Estruturas de dados lineares
Alocação contígua

Marco A L Barbosa malbarbo.pro.br

O código inicial dos exercícios está disponível na página https://malbarbo.pro.br/ensino/2024/6884/.

Introdução

1) O que são estruturas de dados?
2) Qual é a relação entre TADs e estruturas de dados?
3) O que é uma estrutura de dado linear?
4) Quais são as principais característica dos arranjos?

Pilha - Começando

5) O que é uma Pilha?
6) Considerando o algoritmo visto em sala que verifica se os grupos em uma expressão estão corretos, quantas operações de empilha e quantas de desempilha são realizadas para a entrada `[4 + ((2) + {4} * [3]] + ([1] + {3})`?
7) Projete uma função que receba como parâmetro uma pilha e modifique a pilha deixando ela vazia. Note que você precisa importar a classe `Pilha` do arquivo `pilha_arranjo` e o seu arquivo deve estar na mesma pasta que os arquivos `ed.py` e `pilha_arranjo.py`.

```
from pilha_arranjo import Pilha

def esvazia(pilha: Pilha):
    pass
```

8) Projete uma função que inverta a ordem dos caracteres de uma string. Use uma pilha para fazer a implementação.
9) Faça as seguintes alterações na implementação de `Pilha` do arquivo `pilha_arranjo.py`:
   a) Modifique o construtor para que ele receba como parâmetro a quantidade máxima de elementos que a pilha pode armazenar.
   b) Adicione um método para verificar se a pilha está cheia. Use o novo método para simplificar a implementação de `empilha`.
   c) Adicione um método que devolve a capacidade da pilha e ajuste os exemplos para funcionarem com essas modificações.
   d) Adicione um método com tempo de execução constante para esvaziar a pilha.

Pilha - Praticando

10) Projete uma função que receba como parâmetro uma pilha e remova todos os elementos da pilha que sejam vazios. Faça a implementação usando uma pilha auxiliar. Qual a complexidade de tempo da função?

```
>>> p = Pilha()
>>> p.empilha('um')
>>> p.empilha('')
>>> p.empilha('carro')
>>> p.empilha('')
>>> p.empilha('')
>>> remove_vazios(p)
>>> p.desempilha()
'carro'
>>> p.desempilha()
'um'
```

11) Projete uma função que receba como parâmetro uma pilha e modifique a pilha invertendo a ordem dos seus elementos. Faça a implementação usando um pilhas auxiliares. Qual a complexidade de tempo da função?

```
>>> p = Pilha()
>>> p.empilha('banana')
>>> p.empilha('arco')
>>> p.empilha('mouse')
>>> inverte_pilha(p)
>>> p.desempilha()
'banana'
>>> p.desempilha()
'arco'
>>> p.desempilha()
'mouse'
```

Pilha - Avançando

12) A notação que estamos acostumados a escrever expressões aritméticas é chamada de notação infixa, isso porque os operadores ficam entre os operandos, como em `3 + 5 * 6`. Também podemos utilizar a notação posfixa, onde os operadores aparecem depois dos operandos. Na notação posfixa a expressão anterior é escrita como `5 6 * 3 +`. Dois aspectos são interessantes nessa notação: os parênteses não são necessários e o algoritmo de avaliação da expressão é mais simples. O seguinte algoritmo pode ser usado para avaliar expressões na notação posfixa: analise a expressão da esquerda para a direita, se o valor analisado for um operando, empilhe em uma pilha, se o valor for um operador, desempilhe dois valores da pilha, aplique o operador, e empilhe o resultado na pilha. Projete uma função que avalie uma expressão na notação posfixa (considere que a entrada é uma lista de strings onde cada string representa um número ou um dos quadro operadores aritméticos básicos, considere também que a entrada é válida, isto é, representa uma expressão válida).

```
>>> avalia_posfixa(['102'])
102
>>> avalia_posfixa(['55', '5', '/'])
11
>>> avalia_posfixa(['5', '6', '*', '3', '+'])
33
>>> avalia_posfixa(['5', '-6', '*', '3', '+', '10', '-'])
-37
```

Fila - Começando

13) O que é uma Fila?

14) Projete uma função que receba como parâmetro uma fila e modifique a fila invertendo a ordem dos seus elementos. Faça a implementação usando uma pilha auxiliar. Qual a complexidade de tempo da função?

15) Faça as seguintes alterações na implementação de `Fila` do aquivo `fila_arranjo_circular.py`:

   a) Modifique o construtor para que ele receba como parâmetro a quantidade máxima de elementos que a fila pode armazenar.

   b) Adicione um método que devolve a quantidade de elementos na fila. Altere a implementação do método `cheia` para para usar esse novo método. A implementação ficou mais simples?

   c) Adicione um método que devolve a capacidade da fila e ajuste os exemplos para funcionarem com essas modificações.

   d) A implementação usa a ideia de "índice circular", quando o índice chega no final no arranjo, ele volta para o início. Essa ideia é usada nos métodos `enfileira`, `desenfileira` e `cheia`. Crie um método auxiliar para calcular o próximo índice a partir de um índice qualquer e use esse método para simplificar a implementação dos métodos `enfileira`, `desenfileira` e `cheia`.

   e) Adicione um método com tempo de execução constante para esvaziar a fila.

16) O programa `tempo_fila.py` (que usa alguns construções que ainda não vimos, mas não se preocupe com isso), mede o tempo de execução da função `operacoes` em segundos para $n = 1000, 2000, 4000$, tanto para fila implementada com o marcador de início, quanto para a fila implementa com o marcador de início e fim.

