How can I make a unique value priority queue in Python?

Asked 10 years, 3 months ago Active 3 months ago Viewed 11k times

19

4

Python has Queue.PriorityQueue, but I cannot see a way to make each value in it unique as there is no method for checking if a value already exists (like find(name) or similar). Moreover, PriorityQueue needs the priority to remain within the value, so I could not even search for my value, as I would also have to know the priority. You would use (0.5, myvalue) as value in PriorityQueue and then it would be sorted by the first element of the tuple.

The collections.deque class on the other hand does offer a function for checking if a value already exists and is even more natural in usage (without locking, but still atomic), but it does not offer a way to sort by priority.

There are some other implementations on stackoverflow with heapq, but heapq also uses priority within the value (e.g. at the first position of a tuple), so it seems not be great for comparison of already existing values.

Creating a python priority Queue

https://stackoverflow.com/questions/3306179/priority-queue-problem-in-python

What is the best way of creating a atomic priority queue (=can be used from multiple threads) with unique values?

Example what I'd like to add:

- Priority: 0.2, Value: value1
- Priority: 0.3, Value: value2
- Priority: 0.1, Value: value3 (shall be retrieved first automatically)
- Priority: 0.4, Value: value1 (shall not be added again, even though it has different priority)

python priority-queue

Share Follow

edited May 23 '17 at 12:17
Community ♦
1 ● 1

asked May 13 '11 at 20:05
aufziehvogel
6,907 ● 5 ● 32 ● 51

17

```
import heapq

class PrioritySet(object):
    def __init__(self):
        self.heap = []
        self.set = set()

    def add(self, d, pri):
        if not d in self.set:
            heapq.heappush(self.heap, (pri, d))
            self.set.add(d)

    def get(self):
        pri, d = heapq.heappop(self.heap)
        self.set.remove(d)
        return d
```

This uses the priority queue specified in one of your linked questions. I don't know if this is what you want, but it's rather easy to add a set to any kind of queue this way.

Share Follow

answered May 13 '11 at 20:29

Boaz Yaniv

5,996 ● 19 ● 27

7 I'll suggest not to use built-in function names like `self.set` – sleepsort Nov 28 '13 at 7:21

7 maybe pop is the better name for get :) – DikobrAz Aug 20 '14 at 10:10

8

Well here's one way to do it. I basically started from how they defined PriorityQueue in Queue.py and added a set into it to keep track of unique keys:

```
from Queue import PriorityQueue
import heapq

class UniquePriorityQueue(PriorityQueue):
    def _init(self, maxsize):
#        print 'init'
        PriorityQueue._init(self, maxsize)
        self.values = set()

    def _put(self, item, heappush=heapq.heappush):
#        print 'put',item
        if item[1] not in self.values:
            print 'uniq',item[1]
            self.values.add(item[1])
```

```
if __name__=='__main__':
    u = UniquePriorityQueue()

    u.put((0.2, 'foo'))
    u.put((0.3, 'bar'))
    u.put((0.1, 'baz'))
    u.put((0.4, 'foo'))

    while not u.empty():
        item = u.get_nowait()
        print item
```

Boaz Yaniv beat me to the punch by a few minutes, but I figured I'd post mine too as it supports the full interface of PriorityQueue. I left some print statements uncommented, but commented out the ones I put in while debugging it. ;)

Share Follow

answered May 13 '11 at 20:38

John Gaines Jr.
9,966 ● 1 ● 23 ● 25

Thanks for your answer. Didn't know the methods _init, _put and _get yet, but they are really practical when extending a queue. And as you both used sets, I am convinced now that those are the right way to go ;)
– aufziehvogel May 14 '11 at 15:31

2

In case you want to prioritise a task later.

```
u = UniquePriorityQueue()

u.put((0.2, 'foo'))
u.put((0.3, 'bar'))
u.put((0.1, 'baz'))
u.put((0.4, 'foo'))
# Now `foo`'s priority is increased.
u.put((0.05, 'foo'))
```

Here is another implementation follows the official guide:

```
import heapq
import Queue

class UniquePriorityQueue(Queue.Queue):
    """
```

```
        item = list(item)
        priority, task = item
        if task in self.entry_finder:
            previous_item = self.entry_finder[task]
            previous_priority, _ = previous_item
            if priority < previous_priority:
                # Remove previous item.
                previous_item[-1] = self.REMOVED
                self.entry_finder[task] = item
                heappush(self.queue, item)
            else:
                # Do not add new item.
                pass
        else:
            self.entry_finder[task] = item
            heappush(self.queue, item)

    def _qsize(self, len=len):
        return len(self.entry_finder)

    def _get(self, heappop=heapq.heappop):
        """
        The base makes sure this shouldn't be called if `_qsize` is 0.
        """
        while self.queue:
            item = heappop(self.queue)
            _, task = item
            if task is not self.REMOVED:
                del self.entry_finder[task]
                return item
        raise KeyError('It should never happen: pop from an empty priority
queue')
```

Share Follow

edited Aug 7 '17 at 22:19

Matt

25.1k ● 6 ● 76 ● 72

answered Feb 22 '17 at 12:38

colinfang

18.3k ● 14 ● 70 ● 150

0

I like @Jonny Gaines Jr.'s answer but I think it can be simplified. PriorityQueue uses a list undert he hood, so you can just define:

```
class PrioritySetQueue(PriorityQueue):
    def _put(self, item):
        if item not in self.queue:
            super(PrioritySetQueue, self)._put(item)
```

Share Follow

answered May 20 at 23:45

Peter w