The Journal of Open Source Software

Python Sorted Containers

Grant Jenks[1]

1 None

DOI: 10.21105/joss.01330

Software

- Review
- Repository
- Archive

Submitted: 31 January 2019
Published: 03 June 2019

Summary

The standard library of popular languages like C++, Java, and C# provide sorted container data types based on binary tree data structures. While Python (1995) has risen in popularity, the Standard Library still lacks these common data types. Part of the challenge has been Python's rich object model which makes binary trees implemented in Python slow in terms of both memory and processor usage. To overcome the overhead of the interpreter, C-extensions are used by the Python core developers. In doing so, flexibility is tradeoff for performance. The goal of the Python core developers is to provide the right set of high-level APIs so that algorithms and data structures can be implemented efficiently. The Python Sorted Containers library uses Python's high-level APIs to efficiently implement sorted container data structures.

Python's collections data structures are based on three data types: sequences, mappings, and sets. These data types are implemented and most commonly used as list, dictionary, and set objects. The Python Sorted Containers library introduces new variants of these three data types that are each sorted: sorted list, sorted dictionary, and sorted set. In each case, the original semantics are extended to preserve sorted order of the contained elements with respect to mutating operations. When unable to preserve the sorted order constraint, the functionality is either non-existent or an error is raised from the library. Python's "sorted" built-in function also supports a "key" parameter which specifies a callable used to extract a comparison key from elements. When initializing a sorted container data type, the key paramemter is likewise supported.

Internally, Python Sorted Containers uses a list of sublists data structure that is like a B-tree contstrained to two levels of nodes. The maximum of each sublist is maintained in a separate list. To lookup an element, the list of maximums is bisected using the "bisect" module in the Standard Library. Using the bisected maximums index, the corresponding sublist is bisected to find the index of the desired element. To index the k'th element, a separate positional index, known as the "Jenks" index, is built and maintained. The positional index is like the order statistic of a binary tree densely packed into a list. By maintaining the size of the sublists as proportional to the $\sqrt[3]{n}$ the amortized time complexity of all operations is $O(\sqrt[3]{n})$. This bound works well for up to billions of elements which often exhausts all available memory.

Python Sorted Containers overlaps and extends the "bisect" and "heapq" modules provided in the Standard Library. In contrast to the function-oriented interface provided by these modules, Sorted Containers provides an object-oriented interface. Externally, SQLite in-memory indexes, Pandas DataFrame indexes, and Redis sorted sets provide similar functionality. These data structures are applied in priority queue, multiset, nearest neighbors, intervals, and ranking algorithms. Sorted Containers is used by scientific open source projects such as: Angr (2016), a binary analysis platform from UC Santa Barbara; Astropy (2018), a community Python package for astronomy; Dask Distributed (2015), a library for dynamic task scheduling by Anaconda; Trio (2017), an asynchronous I/O library; and Zipline (2016), an algorithmic trading library by Quantopian.

Acknowledgements

Thank you to Daniel Stutzbach for the "blist" (2014) software project to which Sorted Containers owes much of the original interface design.

Thank you to Raymond Hettinger for the "SortedCollection" recipe (2010) which originally inspired the support and design of the "key" parameter feature.

Thank you to Manfred Moitzi for the "bintrees" software project (2017) which motivated the range-based tree traversal interfaces.

Thank you to Dan Stromberg for the benchmark comparisons of less common binary tree data structures (2019) like treap, splay, and scapegoat.

Thank you to the open source community that has contributed bug reports, documentation improvements, and feature guidance in development of the project.

References

Developers, P. C. (1995). *Python programming language*. Python Software Foundation. Retrieved from https://www.python.org/

Hebert, E., Sanderson, S., Jevnik, J., Frank, R., Wiecki, T., & others. (2016). Zipline, a pythonic algorithmic trading library. http://www.zipline.io/.

Hettinger, R. (2010). SortedCollection (python recipe). http://code.activestate.com/recipes/577197-sortedcollection/.

Moitzi, M. (2017). Bintrees: Binary tree package. https://github.com/mozman/bintrees.

Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., Shupe, D. L., et al. (2018). The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package, *156*, 123. doi:10.3847/1538-3881/aabc4f

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In K. Huff & J. Bergstra (Eds.), *Proceedings of the 14th python in science conference* (pp. 126–132). doi:10.25080/Majora-7b98e3ed-013

Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., et al. (2016). SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE symposium on security and privacy*.

Smith, N. J. (2017). Trio: Async programming for humans and snake people. https://trio.readthedocs.io/.

Stromberg, D. (2019). Dictionary-like trees. http://stromberg.dnsalias.org/~dstromberg/datastructures/.

Stutzbach, D. (2014). Blist: An asymptotically faster list-like type for python. http://stutzbachenterprises.com/blist/.