**python优先队列和堆的使用**

**0 前言**

优先队列和堆(大顶堆和小顶堆)经常被用来实现查找数组中最大或者最小的K个元素。之前一直使用C++对于C++的优先队列和堆比较熟悉，python的不太熟悉，这里进行记录。

**1 优先队列**

首先来看优先队列，优先队列中的元素是有优先级的，优先级高的元素排在前面。**在python的中排序关键字小的排在前面。可以理解为排序关键字给出的是一个排名。** python3队列相关的模块在queue中，python2中的优先队列模块在Queue包中。  
存储内置数据类型：

from Queue import PriorityQueue

pq = PriorityQueue()

for i in range(3,0,-1):

pq.put(i)

while not pq.empty():

print pq.get()

1 2 3

**存放元组：**  
如果存放元组，则默认比较元组的第一个元素，小的在队列头部，如果，第一元素相同比较第二个元素，如果还相同依次往后比较。其实这应该是是内置的元组大小比较函数定义的比较方式。

from queue import PriorityQueue

pq = PriorityQueue()

pq.put((1, 2))

pq.put((1, 0))

pq.put((2, 3))

while not pq.empty():

print (pq.get())

**存放自定义类型：**  
自定义数据类型，需要自定义\_\_cmp\_\_或者\_\_lt\_\_比价函数。

from queue import PriorityQueue

class Job(object):

def \_\_init\_\_(self, priority, description):

self.priority = priority

self.description = description

print('New job:', description)

return

def \_\_lt\_\_(self, other):

return self.priority < other.priority

''' 或者使用\_\_cmp\_\_函数

def \_\_cmp\_\_(self, other):

if self.priority < other.priority:

return -1

elif self.priority == other.priority:

return 0

else:

return 1

'''

q2 = PriorityQueue()

q2.put(Job(5, 'Mid-level job'))

q2.put(Job(10, 'Low-level job'))

q2.put(Job(1, 'Important job')) #数字越小，优先级越高

while not q2.empty():

next\_job = q2.get() #可根据优先级取序列

print('Processing job', next\_job.description)

('New job:', 'Mid-level job')

('New job:', 'Low-level job')

('New job:', 'Important job')

('Processing job', 'Important job')

('Processing job', 'Mid-level job')

('Processing job', 'Low-level job')

**2 堆**

**2.1 小顶堆**

python提供了heapq模块，提供了堆的支持，是基于python的list实现的。**heapq提供的默认是一个小顶堆的实现，即最小的元素排在前面。**  
堆（Heap）是一种特殊形式的完全二叉树，其中父节点的值总是大于子节点，根据其性质，Python 中可以用一个满足 heap[k] <= heap[2*k+1] and heap[k] <= heap[2*k+2] 的列表来实现（heapq 也确实是这么做的）  
下面介绍headp模块的使用：  
heapq模块中有6个函数：

1、heappush(heap, x)：向堆中添加元素

from heapq import \*

heap = []

for i in range(3):

heappush(heap, i)

print(heap) #[0, 1, 2]

heappush(heap, 0.5)

print(heap) #[0, 0.5, 2, 1]

heappush(heap, 1.5)

print(heap) #[0, 0.5, 2, 1, 1.5]

2、heappop(heap)：弹出堆中最小的元素，并且维持剩余元素的堆结构

from heapq import \*

heap = []

for i in range(3):

heappush(heap, i)

print(heap) #[0, 1, 2]

heappop(heap) #heappop函数会返回弹出的值

print(heap) #[1, 2]

3、heapify(heap)：将列表转换为堆

from heapq import \*

heap = [5, 8, 0, 4, 6, 7]

heapify(heap)

print(heap) #[0, 4, 5, 8, 6, 7]

4、heapreplace(heap, x)：弹出堆中最小的元素，然后将新元素插入。

from heapq import \*

heap = [5, 8, 0, 4, 6, 7]

heapify(heap)

print(heapreplace(heap, 5.5)) #0

print(heap) #[4, 5.5, 5, 8, 6, 7]

5、nlargest(n, iter)、nsmallest(n, iter)：用来寻找任何可迭代对象iter中的前n个最大的或前n个最小的元素。

from heapq import \*

lst = [5, 8, 0, 4, 6, 7]

print(nsmallest(3, lst))

print(nlargest(3, lst))

**2.2 大顶堆**

python没有提供大顶堆的实现，想要使用大顶堆需要一些trick。  
heappush(e)改为heappush(-e)，heappop(e)为-heappop(e)，也就是说存入和取出的数都是相反数，其他逻辑和TopK相同。  
按照这种思路自己封装一下就可以实现大顶堆。像优先队列一样，heapq同样支持自定义的数据类型，需要给自定义的数据类型定义\_\_cmp\_\_函数。

参考文章：

1. [python使用heapq实现小顶堆（TopK大）/大顶堆（BtmK小）](https://blog.csdn.net/tanghaiyu777/article/details/55271004)
2. [Python中的堆：heapq模块](https://blog.csdn.net/qq_34840129/article/details/80846838)
3. [python中使用优先队列](https://www.yangyanxing.com/article/priorityqueue_in_python.html)
4. [Python 的堆与优先队列 - PyTips 0x10](https://www.jianshu.com/p/5c55282b1d54)