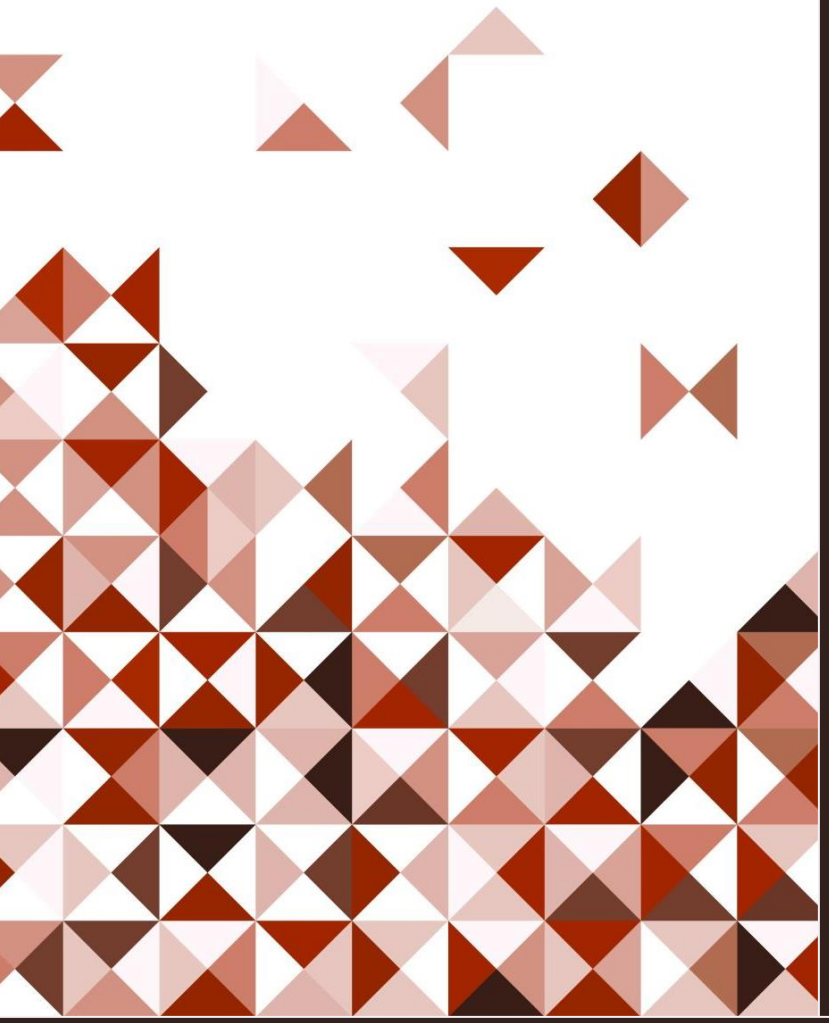


Estrutura de Dados Abstratas

Pilhas / Filas / Listas

Estrutura de Dados – 2021.2

Objetivos

- Entender os tipos de dados abstratos pilha, fila, deque e lista.
- Ser capaz de implementar a pilha, fila, lista e deque usando Python.
- Compreender o desempenho das implementações de estruturas de dados lineares básicas.
- Entender os formatos de expressão de prefixo, infixo e pós-fixo.
- Ser capaz de reconhecer as propriedades do problema onde pilhas, filas e deques são estruturas de dados apropriadas.
- Ser capaz de comparar o desempenho de nossa implementação de lista vinculada com a implementação de lista do Python.

O que são estruturas lineares?

- Pilhas, filas, deques e listas são exemplos de coleções de dados cujos itens são ordenados dependendo de como são adicionados ou removidos.
- Depois que um item é adicionado, ele permanece nessa posição em relação aos outros elementos que vieram antes e depois dele. Coleções como essas são frequentemente chamadas de estruturas de dados lineares.
- As estruturas lineares podem ser consideradas como tendo duas extremidades. Às vezes, essas extremidades são chamadas de "esquerda" e "direita" ou, em alguns casos, "frontal" e "traseira". Você também pode chamá-los de "superior" e "inferior". Os nomes dados às extremidades não são significativos.

