schulzdLab1

March 14, 2021

1 Lab 1: Comparing the Run Times of Common Python Data Structures

David Schulz

1.1 Introduction

In CS2852 Data Structures, we learned about a number of data structures such as Array Lists, Linked Lists, Stacks implemented with Array Lists, Queues implemented with Linked Lists, Hash tables, and Binary Search Trees. We also learned how to model the asymptotic run time of the operations (e.g., contains, add, and remove) associated with these data structures using big-O notation.

In this class, we are going to use Python with Jupyter Notebooks. These notebooks will allow us to include plots alongside our implementations of algorithms and data structures.

The data structures and algorithms we learned are not specific to Java, however, and are also applicable to Python. Most programming languages include similar data structures and the caveats about the run times of their operations apply. Python contains several built-in data structures, the Python standard library contains additional data structure implementations, and Python and its standard library also implement algorithms for common operations.

In this lab, we are going to practice benchmarking in Python by comparing operations on lists and deques.

1.2 Problem 1: Benchmark the append() operation on Lists and Deques

[1]:
```
from collections import deque
import time

list1 = []
deque1 = deque()

start_time = time.perf_counter()
for i in range(1000000):
    list1.append(1)
end_time = time.perf_counter()
list1_elapsed = end_time - start_time

start_time = time.perf_counter()
for i in range(1000000):
```

```
    deque1.append(1)
end_time = time.perf_counter()
deque1_elapsed = end_time - start_time

print("List Count:", len(list1))
print("Deque Count:", len(deque1))
print()
print("List Time:", list1_elapsed)
print("Deque Time:", deque1_elapsed)
```

```
List Count: 1000000
Deque Count: 1000000

List Time: 0.0873137479647994
Deque Time: 0.06876155699137598
```

1.3 Problem 2: Benchmark the insert(0, ITEM) operation on Lists and Deques

[2]:
```
list2 = []
deque2 = deque()

start_time = time.perf_counter()
for i in range(1000000):
    list2.insert(0, 1)
end_time = time.perf_counter()
list2_elapsed = end_time - start_time

start_time = time.perf_counter()
for i in range(1000000):
    deque2.insert(0, 1)
end_time = time.perf_counter()
deque2_elapsed = end_time - start_time

print("List Count:", len(list2))
print("Deque Count:", len(deque2))
print()
print("List Time:", list2_elapsed)
print("Deque Time:", deque2_elapsed)
```

```
List Count: 1000000
Deque Count: 1000000

List Time: 263.3511101480108
Deque Time: 0.09620975004509091
```

1.4 Problem 3: Compare the "in" (contains) operation on Lists and Sets

[5]:
```
list3 = []
set3 = set()

for i in range(100000):
    list3.append(i)
    set3.add(i)

start_time = time.perf_counter()
for i in range(100000):
    -5 in list3
end_time = time.perf_counter()
list3_elapsed = end_time - start_time

start_time = time.perf_counter()
for i in range(100000):
    -5 in set3
end_time = time.perf_counter()
set3_elapsed = end_time - start_time

print("List Count:", len(list3))
print("Set Count:", len(set3))
print()
print("List Time:", list3_elapsed)
print("Set Time:", set3_elapsed)
```

```
List Count: 100000
Set Count: 100000

List Time: 79.76308657799382
Set Time: 0.0045263179345056415
```

1.5 Reflection Questions

1. Create a table using Markdown syntax of the run times from the benchmarks. The table should have 3 columns: the operation, the first data structure, and the second data structure. Each cell should contain the run-time in seconds. Make sure the table includes a header.

Operation	List	Deque/Set
append()	0.087	0.069
insert()	263.35	0.096
in	79.76	0.0045

2. In which cases were the run times approximately similar versus different?
    - The times for the append() operation were similar, but the insert() and in operations

were very different.

3. Add two more columns to your table. In these columns, put the big-o notation for the operations for those data structures based on what you remember from CS2852.

Operation	List	Big-O of List Op	Deque/Set	Big-O of Other Op
append()	0.087	O(1)	0.069	O(1)
insert()	263.35	O(n)	0.096	O(1)
in	79.76	O(n)	0.0045	O(1)

4. Are the ordering of the run times consistent with the ordering based on big-o notation?
    - Yes, they are very clearly consistent.