Relatório — Projeto classe_pilha

Nome: Bruno Videira Pinho DRE: 119.161.539

Estruturas de Dados Utilizadas

O projeto implementa as principais estruturas de dados lineares clássicas, incluindo:

- **Array**: Sequência indexada
- **SinglyLinkedList**: Lista encadeada simples
- **DoublyLinkedList** Lista encadeada dupla
- **Stack**: Pilha
- **Queue**: Fila simples
- **OneWayNode** e **TwoWaysNode**: Nós para listas encadeadas

Heraça de classes

```
LinearStruct                (Classe abstrata)
├── Array                   (Usa lista - padrão do python)
└── SinglyLinkedList        (usa OneWayNode)
    └── DoublyLinkedList    (usa TwoWaysNode)

Queue                       (usa Array)
Stack                       (usa SinglyLinkedList)
OneWayNode
TwoWaysNode
```

Divisão de Módulos

- **base.py**: Define a classe abstrata `LinearStruct` e a exceção `MethodNotImplemented`.
- **my_array.py**: Implementa a classe `Array`, uma sequência indexada com capacidade fixa ou dinâmica.
- **my_singly_linked_list.py**: Implementa a classe `SinglyLinkedList` usando nós do tipo `OneWayNode`.
- **my_doubly_linked_list.py**: Implementa a classe `DoublyLinkedList` usando nós do tipo `TwoWaysNode`.
- **my_stack.py**: Implementa a classe `Stack` usando como extrutura de armazenamento a classe `SinglyLinkedList`.
- **my_queue.py**: Implementa a classe `Queue` usando como extrutura de armazenamento a classe `Array`.
- **nodes.py**: Define as classes de nós `OneWayNode` e `TwoWaysNode`.
- **tests.py**: Contém testes unitários para todas as classes e operações

Principais Rotinas e Métodos

- **Métodos de Inserção**: Inserção no início, fim e por posição.
- **Métodos de Remoção de dados**: Remoção do início, fim e por posição.
- **Acesso e Atualização**: Métodos para acessar ou modificar elementos por posição.

- **Operadores Especiais**: Implementação de `__getitem__`, `__setitem__`, `__iter__` e `__next__` para suportar indexação e iteração (não implementado nas classes Stack e Queue).
- **Stack**: Métodos `push`, `pop`, `swap`, controle de overflow/underflow e verificação de tipo.
- **Queue**: Métodos `insert`, `pop`, `swap`, controle de overflow/underflow e verificação de tipo.
- **Listas Encadeadas**: Métodos para manipulação eficiente dos nós, incluindo inserção, remoção e iteração.

Complexidade de Tempo e Espaço

- **Array**: Acesso geral de dados O(1), inserção/remoção por posição O(n). Usam espaço proporcional à capacidade.
- **SinglyLinkedList/DoublyLinkedList**: Inserção/remoção no início e no final O(1), acesso por posição O(n). Usam espaço proporcional ao número de elementos.
- **Stack/Queue**: Operações principais (push/pop/insert/remove) O(1).

Problemas e Observações

- **Hierarquia imcompleta**: Não foi possível implementar hierarquia da classe abstrata `LinearStruct` com todas as outras classes.
    - Mesmo nas listas encadeadas não foi possível implementar por completo a classe abstrata `LinearStruct`, métodos `to_full` e `is_full`
- **Testes**: Teste unitários podem ser encontrados no arquivo `tests.py`, implementados com a biblioteca *pytest*
- **Iteração**: Todas as classes que herdam de `LinearStruct` podem ser usados com iteração (`for x in estrutura`).
- **Exceção**: Diversos lançamentos de excessão foram implementadas na extruturas.
    - Idealmente cada extrutura deve lançar suas próprias excessões (esse foi o intuito, pelo menos).

Conclusão

O projeto cumpre os requisitos de abstração e modularidade, implementando e testando as principais estruturas de dados lineares.