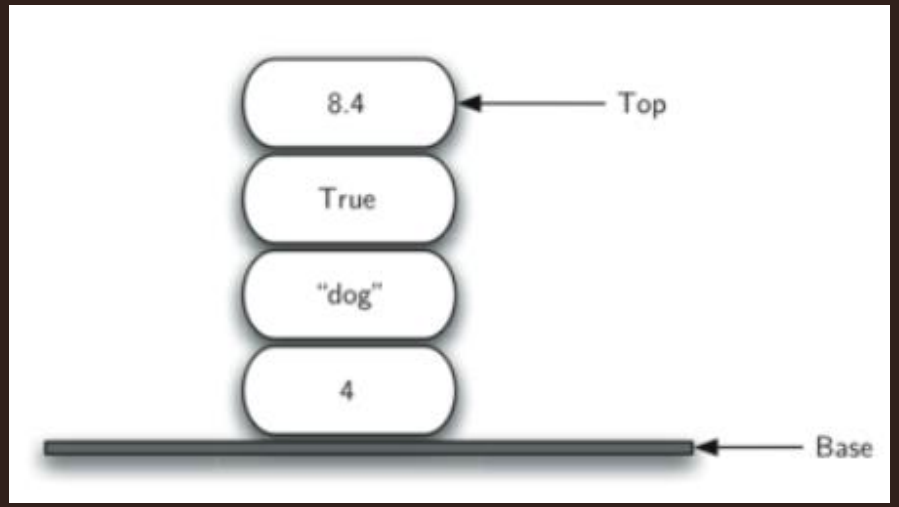
- O que distingue uma estrutura linear de outra é a forma como os itens são adicionados e removidos, em particular o local onde ocorrem essas adições e remoções.
- Essas variações dão origem a algumas das estruturas de dados mais úteis na ciência da computação. Eles aparecem em muitos algoritmos e podem ser usados para resolver uma variedade de problemas importantes.

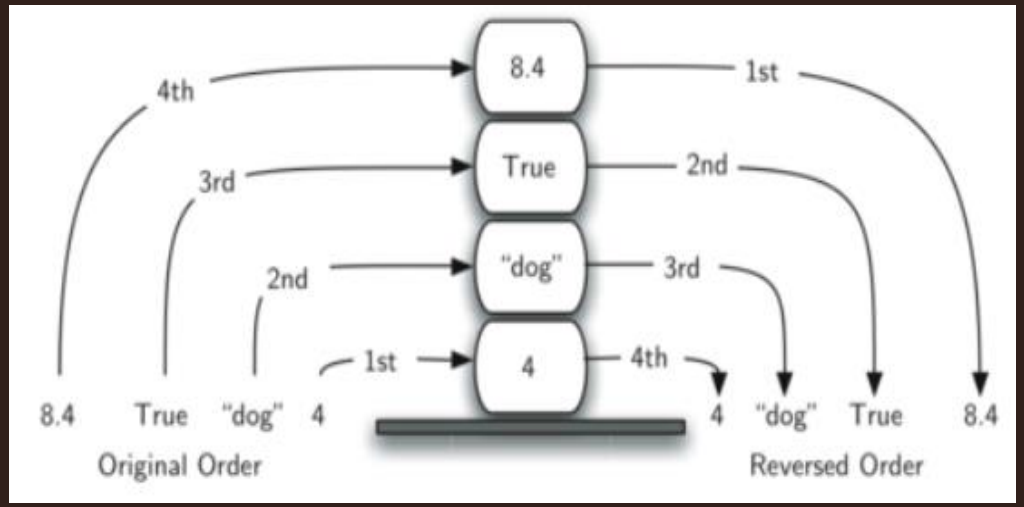
Pilhas
(Stacks)

O que é uma pilha?

- Uma pilha (às vezes chamada de “pilha push-down”) é uma coleção ordenada de itens em que a adição de novos itens e a remoção de itens existentes sempre ocorrem na mesma extremidade. Essa extremidade é comumente chamada de "topo". A extremidade oposta ao topo é conhecida como "base".
- A base da pilha é significativa, pois os itens armazenados na pilha que estão mais próximos da base representam aqueles que estão na pilha há mais tempo. O item adicionado mais recentemente é aquele que pode ser removido primeiro. Esse princípio de ordenação às vezes é chamado de LIFO (last-in/first-out). Ele fornece uma ordenação com base no período de tempo na coleção. Os itens mais novos estão perto do topo, enquanto os itens mais antigos estão perto da base.

