Ch22-Queues

September 10, 2025

1 Queues

http://openbookproject.net/thinkcs/python/english3e/queues.html - another container adapter or ADT, which is typically a first-in-first-out (FIFO) data structure - mimics real-world queue/line of people waiting for something - rule that determines who goes next is called the queuing policy - the most general queuing policy is priority queuing, in which each item/person is assigned a priority and the element with highest priority goes first, regardless of the order of the arrival - built-in alternative - deque: https://docs.python.org/3/library/collections.html#collections.deque

1.1 The Queue ADT

- can be implemented using built-in list or linked list as the container to hold the elements
- queque ADT is defined by the following operations:
  - ___init___ : initialize the queue
  - insert : add a new item to the queue
  - remove : remove and return the first element that was added
  - is_empty : check whether the queue is empty

1.2 Linked Queue

[1]:
```
class Node:
    def __init__(self, data):
        self.cargo = data
        self.next = None

    def __str__(self):
        return "{}".format(self.cargo)
```

[4]:
```
class Queue:
    def __init__(self):
        self.length = 0
        self.head = None
        self.tail = None

    def is_empty(self):
        return self.length == 0
```

```
    def insert(self, data):
        node = Node(data)
        if not self.head: # empty queue
            self.head = self.tail = node
        else:
            # add new node as the last node
            self.tail.next = node
            self.tail = node
        self.length += 1

    def remove(self):
        data = self.head.cargo
        # make the head point to 2nd element
        self.head = self.head.next
        self.length -= 1
        # update tail if the queue becomes empty after removing the first node
        if self.length == 0:
            self.tail = None

        return data

    def __len__(self):
        return self.length
```

1.2.1 quick test of Queue ADT

[5]:
```
q = Queue()
q.insert(1)
q.insert(2)
q.insert('a')
assert q.remove() == 1
assert len(q) == 2
q.insert(100)
assert q.remove() == 2
```

1.3 Priority Queue ADT

- better built-in alternative: heapq- https://docs.python.org/3/library/heapq.html
- Priority Queue ADT implementation
- use the same methods/interfaces as Queue with only difference in remove function; where the item removed is the highest priority
  - item removed is not necessarily the first one that was added; rather an item in the queue with highest priority
  - e.g., if the items in the queue have names, we might choose item in alphabetical order
  - if they're bowling scores, we might go from highest to lowest, but if they're golf scores, we would go from lowest to highest

[6]:
```
class PriorityQueue:
    def __init__(self):
        self.items = []

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

    def insert(self, data):
        self.items.append(data)

    def remove(self):
        maxi = 0
        for i in range(1, len(self.items)):
            if self.items[i] > self.items[maxi]:
                maxi = i
        item = self.items[maxi]
        del self.items[maxi]
        return item

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

1.3.1 test priority queue ADT

[7]:
```
q = PriorityQueue()
for num in [11, 12, 14, 13]:
    q.insert(num)
```

[8]:
```
while not q.is_empty():
    print(q.remove())
```

```
14
13
12
11
```

1.4 The Golfer class

- keeps track of the names and scores of golfers

[10]:
```
class Golfer:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __str__(self):
        return "{0:16}: {1}".format(self.name, self.score)
```

```
    # lowest score gets highest priority
    def __gt__(self, other):
        return self.score < other.score # lower score has the higher priority
```

[11]:
```
tiger = Golfer('Tigher Woods', 40)
phil = Golfer('Phil Mickelson', 30)
hal = Golfer('Hal Sutton', 20)
pq = PriorityQueue()
pq.insert(tiger)
pq.insert(phil)
pq.insert(hal)
print(pq.remove())
```

```
Hal Sutton      : 20
```

[12]:
```
while not pq.is_empty():
    print(pq.remove())
```

```
Phil Mickelson  : 30
Tigher Woods    : 40
```

1.5 exercises

1. Write an implementation of the Queue ADT using Python list. Compare the performance of this implementation to the ImprovedQueue for a range of queue lengths.

- Write an implementation of the Priority Queue ADT using linked list. You should keep the list sorted so that removal is a constant time operation. Compare the performance of this implementation with the Python list implementation.

1.6 Kattis problems that can be solved using Queue

1. Solve kattis problem Bank Queue: https://open.kattis.com/problems/bank

- Server - https://open.kattis.com/problems/server
- Ferry Loading III - https://open.kattis.com/problems/ferryloading3
- Foosball Dynasty - https://open.kattis.com/problems/foosball
- Disastrous Downtime: https://open.kattis.com/problems/downtime

[ ]: