CISS433: Python Programming
Lecture 20: Stacks, Queues, Deques

Yihsiang Liow

Agenda

- Study stacks and queues.
- Study deques.

Stacks and Queues

- ***Container***: models something that is used to contain other things.
  - Example: C/C++ arrays and Python lists are containers. Linked lists are also containers.

Stacks and Queues

- ***Stack***: a container which is last-in-first-out (LIFO), i.e., the last item that went in is the first to come out
  - If you put 3 into a stack, then 5, then 1, on retrieving from the stack you get 1, then 5, then 3
  - ***Push***: put something into a stack
  - ***Pop***: get something from a stack
  - ***Top***: The last item (or a reference to it) that is pushed onto a stack
  - Visual: Think of a stack of plates at the buffet

Stacks and Queues

- ***Queue***: a container which is first-in-first-out (FIFO), i.e., the first item that went in is the first to come out
  - If you put 3 into a queue, then 5, then 1, on retrieving from the stack you get 3, then 5, then 1
  - ***Enqueue***: putting something into a queue
  - ***Dequeue***: getting something from a queue
  - ***Head***: Refers to item first enqueued
  - ***Tail***: Refers to item last enqueued
  - Head and tail are sometimes called front and back
  - Visual: Think of a line of people at a cashier

Stacks and Queues

- If a container has fixed capacity and you try to put in something, you get an ***overflow*** condition
- If a container is empty and you try to remove something from it, you get an ***underflow*** condition
- Besides putting things into the container, it's useful to have the following features:
  - `empty`: function that tells you if container is empty
  - `full`: a function that tells you if container is full (if the container has a fixed capacity)
  - `length`: number of items in the container

Stacks and Queues

- Note that it is a common practice for boolean functions to begin with `is`.
  - Example: instead of `empty()`, you might want to call the function `isempty()`
- You might need an initialization function.

Abstract Data Types

- ADT = Abstract data types: a data type with operations but without details on implementation
- Stacks, queues are examples of ADT

Realization

- Note that for stacks and queues, the ordering of things you put in is important. Furthermore, there is a “before-after” relationship between items in the stack and queue – they are linear.
- So you can implement stacks and queues using either lists or linked lists

Non-OO Stack Using List

- Python list is already a stack.
  - Lists come with the pop method
  - There is no push, but you can use ...
  - To check if the list is empty ...
  - Python list do not have fixed length, so don't worry about overflow (Unless if you run out of memory which is an ***exception***. See notes on Exceptions later.)
- So ...

Non-OO Stack Using List

```
def empty(stack):
    return len(stack)==0

def full(stack):
    return False

def length(stack):
    return len(stack)

def push(stack,x):
    stack.append(x)

def pop(stack):
    if empty(stack):
        return None
    else:
        return stack.pop()
```

```
if __name__ == "__main__":
    stack = []
    print stack
    push(stack, 3)
    print stack
    push(stack, 5)
    print stack
    push(stack, 1)
    print stack
    print pop(stack), stack
    print pop(stack), stack
    print pop(stack), stack
    print pop(stack), stack
```

OO Stack Using List

```
class Stack:

    def __init__(self):
        self.__list = []

    def empty(self):
        return len(self.__list)==0

    def full(self):
        return False

    def __len__(self):
        return len(self.__list)

    def push(self,x):
        self.__list.append(x)

    def pop(self):
        if self.empty():
            return None
        else:
            return \
  self.__list.pop()
```

```
    def __repr__(self):
        return "%s (top)" % \
                self.__list


if __name__ == "__main__":
    stack = Stack()
    print stack
    stack.push(3)
    stack.push(5)
    stack.push(1)
    print stack
    print stack.pop(),stack
    print stack.pop(),stack
    print stack.pop(),stack
    print stack
```

Linked List Implementation

- Obviously you can also implement stacks and queues using linked lists
- In particular:
  - Stack: You need insert at tail and remove from tail
  - Queue: You need insert at tail and remove from head

Priority Queues

- ***Priority queue***: Same as a queue except that objects being inserted into the data structure comes with a priority number. High-priority objects are closer to the head of the queue
- Example: You have a queue of x0,x1,x2,x3 with priorities 1,2,2,3. If you insert object y with priority 2, then queue becomes:

	(head)				(tail)
object	x0	x1	x2	**y**	x3
priority	1	2	2	**2**	3

- Implementation: usually heaps

Deque

- ***Deque*** = ***double-ended queue***: For this queue, you can enqueue and dequeue at either the head or the tail.

Python's deque

- Python has a deque class:

```
from collections import deque
d = deque([1,2,3])
print(d, len(d))
d.append('a')
d.appendleft('b')
print(d)
d.extend(['A','B','C'])
d.extendleft(['X', 'Y', 'Z'])
print(d)
x = d.pop(); y = d.popleft()
print(x, y, d)
d.clear()
print(d)
d.pop()
```

Heaps

- Minheap: Data is organized so that smallest value is removed first.
- Maxheap: Data is organized so that largest value is removed first.
- Heaps are frequently implemented using an array.
- Python: List is used. The heap is minheap.

Heaps

```
import heapq
xs = []
heapq.heappush(xs, 5); print(xs)
heapq.heappush(xs, 2); print(xs)
heapq.heappush(xs, 0); print(xs)
heapq.heappush(xs, 3); print(xs)
heapq.heappush(xs, 1); print(xs)
x = heapq.heappop(xs); print(x, xs)
x = heapq.heappop(xs); print(x, xs)
x = heapq.heappop(xs); print(x, xs)
x = heapq.heappop(xs); print(x, xs)
x = heapq.heappop(xs); print(x, xs)
xs = [5,2,0,3,1]
heapq.heapify(xs); print(xs)
```

Heaps

So if you have a list of jobs with priority number (where low priority # means higher priority):

```
import heapq
xs = [(5, 'job1'),
      (2, 'job2'),
      (0, 'job3'),
      (3, 'job4'),
      (1, 'job5')]
heapq.heapify(xs); print xs
job = heapq.heappop(xs)
print(job)
print(xs)
```

Heaps

- What is the runtime of ...
- Heapify?
- Insert into a minheap (or maxheap)?
- Delete min from a minheap? (or max from maxheap)