INSTITUTO FEDERAL DE ALAGOAS – IFAL / CAMPUS MACEIÓ
Curso Bacharelado em Sistemas de Informação
Disciplina de Estruturas de Dados

Prof. MSc. Ricardo Nunes
Ricardo (arroba) ifal.edu.br

Mini-projeto 1: Implementação de um Conjunto Usando Fila

Descrição: Neste projeto deve-se implementar uma estrutura de dados de conjunto (set) utilizando uma fila. A estrutura deve garantir que não haja elementos duplicados.

Realização e Entrega

- A partir de *07/10 até 14/10*
- Pode ser realizado *individualmente* ou em equipe com até *3 pessoas*
- **Este mini-projeto é opcional (extra)**
- Conteúdo a ser entregue:
  - arquivos do código fonte ou um link (github ou replit)
  - arquivo **leiame.txt** contendo:
    - integrantes da equipe (pode estar no link)
    - lista com todo o conteúdo de consultado para realização do projeto (sites, livros, artigos, códigos prontos, etc.)
    - comentário da equipe sobre se conseguiu ou não realizar tudo o que foi proposto.
    - comentário destacando possíveis problemas identificados no código, dificuldades encontradas na escrita e/ou funcionalidades que deveriam ser implementadas, mas não foram.
  - Discussão sobre desempenho das operações do tipo SetWithQueue
- Apresentar o projeto funcionando ao professor

Atividade 01: implementar SetWithQueue

- Implementar um tipo `SetWithQueue` que usa uma fila (Fila_Array) para armazenar elementos.
- Operações obrigatórias:
  - `add(element)` : Adiciona um elemento ao conjunto. Se o elemento já existir, não deve ser adicionado novamente.
  - `remove(element)` : Remove um elemento do conjunto. Se o elemento não existir, deve lançar uma exceção.
  - `contains(element)` : Retorna `True` se o elemento estiver no conjunto, `False` caso contrário.
  - `size()` : Retorna o número de elementos no conjunto.
  - `list()` : Retorna todos os elementos do conjunto.

Discussão sobre a Implementação

- Os alunos devem usar o tipo FilaArray, apresentado em sala de aula, para armazenar os elementos do conjunto e verificar a duplicidade. Ao adicionar um novo elemento, a implementação deve garantir que o elemento não esteja presente antes de adicioná-lo.

Atividade 02: Discussão sobre desempenho

- Acrescente no arquivo leiame.txt (ou readme.txt) discussão sobre desempenho de cada uma das operações desenvolvidas no projeto.

Critérios de Avaliação

1. Executar sem erros
2. Utilizar implementação do "Fila Array" como base para implementar o tipo SetWithQueue
3. Realizar a especificação solicitada
4. Passar no conjunto de testes apresetnados

5. Apresentar comentário sobre o desempenho das operações desenvolvidas
6. Indicar qualquer material consultado (codigo, livro, site, etc.) para realização do trabalho

Conjunto de Testes para `SetWithQueue`

Abaixo é fornecido um conjunto de de testes para verificar se o tipo implementado está correto.