Pilhas


8.4
Top
True
"dog"
4
Base

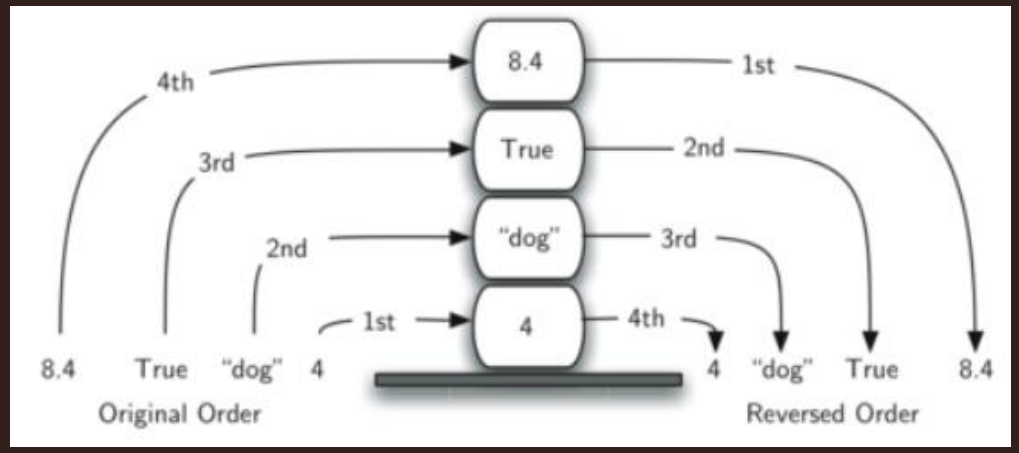

8.4
1st
4th
True
2nd
3rd
"dog"
3rd
2nd
4
4th
1st
8.4
True
"dog"
4
Original Order
4
"dog"
True
8.4
Reversed Order

Pilhas – tipo abstrato de dado

- O tipo de dados abstratos **pilha** é definido pela seguinte estrutura e operações.
  - Uma pilha é estruturada, como uma coleção ordenada de itens onde os itens são adicionados e removidos da extremidade chamada de "topo". As pilhas são ordenadas LIFO. As operações de pilha são fornecidas abaixo:
  - **Stack()** cria uma nova pilha vazia. Não precisa de parâmetros e retorna uma pilha vazia.
  - **push(item)** adiciona um novo item ao topo da pilha. Ele precisa do item e não retorna nada.
  - **pop()** remove o item superior da pilha. Não precisa de parâmetros e retorna o item. A pilha é modificada.
  - **peek()** retorna o primeiro item da pilha, mas não o remove. Não precisa de parâmetros. A pilha não é modificada.
  - **is_empty()** testa se a pilha está vazia. Não precisa de parâmetros e retorna um valor booleano.
  - **size()** retorna o número de itens na pilha. Não precisa de parâmetros e retorna um inteiro.

Pilha Exemplo

Stack Operation	Stack Contents	Return Value
s.is_empty()	[]	True
s.push(4)	[4]	
s.push('dog')	[4,'dog']	
s.peek()	[4,'dog']	'dog'
s.push(True)	[4,'dog',True]	
s.size()	[4,'dog',True]	3
s.is_empty()	[4,'dog',True]	False
s.push(8.4)	[4,'dog',True,8.4]	
s.pop()	[4,'dog',True]	8.4
s.pop()	[4,'dog']	True
s.size()	[4,'dog']	2



Implementando uma Pilha em Python

- Atenção para o uso de Python para implementar a pilha.
- Lembre-se de que, quando damos a um tipo de dado abstrato uma implementação física, nos referimos à implementação como uma estrutura de dados.
- Como em qualquer linguagem de programação orientada a objetos, a implementação de escolha para um tipo de dados abstrato, como uma pilha, é a criação de uma nova classe. As operações de pilha são implementadas como métodos.
- Além disso, para implementar uma pilha, que é uma coleção de elementos, faz sentido utilizar o poder e a simplicidade das coleções primitivas fornecidas pelo Python, sendo assim, usaremos uma lista.

- Lembre-se de que a classe de lista em Python fornece um mecanismo de coleta ordenado e um conjunto de métodos. Por exemplo, se temos a lista [2, 5, 3, 6, 7, 4], precisamos apenas decidir qual extremidade da lista será considerada o topo da pilha e qual será a base. Assim que essa decisão for tomada, as operações podem ser implementadas usando os métodos de lista, como append e pop.
- A implementação da pilha assume que o final da lista conterá o elemento superior da pilha. Conforme a pilha cresce (conforme ocorrem as operações push), novos itens são adicionados ao final da lista. As operações pop irão manipular a pilha até chegar a base.

```python
# Completed implementation of a stack ADT
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items) - 1]

    def size(self):
        return len(self.items)
```

```python
s = Stack()

print(s.is_empty())
s.push(4)
s.push('dog')
print(s.peek())
s.push(True)
print(s.size())
print(s.is_empty())
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
```

