PEP 3128: Python BList

CSCI 3155

Fadhil Suhendi, Zhilli Yang, Thomas Kelly

Introduction of BList

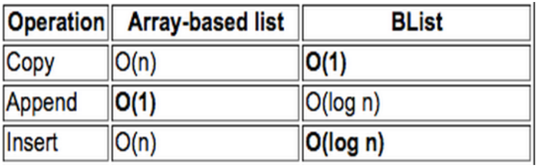
- New Data Structure for Python
- Based on principles of B+tree
- Blist has array-like and tree-like aspects
- Replace existing list type
- This data structure was rejeted

Motivation

- For large inputs
- Offers array-like perfomance on small list
- It also offers asymptotic perfomacne for deleting and inserting

Comparison with List

- Comparison based on perfomance



Operation	**Array-based list**	**BList**
Copy	O(n)	**O(1)**
Append	**O(1)**	O(log n)
Insert	O(n)	**O(log n)**

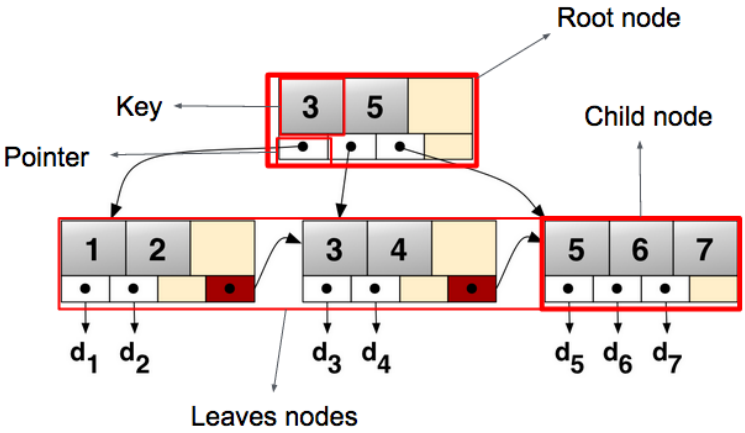
Comparison with List

- List is using dynamically array type
- Blist is using a flexible, tree structure
- Blist uses twice memory usage as array-list
- Blist root node has at leat 2 childrens
- List has fewer than 2 childrens

About B+tree

- N-ary tree
- Large number of children per node
- No data in internal nodes
- Leaves nodes are in the same level
- Data pointers

ABout B+tree


Root node
Key
Pointer
Child node
3
5
1
2
3
4
5
6
7
d_1 d_2
d_3 d_4
d_5 d_6 d_7
Leaves nodes

Implementation

- Without using blist

```
items = [5,6,2]
more_items = function_that_returns_a_list()
```

- Using blist:

```
from blist import blist
items = ([5,6,2])
more_items = blist(function_that_returns_a_list()
```

Examples of blist

```
from blist import *
x = blist([0]) # x is a blist with one element
x *= 2**29 # x is a blist with > 500 million elements
x.append(5) # append to x
y = x[4:-234234] # Take a 500 million element slice f
del x[3:1024] # Delete a few thousand elements from x

from blist import sortedlist
my_list = sortedlist([3, 7, 2, 1])  #sortedlist([1, 2
my_list.add(5)  #sortedlist([1, 2, 3, 5, 7])
```

Pros

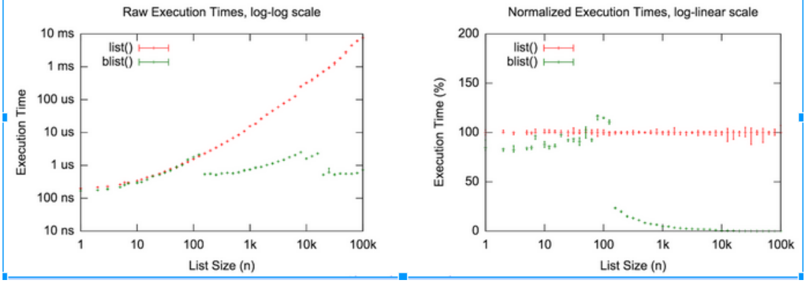
- BList is useful for intermediate-users, but no for beginers users.
- Blist has a better perfomance than array-based list
- Matching the list-API reduces the learning curve to near-zero

Cons

- Performance & desirability in real-world applications has yet to be proven – more testing is required
- Increasing the number of data types available to developers makes their job more difficult
- Adding BList to Python's default library may have an impact on extension modules, requiring extensive debugging
- The are some important use cases where BLists are actually slower than regular Lists
- The array-based List code is easy to maintain & simple to understand, while BLists would present a learning curve for many developers