Codigo disponível em https://github.com/ricardo9n/estd/blob/main/mp-testes/set_with_queue__tests.py

```
def test_add_unique_elements():
    set_queue = SetWithQueue()
    set_queue.add(1)
    set_queue.add(2)
    set_queue.add(3)
    assert set_queue.list() == [1, 2, 3], "Erro: Esperado [1, 2, 3]"

def test_add_duplicate_elements():
    set_queue = SetWithQueue()
    set_queue.add(1)
    set_queue.add(2)
    set_queue.add(3)
    set_queue.add(2)  # Deve ser ignorado
    set_queue.add(2)  # Deve ser ignorado
    set_queue.add(2)  # Deve ser ignorado
    set_queue.add(3)  # Deve ser ignorado
    set_queue.add(2)  # Deve ser ignorado
    set_queue.add(2)  # Deve ser ignorado
    assert set_queue.list() == [1, 2, 3], "Erro: Esperado [1, 2, 3]"

def test_remove_existing_element():
    set_queue = SetWithQueue()
    set_queue.add(1)
    set_queue.add(2)
    set_queue.add(3)
    set_queue.remove(1)
    assert set_queue.list() == [2,3], "Erro: Esperado [2,3]"
    set_queue.remove(2)
    assert set_queue.list() == [3], "Erro: Esperado [3]"
    set_queue.remove(3)
    assert set_queue.list() == [], "Erro: Esperado []"

def test_remove_non_existing_element():
    set_queue = SetWithQueue()
    set_queue.add(1)
    try:
        set_queue.remove(2)  # Deve lançar exceção
        assert False, "Erro: Exceção não lançada para elemento não existente"
    except ValueError as e:
        assert str(e) == "Element not found", "Erro: Mensagem de erro inesperada"

def test_contains_existing_element():
    set_queue = SetWithQueue()
    set_queue.add(1)
    assert set_queue.contains(1) is True, "Erro: Esperado True"

def test_contains_non_existing_element():
    set_queue = SetWithQueue()
    assert set_queue.contains(2) is False, "Erro: Esperado False"

def test_size_of_set():
    set_queue = SetWithQueue()
    set_queue.add(1)
    set_queue.add(2)
    assert set_queue.size() == 2, "Erro: Esperado 2"
    set_queue.remove(1)
    assert set_queue.size() == 1, "Erro: Esperado 1"

def test_list_empty_set():
    empty_set = SetWithQueue()
```

```
    assert empty_set.list() == [], "Erro: Esperado []"

def test_list_after_operations():
    set_queue = SetWithQueue()
    set_queue.add(1)
    set_queue.add(2)
    set_queue.remove(1)
    assert set_queue.list() == [2], "Erro: Esperado [2]"

def test_add_multiple_unique_elements():
    set_queue = SetWithQueue()
    for i in range(10):
        set_queue.add(i)
    assert set_queue.size() == 10, "Erro: Esperado 10"
    assert set_queue.list() == list(range(10)), "Erro: Esperado [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]"

def test_add_multiple_duplicate_elements():
    set_queue = SetWithQueue()
    for i in range(5):
        set_queue.add(1)  # Adicionando duplicados
    assert set_queue.size() == 1, "Erro: Esperado 1 após adição de duplicados"

def run_tests():
    test_add_unique_elements()
    test_add_duplicate_elements()
    test_remove_existing_element()
    test_remove_non_existing_element()
    test_contains_existing_element()
    test_contains_non_existing_element()
    test_size_of_set()
    test_list_empty_set()
    test_list_after_operations()
    test_add_multiple_unique_elements()
    test_add_multiple_duplicate_elements()
    print("Todos os testes passaram!")

if __name__ == "__main__":
    run_tests()
```

Descrição dos Testes

Cada função representa um teste específico para a implementação da classe `SetWithQueue`. Aqui está um resumo das funções:

- **test_add_unique_elements**: Verifica a adição de elementos únicos.
- **test_add_duplicate_elements**: Verifica se elementos duplicados são ignorados.
- **test_remove_existing_element**: Verifica a remoção de um elemento existente.
- **test_remove_non_existing_element**: Verifica se uma exceção é lançada ao tentar remover um elemento não existente.
- **test_contains_existing_element**: Verifica se o conjunto contém um elemento existente.
- **test_contains_non_existing_element**: Verifica se o conjunto não contém um elemento não existente.
- **test_size_of_set**: Verifica o tamanho do conjunto após várias operações.
- **test_list_empty_set**: Verifica a lista de um conjunto vazio.
- **test_list_after_operations**: Verifica a lista de elementos após operações de adição e remoção.
- **test_add_multiple_unique_elements**: Adiciona múltiplos elementos únicos e verifica a lista e o tamanho.
- **test_add_multiple_duplicate_elements**: Adiciona múltiplos duplicados e verifica o tamanho do conjunto.

Comentários adicionais:

- Uso de assert: Cada teste utiliza assert para validar o resultado. Se a condição não for verdadeira, uma mensagem de erro personalizada será exibida.
- Testes de Exceção: No teste de remoção de um elemento inexistente, o código captura a exceção e valida se a mensagem de erro é a esperada.
- Resultados: Se todos os testes forem bem-sucedidos, a mensagem "Todos os testes passaram!" será exibida no final da execução.

Execução dos Testes

Para executar os testes, basta rodar o código em um ambiente Python. A utilização de assert torna o processo de teste mais rigoroso e a identificação de falhas mais direta. Se um teste falhar, a execução será interrompida e uma mensagem de erro será apresentada.