- É importante notar que poderíamos ter escolhido implementar a pilha usando uma lista em que o topo está no início em vez de no final.
- Nesse caso, os métodos pop e append anteriores não funcionariam mais e teríamos que indexar a posição 0 (o primeiro item da lista) explicitamente usando pop e insert. Veja a implementação.

```python
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, item):
        self.items.insert(0, item)

    def pop(self):
        return self.items.pop(0)

    def peek(self):
        return self.items[0]

    def size(self):
        return len(self.items)


s = Stack()
s.push('hello')
s.push('true')
print(s.pop())
```

- Essa capacidade de alterar a implementação física de um tipo de dado abstrato enquanto mantém as características lógicas é um exemplo de abstração em funcionamento.
- No entanto, embora a pilha funcione de qualquer maneira, se considerarmos o desempenho das duas implementações, definitivamente há uma diferença.
- Lembre-se de que as operações append() e pop() foram ambas $O(1)$. Isso significa que a primeira implementação executará push e pop em tempo constante, não importa quantos itens estejam na pilha.
- O desempenho da segunda implementação sofre porque as operações insert(0) e pop(0) requerem $O(n)$ para uma pilha de tamanho n.
- Claramente, mesmo que as implementações sejam logicamente equivalentes, elas teriam tempos muito diferentes ao realizar o teste de benchmark.

Problema

- **Verificação de parênteses, colchetes e chaves**
- Agora voltamos nossa atenção para o uso de pilhas para resolver problemas reais de ciência da computação. Você tem sem dúvida, expressões aritméticas escritas como (5 + 6) * (7 + 8) / (4 + 3) onde parênteses são usados para ordenar o desempenho das operações (**ver código de verificação geral**).

- **Exercício:** Criar um conversor de bases numéricas (decimal para binário)

Problema

- **Expressões Infixa, Prefixa e Posfixa**
- Quando você escreve uma expressão aritmética como $B * C$, a forma da expressão fornece informações para que você possa interpretá-la corretamente. Nesse caso, sabemos que a variável B está sendo multiplicada pela variável C uma vez que o operador de multiplicação * aparece entre eles na expressão. Esse tipo de notação é conhecido como **infixa**, pois o operador está entre os dois operandos nos quais está trabalhando.

- No entanto: $A + B * C$ seria escrito como + $A * BC$ no prefixa. O operador de multiplicação vem imediatamente antes dos operandos B e C, denotando que * tem precedência sobre +. O operador de adição então aparece antes do A e do resultado da multiplicação.
- No posfixa, a expressão seria $ABC * +$. Novamente, a ordem das operações é preservada, uma vez que * aparece imediatamente após B e C, denotando que * tem precedência, com + vindo depois.
- Embora os operadores tenham se movido e agora apareçam antes ou depois de seus respectivos operandos, a ordem dos operandos permaneceu exatamente a mesma em relação ao outro.

Infix Expression	Prefix Expression	Postfix Expression
$A + B$	$+AB$	$AB+$
$A + B * C$	$+A * BC$	$ABC * +$

Infix Expression	Prefix Expression	Postfix Expression
$(A + B) * C$	$* + ABC$	$AB + C*$

- Ver código conversão de infixa para posfixa

- **Avaliação Posfixa**
- Como um exemplo de pilha final, consideraremos a avaliação de uma expressão que já está em notação pós-fixada. Nesse caso, uma pilha é novamente a estrutura de dados escolhida.
- No entanto, conforme você varre a expressão pós-fixada, são os operandos que devem esperar, não os operadores como no algoritmo de conversão. Outra forma de pensar na solução é que sempre que um operador for visto na entrada, os dois operandos mais recentes serão usados na avaliação.

- Para ver isso com mais detalhes, considere a expressão pós-fixada 456 * +. Ao varrer a expressão da esquerda para a direita, você encontra primeiro os operandos 4 e 5. Nesse ponto, você ainda não tem certeza do que fazer com eles até ver o próximo símbolo. Colocar cada um na pilha garante que estejam disponíveis se um operador vier em seguida.
- Nesse caso, o próximo símbolo é outro operando. Então, como antes, pressione-o e verifique o próximo símbolo. Agora vemos um operador, *. Isso significa que os dois operandos mais recentes precisam ser usados em uma operação de multiplicação. Ao estourar a pilha duas vezes, podemos obter os operandos adequados e, em seguida, realizar a multiplicação (neste caso, obtendo o resultado 30).

- Agora podemos lidar com esse resultado colocando-o de volta na pilha para que possa ser usado como um operando para os operadores posteriores na expressão. Quando o operador final for processado, haverá apenas um valor restante na pilha. Pop e retorne como resultado da expressão.
- (Ver código de avaliação pós-fixa)