pqdict Documentation

Release 1.0.0

Nezar Abdennur

August 05, 2016

Contents

1 Quickstart **3**
- 1.1 What is an “indexed” priority queue? . . . 3
- 1.2 Usage . . . 3
- 1.3 License . . . 5
- 1.4 Documentation . . . 5

2 Examples **7**

3 API Reference **9**
- 3.1 pqdict class . . . 9
- 3.2 Functions . . . 11

Release v1.0.

CHAPTER 1

Quickstart

1.1 What is an "indexed" priority queue?

A priority queue allows you to serve or retrieve items in a prioritized fashion. A priority queue supports inserting elements with priorities, and removing or peeking at the top priority element. The vanilla priority queue interface can be extended to support random access, insertion, removal and changing the priority of any element in the queue. An *indexed* priority queue does these latter operations efficiently.

The priority queue is implemented as a binary heap of (key, priority value) pairs, which supports:

- O(1) search for the item with highest priority
- O(log n) removal of the item with highest priority
- O(log n) insertion of a new item

An internal index maps elements to their location in the heap and is kept up to date as the heap is manipulated. As a result, pqdict also supports:

- O(1) lookup of any item by key
- O(log n) removal of any item
- O(log n) updating of any item's priority level

Indexed priority queues are useful in applications where priorities of already enqueued items may frequently change (e.g., schedulers, optimization algorithms, simulations, etc.).

1.2 Usage

The default heap is a min-heap, meaning **smaller** values give an item **higher** priority.

```
>>> from pqdict import pqdict
>>> pq = pqdict({'a':3, 'b':5, 'c':8})
>>> list(pq.popkeys())
['a', 'b', 'c']
```

To create a max-heap instead (**larger** values give **higher** priority), pass in the option `reverse=True`.

```
>>> from pqdict import pqdict
>>> pq = pqdict({'a':3, 'b':5, 'c':8}, reverse=True)
>>> list(pq.popkeys())
['c', 'b', 'a']
```

Alternatively, you can use the alias functions `minpq()` and `maxpq()`.

```
>>> from pqdict import minpq
>>> pq = minpq(a=3, b=5, c=8)
```

By default, items are ordered by **value**. Analogous to the built-in `sorted()`, you can provide a **priority key function** to transform the values for sorting.

```
>>> from pqdict import pqdict
>>> pq = pqdict({'a':(2,3), 'b':(8,5), 'c':(10,8)}, key=lambda x: x[1])
```

Views and regular iteration don't affect the heap, but the output is **unsorted**. This applies to `pq.keys()`, `pq.values()`, `pq.items()` and using `iter()` (e.g., in a for loop).

```
queue = pqdict({'Alice':1, 'Bob':2})
for customer in queue:
    serve(customer) # Bob may be served before Alice!
```

```
>>> list(pqdict({'a': 1, 'b': 2, 'c': 3, 'd': 4}).keys())
['a', 'c', 'b', 'd']
```

"Heapsort iterators" output data in **descending order** of priority by removing items from the collection. The following methods return heapsort iterators: `pq.popkeys()`, `pq.popvalues()`, and `pq.popitems()`.

```
for customer in queue.popkeys():
    serve(customer) # Customer satisfaction guaranteed :)
# queue is now empty
```

`pqdict` supports all Python dictionary methods...

```
>>> from pqdict import pqdict
>>> pq = pqdict({'a':3, 'b':5, 'c':8})
>>> pq['d'] = 6.5
>>> pq['e'] = 2
>>> pq['f'] = -5
>>> 'f' in pq
True
>>> pq['f']
-5
>>> pq.pop('f')
-5
>>> 'f' in pq
False
>>> del pq['e']
>>> pq.get('e', None)
None
```

...and exposes a priority queue API.

```
>>> pq.top()
'c'
>>> pq.topitem()
('c', 1)
# manual heapsort...
>>> pq.pop()  # no args
'c'
>>> pq.popitem()
('a', 3)
>>> pq.popitem()
('b', 5)
```

```
>>> pq.popitem()
('d', 6.5)
>>> pq.popitem()  # ...and we're empty!
KeyError
```

Note: **Value mutability**. If you use mutable objects as values in a `pqdict`, changes to the state of those objects can break the priority queue. If this does happen, the data structure can be repaired by calling `pq.heapify()`. (But you probably shouldn't be using mutable values in the first place.)

Note: **Custom precedence function**. The only difference between a min-pq and max-pq is that precedence of items is determined by comparing priority keys with the builtin < and > operators, respectively. If you would like to further customize the way items are prioritized, you can pass a boolean function `precedes(pkey1, pkey2)` to the initializer.

