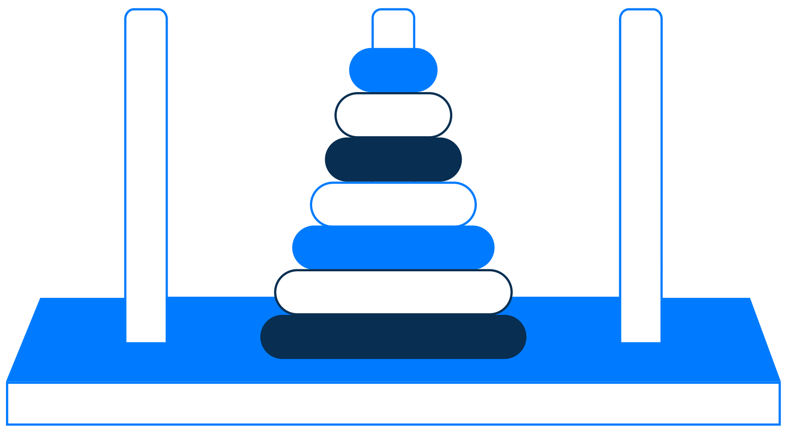
Theory: Stack in Python

32 minutes 8 / 8 problems solved

Start practicing

1725 users solved this topic. Latest completion was about 13 hours ago.

As you already know, **a stack** is a data structure that stores data in the last-in-first-out (LIFO) fashion, meaning that the last element added to the stack will be the first removed from it. The simplest example of a stack in real life is a stack of plates: the ones you put the last are on top, and they are also the first ones that can be removed. Another illustration of a stack is the Tower of Hanoi, a toy that became the inspiration for a famous game. The rings on the very top of it are the last being put, but also the first that can be taken away:



Stack implements two basic operations: **push**, that adds an element to the stack, and **pop**, that returns its top element and removes it from the stack. Often, the third operation, **peek**, is also supported. It returns the current top element without removing it from the stack.

In Python, there is no conventional stack implementation. However, several data structures can be used as one. In this topic, you will learn about some of them.

§1. Using lists as stack

Built-in Python lists implement all the necessary stack functionality, and therefore can be used as such. Indeed, `append()` adds the elements to the list (the analog of the push operator), while `pop()` removes its last element. Note that the top of the stack is actually at the end of the list. Let's take a look at the example:

Current topic:

✓ Stack in Python

Topic depends on:

✓ Stack

✓ Operations with list Stage 3 6★

✓ Load module Stage 1 5★

✓ Algorithms in Python

```
my_stack = []

my_stack.append('Out')
my_stack.append('First')
my_stack.append('In')
my_stack.append('Last')

print(my_stack)

# ['Out', 'First', 'In', 'Last']

print(my_stack.pop())

# Last

print(my_stack.pop())

# In

print(my_stack.pop())

# First

print(my_stack.pop())

# Out
```

Nice, isn't it? Well, despite the simplicity of such an approach, using list as a stack isn't a good idea because it can lead to some memory issues. Due to the way lists are implemented in the Python source code, the time complexity of some operations may not be that consistent. When the list grows bigger, Python might need to find a large block of memory to relocate it to, which can slow down some `append()` calls.

§2. Using collections.deque() as a stack

The problem described above can be avoided by using **deque**.

Deque is a linear data structure that supports adding and removing elements from both sides. In Python, you can find deque implementation in the `collections` module. The essential methods are `append(element)` and `appendleft(element)`, which add a new element to the right or left side of the deque respectively, as well as `pop()` and `popleft()`, which remove the first element from the corresponding side of the deque.

Since deque is implemented as a doubly linked list, its elements don't have to be stored next to each other in memory. By contrast, only smaller chunks of elements are stored together, and each such chunk stores the references to the previous and to the next chunk. This enables faster append operations.

You can use deque as a stack in your program. Let's take a look at the example:

```
from collections import deque

my_stack = deque()

my_stack.append('k')
my_stack.append('c')
my_stack.append('a')
my_stack.append('t')
my_stack.append('s')



print(my_stack.pop())



# 's'



print(my_stack.pop())



# 't'



print(my_stack.pop())



# 'a'



print(my_stack.pop())



# 'c'



print(my_stack.pop())



# 'k'


```

Note that if you call `pop()` one more time, you will get an exception because there are no more elements in the stack to be popped:

```
my_stack.pop()

# IndexError: pop from an empty deque
```

To avoid this, you can always check if the queue is non-empty before trying to get the next element from it. This can be done with the help of `len()`:

```
len(my_stack)

# 0

my_stack.append('a')
len(my_stack)

# 1
```

§3. Conclusions

- Stack is a LIFO data structure.
- You can use Python lists as stack, but it might lead to memory issues.

- `collections.deque()` implements deque as a doubly-linked list.
- Using deque as a stack in your program enables faster append operations.

Report a typo

Feedback sent!

Start practicing

Comments (5) Hints (0) Useful links (0) Show discussion