   Modifique o arquivo `tempo_fila.py` e faça a implementação da função `operacoes`. Em seguida, execute o arquivo com o comando `python tempo_fila.py`.

   Analise os tempos de execução obtidos e explique a diferença.

Filas - Praticando

17) Implemente o TAD Fila usando duas pilhas. Qual é a complexidade de tempo das operações?

18) Implemente o TAD Pilha usando duas filas. Qual é a complexidade de tempo das operações?

Filas - Avançando

19) Defina um TAD para fila dupla com métodos para inserir e remover da esquerda e direita. Implemente o TAD usando a estratégia de arranjo circular.

20) Modifique o programa `tempo_fila.py` para mostrar a diferença do tempo de execução do método `popleft` da classe `collections.deque` e do método `pop(0)` da classe `list` (pré-definidos em Python). Execute o programa, analise os tempos de execução obtidos e explique a diferença.

Listas - Começando

21) O que é uma Lista?

22) Como os arranjos dinâmicos podem ser implementados usando arranjos estáticos?

23) Altere a implementação de `Lista` do arquivo `lista_arranjo.py` e adicione um método `acrescenta` que acrescenta um elemento no final da lista.

24) Altere a implementação de `Lista` do arquivo `lista_arranjo.py` e adicione um método `eq` que verifica se duas listas são iguais.

25) Altere a seguinte função para usar uma `Lista` do arquivo `lista_arranjo.py` ao invés do `list` do Python.

```python
def primos(lim: int) -> list[int]:
    '''
    Encontra todos os números primos menores que *lim*.
```

```
    Exemplos:
    >>> primos(2)
    []
    >>> primos(20)
    [2, 3, 5, 7, 11, 13, 17, 19]
    '''
    primos: list[int] = []
    n = 2
    while n < lim:
        eh_primo = True
        i = 0
        while eh_primo and i < len(primos):
            if n % primos[i] == 0:
                eh_primo = False
            i = i + 1

        if eh_primo:
            primos.append(n)

        n = n + 1
    return primos
```

Listas - Praticando

26) Projete uma função que receba como parâmetro uma Lista e modifique a lista removendo todos as ocorrências de elementos iguais consecutivos. Por exemplo, após executar a função para a lista `[4, 4, 4, 1, 2, 2]` ele é alterada para `[4, 1, 2]`. Qual é a complexidade de tempo da sua função?

27) Altere a implementação do método `lista.remove` para que `valores` nunca fique com menos que 25% da sua capacidade utilizada (exceto quanto a capacidade for menor ou igual a 10). Como isso afeta a complexidade de tempo? Dica: veja o método `__cresce` e seu uso em `insere`. Escreve um método auxiliar `__diminui`, que reduz a capacidade de `valores` pela metade.

Listas - Avançando

28) Implemente o TAD Lista usando um arranjo circular de tamanho fixo. Nos métodos de inserção e remoção por índice, mova os elementos para o extremos mais próximo. Que vantagens essa implementação tem em relação a implementação de Lista do arquivo `lista_arranjo.py`.

29) Modifique a implementação anterior para usar um arranjo dinâmico.

Desafios

30) Projete uma função que avalie uma expressão aritmética na notação infixa. Para isso, implemente o algoritmo Shunting yard, que converte uma expressão na notação infixa para a notação posfixa. Depois de converter a expressão, avalie ela usando a função `avalia_posfixa`.

31) Faça uma implementação alternativa do TAD Pilha que use uma única string para armazenar todos os elementos da pilha. Qual é a complexidade de tempo das operações?

32) Projete uma função que receba como parâmetro um fila e conte a quantidade de elementos da fila. A sua implementação não deve criar uma coleção (lista, pilha, fila, etc) auxiliar e deve deixar a fila como ela foi dada na entrada.

33) Os programas em Python precisam estar corretamente indentados, senão, um erro é gerado antes da execução. A indentação de cada linha de código é determinada pela quantidade de espaços em branco no início da linha. Em geral, a quantidade de espaços é múltipla de 4, mas pode ser qualquer

quantidade. Um programa em Python está bem indentado se todos os blocos estão bem indentados. Um bloco está bem indentado se todas as instruções do bloco tem a indentação. Um novo bloco inicia após o término de um alinha com `:` e termina implicitamente com o recuo da indentação. Considere o segui trecho de código

```
1  def media_positivos(lst: list[int]) -> float:
2      soma = 0
3      num = 0
4      for x in lst:
5          if x > 0:
6              num += 1
7             soma += x
8      if num == 0:
9        media = 0.0
10     else:
11        media = soma / num
12     return soma
```

As linhas 1, 4, 5, 8 e 10, terminam com dois ponto, o que indica o ínicio de um bloco. As intruções das linhas 2, 3, 4, 8, 10 e 11 fazem parte do bloco da linha 1 pois têm a mesma identação. A instrunção da linha 3 faz parte do bloco da linha 4. A instrução da linha 9, faz parte do bloco da linha 8. A instrução da linha 11 faz parte do bloco da linha 1. Existe um problema de identação no bloco da linha 5. O problema está na identação da linha 7. Se a linha 7 divesse a mesma identação da linha 6 (seja aumentando a indentação da linha 7 ou diminuindo a indentação da linha 6), então ela faria parte do bloco da linha 5. Se a linha 7 tivesse a mesma identação da linha 5, então ela faria parte do bloco da linha 4. Com a identação atual, a linha 7 fica fora desses blocos e por isso o código está mal indentado.

Projete uma função que verifique se um código Python está bem indentado. Use uma lista de linhas (strings) como entrada. Assuma que o código não tenha comentários.