The module functions `nsmallest` and `nlargest` work like the same functions in `heapq` but act on mappings instead of sequences, sorting by value:

```
>>> from pqdict import nlargest
>>> billionaires = {'Bill Gates': 72.7, 'Warren Buffett': 60.0, ...}
>>> top5_names = nlargest(5, billionaires)
```

1.3 License

This module is released under the MIT license. The augmented heap implementation was adapted from the `heapq` module in the Python standard library, which was written by Kevin O'Connor and augmented by Tim Peters and Raymond Hettinger.

1.4 Documentation

Documentation is available at http://pqdict.readthedocs.org/en/latest/.

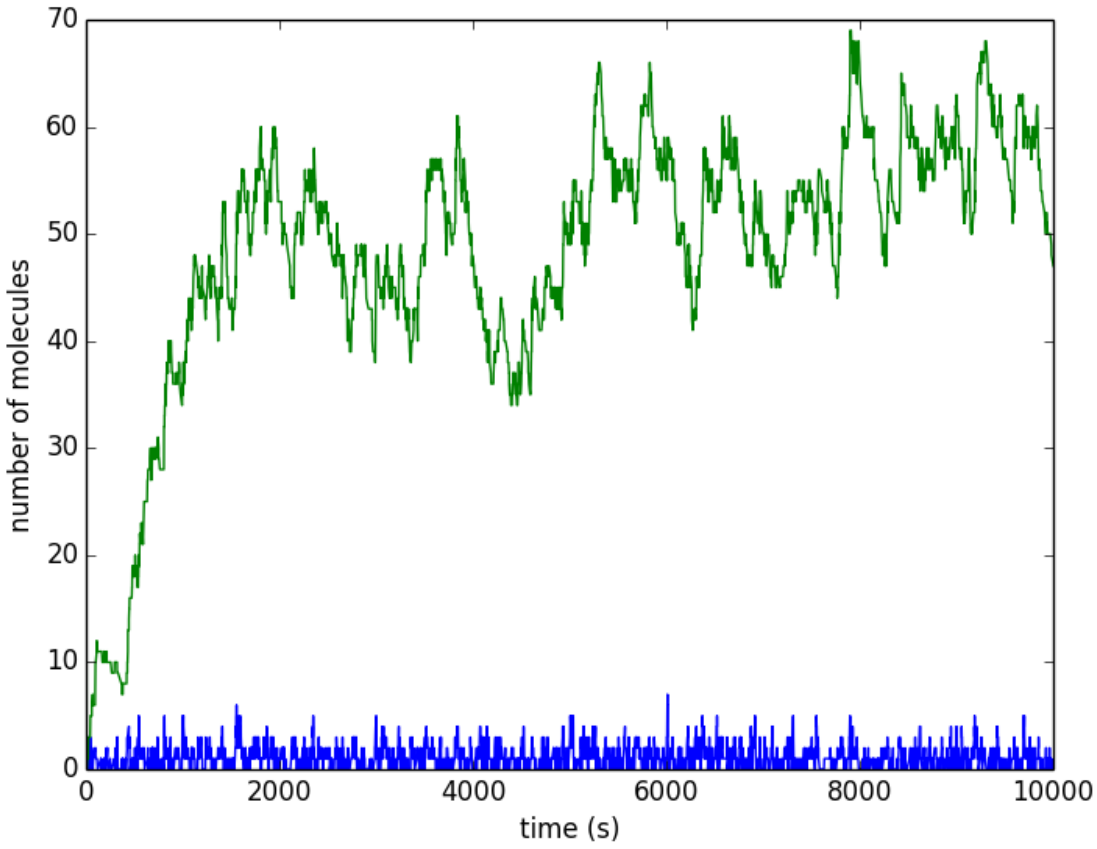
CHAPTER 2

Examples

Here's what Dijkstra's algorithm looks like using a `pqdict`.

This is an example of a stochastic simulation algorithm that simulates chemical reactions with small numbers of molecules.

Output:



CHAPTER 3

API Reference

3.1 pqdict class

class pqdict.**pqdict**(*data=None*, *key=None*, *reverse=False*, *precedes=<built-in function lt>*)

A collection that maps hashable objects (keys) to priority-determining values. The mapping is mutable so items may be added, removed and have their priority level updated.

Parameters **data** : mapping or iterable, optional

Input data, e.g. a dictionary or a sequence of items.

key : callable, optional

Optional priority key function to transform values into priority keys for sorting. By default, the values are not transformed.

reverse : bool, optional

If `True`, *larger* priority keys give items *higher* priority. Default is `False`.

precedes : callable, optional (overrides `reverse`)

Function that determines precedence of a pair of priority keys. The default comparator is `operator.lt`, meaning *smaller* priority keys give items *higher* priority.

Attributes

Methods

classmethod **fromkeys**(*iterable*, *value*, ***kwargs*)

Return a new pqict mapping keys from an iterable to the same value.

heapify([*key*])

Repair a broken heap. If the state of an item's priority value changes you can re-sort the relevant item only by providing `key`.

