py`, etc.), o que facilitou a manutenção e o teste individual de cada classe. A classe abstrata `LinearDataStructure` definiu os métodos comuns, enquanto os testes unitários em `test_data_structures.py` verificaram a corretude das implementações.

Descrição das Rotinas e Funções:
- Inserção/Remoção: Todas as classes implementam métodos para inserir e remover elementos no início, fim e em posições específicas, com tratamentos para underflow e overflow.
- Consulta: Métodos como `peek_start`, `peek_end`, e `peek_at_index` permitem acessar dados sem removê-los.
- Ordenação: O algoritmo `bubble_sort` foi implementado nas listas para ordenação in-place.
- Problemas Resolvidos:
  - BandejaoQueue: Simula uma fila de usuários com estimativa de tempo de espera e ajustes dinâmicos.
  - GeometricFigure: Armazena vértices de figuras geométricas em uma lista circular e calcula perímetros.

Complexidades de Tempo e Espaço:
- Array: Inserção/remoção no início/fim têm complexidade O(n) devido aos deslocamentos, enquanto acesso por índice é O(1).
- Listas Encadeadas: Inserção/remoção no início/fim são O(1), mas inserção em posições arbitrárias é O(n).
- Pilha/Queue: Operações principais (push/pop, enqueue/dequeue) são O(1).

Problemas e Observações:
- Lista Circular: A implementação exigiu cuidado extra para manter a propriedade circular, especialmente nas operações de inserção e remoção.

- BandejaoQueue: A atualização dos tempos de espera após desistências ou ajustes foi um desafio, mas a solução com fila temporária mostrou-se eficaz.
- GeometricFigure: O cálculo do perímetro foi simples, mas garantir que os vértices formem figuras não cruzadas ficou como melhoria futura.

Conclusão
Os resultados obtidos foram satisfatórios, com todas as estruturas de dados funcionando conforme o esperado e os problemas resolvidos de maneira eficiente. A modularização do código facilitou a implementação e os testes, enquanto a herança da classe abstrata garantiu consistência. Alguns desafios, como a complexidade de operações em arrays e a manipulação de listas circulares, reforçaram a importância de escolher a estrutura adequada para cada cenário. No geral, a tarefa proporcionou um aprendizado valioso sobre estruturas de dados e suas aplicações práticas.

Observação Final:
A implementação da lista dupla e da fila de prioridades com ordenação foi particularmente interessante, mostrando como estruturas mais complexas podem otimizar operações específicas. No futuro, seria interessante explorar versões dinâmicas dessas estruturas para melhorar ainda mais a eficiência.