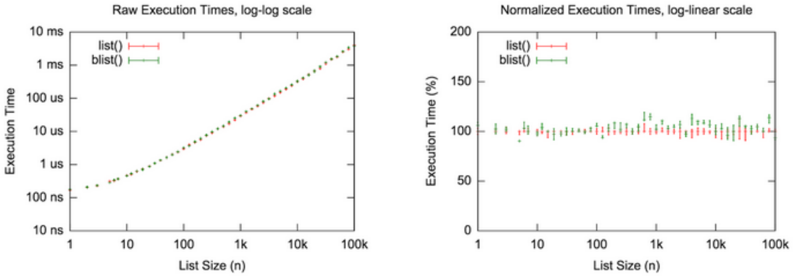
Perfomance of Blist: Simple Addition

Setup: `x = TypeToTest(range(n))`
Timed: `x + x`

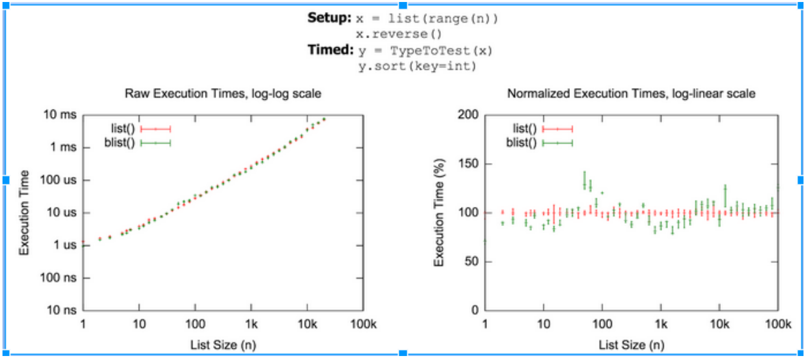


Perfomance of Blist: For Loop Iteration

Setup: `x = TypeToTest(range(n))`
Timed:
```
for i in x:
    pass
```

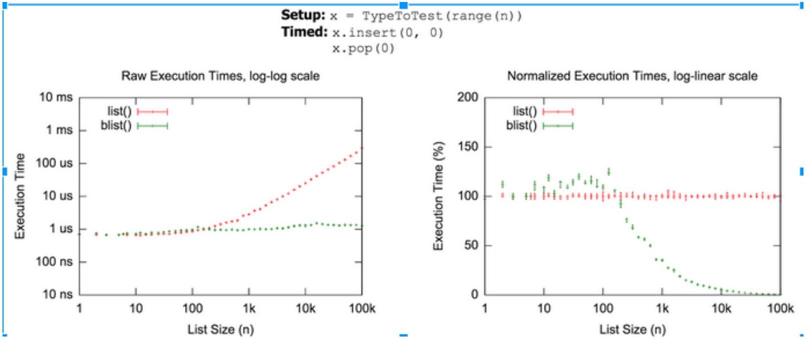


Perfomance of Blist: Sorting a Reversed List



Setup:
```
x = list(range(n))
x.reverse()
```
Timed:
```
y = TypeToTest(x)
y.sort(key=int)
```

Perfomance of Blist: FIFO Operations


Setup: `x = TypeToTest(range(n))`
Timed: `x.insert(0, 0)`
`x.pop(0)`

. . . So why was it rejected?

- Various reasons, but mostly because it isn't very user-friendly
- "There is too much value in a simple C API, low space overhead for small lists, good performance [in] common use cases, and having performance that is easily understood. The BList implementation lacks these virtues and it trades-off a little performance in common cases . . . for much better performance in uncommon cases." – Raymond Hettinger
- So even though it outperforms List on many different tasks involving very large numbers of elements, Python rejected BList because it was too difficult for inexperienced users to use & understand
- This seems to suggest that Python devs prefer user-friendly code over more powerful code – "If it ain't broke, don't fix it!"

Other reasons. . .

- So even though it outperforms List on many different tasks involving very large numbers of elements, Python rejected BList because it was too difficult for inexperienced users to use & understand
- This seems to suggest that Python devs prefer user-friendly code over more powerful code – “If it ain’t broke, don’t fix it!”

Summary

- Blist is a new data structure
- The strongest part of BList is about perfomances
- The weakness part of BList is about memory usage

Conclusion

- In 2008 BList was rejected by community. However, due to the powerful perfomance of Blist, BList is still released as an extension module of python programming language. If many python users will use Blist, it is possible that in the future, Blist will be considered to replace the the array-based list.

References

- https://www.python.org/dev/peps/pep-3128/#id6
- https://pypi.python.org/pypi/blist/?
- http://legacy.python.org/dev/peps/pep-3128/
- http://stutzbachenterprises.com/performance-blist
- http://byumcl.bitbucket.org/bootcamp2013/labs/perfomance