Properties

keyfn
Priority key function

precedes
Priority key precedence function

Dictionary API

These do as expected. See `dict` for more details.

len(*pq*)

pq[key]

pq[key] = value

del pq[key]

key in pq

get(*key*[, *default*])
D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

pop([*key*[, *default*]])
If `key` is in the pqdict, remove it and return its priority value, else return `default`. If `default` is not provided and `key` is not in the pqdict, raise a `KeyError`.

clear() → None. Remove all items from D.

update([*other*])
D.update([E,]**F) -> None. Update D from mapping/iterable E and F. If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

setdefault(*key*[, *default*])
D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d if k not in D

copy()
Return a shallow copy of a pqdict.

Iterators and Views

> **Warning:** For the following sequences, order is arbitrary.

iter(*pq*)

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

items() → a set-like object providing a view on D's items

Note: In Python 2, the above methods return lists rather than views and `pqdict` includes additional iterator methods `iterkeys()`, `itervalues()` and `iteritems()`.

Priority Queue API

top()
Return the key of the item with highest priority. Raises `KeyError` if pqdict is empty.

pop([*key*[, *default*]])
If `key` is not provided, remove the top item and return its key, or raise `KeyError` if the pqdict is empty.

additem(*key*, *value*)
Add a new item. Raises `KeyError` if key is already in the pqdict.

updateitem(*key*, *new_val*)
Update the priority value of an existing item. Raises `KeyError` if key is not in the pqdict.

topitem()
Return the item with highest priority. Raises `KeyError` if pqdict is empty.

popitem()
Remove and return the item with highest priority. Raises `KeyError` if pqdict is empty.

pushpopitem(*key*, *value*)
Equivalent to inserting a new item followed by removing the top priority item, but faster. Raises `KeyError` if the new key is already in the pqdict.

replace_key(*key*, *new_key*)
Replace the key of an existing heap node in place. Raises `KeyError` if the key to replace does not exist or if the new key is already in the pqdict.

swap_priority(*key1*, *key2*)
Fast way to swap the priority level of two items in the pqdict. Raises `KeyError` if either key does not exist.

Heapsort Iterators

Note: Iteration is in descending order of priority.

Danger: Heapsort iterators are destructive: they are generators that pop items out of the heap, which amounts to performing a heapsort.

popkeys()
Heapsort iterator over keys in descending order of priority level.

popvalues()
Heapsort iterator over values in descending order of priority level.

popitems()
Heapsort iterator over items in descending order of priority level.

Warning: The names of the heapsort iterators in v0.5 (iterkeys, itervalues, iteritems) were changed in v0.6 to be more transparent: These names are not provided at all in Python 3, and in Python 2 they are now part of the dictionary API.

3.2 Functions

pqdict.**nsmallest**(*n*, *mapping*)
Takes a mapping and returns the n keys associated with the smallest values in ascending order. If the mapping

has fewer than n items, all its keys are returned.

Equivalent to: `next(zip(*heapq.nsmallest(mapping.items(), key=lambda x: x[1])))`

Returns list of up to n keys from the mapping

pqdict.**nlargest**(*n*, *mapping*)

Takes a mapping and returns the n keys associated with the largest values in descending order. If the mapping has fewer than n items, all its keys are returned.

Equivalent to: `next(zip(*heapq.nlargest(mapping.items(), key=lambda x: x[1])))`

Returns list of up to n keys from the mapping

- genindex

A

additem() (pqdict.pqdict method), 11

C

clear() (pqdict.pqdict method), 10
copy() (pqdict.pqdict method), 10

F

fromkeys() (pqdict.pqdict class method), 9

G

get() (pqdict.pqdict method), 10

H

heapify() (pqdict.pqdict method), 9

I

items() (pqdict.pqdict method), 10
iter() (pqdict.pqdict method), 10

K

keyfn (pqdict.pqdict attribute), 9
keys() (pqdict.pqdict method), 10

L

len() (pqdict.pqdict method), 10

N

nlargest() (in module pqdict), 12
nsmallest() (in module pqdict), 11

P

pop() (pqdict.pqdict method), 10, 11
popitem() (pqdict.pqdict method), 11
popitems() (pqdict.pqdict method), 11
popkeys() (pqdict.pqdict method), 11
popvalues() (pqdict.pqdict method), 11
pqdict (class in pqdict), 9
precedes (pqdict.pqdict attribute), 10
pushpopitem() (pqdict.pqdict method), 11

R

replace_key() (pqdict.pqdict method), 11

S

setdefault() (pqdict.pqdict method), 10
swap_priority() (pqdict.pqdict method), 11

T

top() (pqdict.pqdict method), 11
topitem() (pqdict.pqdict method), 11

U

update() (pqdict.pqdict method), 10
updateitem() (pqdict.pqdict method), 11

V

values() (pqdict.pqdict method), 10