Stephen Provost

CS 301

Assignment 2

2/24/2020

Having Fun With Lists and Dictionaries in Python

Lists and dictionaries in Python is something that manages to boggle the mind due to the fact that some of the built-in functions are so clean, resulting in minimal change times even for some of the methods that had resulted in very large times for our answers in Assignment one. Comparing the differences can result in some knowledge regarding the Big O operations, but even so, comparatively, the two seem to be very similar.

Item Adding:

For instance, adding items to the programs results in very similar information, with a clear indication that the more data that you have, the longer it takes to fill the list, even when the randomizer is excluded from the timing mechanism:

|  |  |
| --- | --- |
| List | Dict |
| A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

Displayed here are two different tests, one involving Lists, and the other involving Dictionaries, and ranging from left to right the lists go from 100, 1,000, 10,000, 100,000, 1,000,000, to 10,000,000. I assumed that dictionaries would take more time to add to the list, due to the fact that you’re adding a key and a value, but they took much less time. However, both of these seem to be of the order O(n^2), making me believe that the lists and dictionaries have to determine the array size before adding the next item to the list/dictionary.

Ranges and Indexes:

Another example would be using access methods in several different ways, such as negative or positively. These are methods only available for lists, as dictionaries require you to have the specific key that you’re looking for, in order for it to find the value associated with it. This is a result of dictionaries not being indexed.

|  |  |
| --- | --- |
| Accessing from Range | Accessing Specific Index |
| A screenshot of a social media post  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

Accessing a range seems to be O(n), due to the fact that it is a very straight line unaffected by size of the list itself, but by how much it has to find. Accessing a specific index though is O(1), or constant time, which makes me believe that the list is able to immediately grab an item from an index without having to cycle through the list. Additionally, two other methods that are mirroring these times are the reverses of these, accessing the list from negative access list (a single item), and accessing several items from a negative range. The former is O(1), and the latter is O(n). These further support the idea that it is not necessary for the list to roll through the entirety of the items to find specific points.

Additionally, two other methods that are O(1) are ones that change or return a specific file in either the list or dictionary, further emphasizing the fact that lists access only an indexed item, without having to count through each indexed item. Changing item, and changing keys, simply and almost instantly manipulate the information to change it to whatever the user has indicated, regardless of the size of the list or the location of the entry to be changed.

Deletion:

Removing entries from lists and dictionaries provide additional, interesting clarifications about the objects as well. For instance, using the command ‘pop’ for a dictionary is O(1), resulting in constant time being utilized no matter the location of the item in the object, whereas a list seems to be O(n) or O(k), as it has to go through the list until it reaches the item and removes it.

|  |  |
| --- | --- |
| List Pop | Dict Pop |
| A screenshot of a social media post  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

Removing, meanwhile, seems to be O(n) or O(k) as well due to the fact that it seems to take longer, although I would assume that it’s Order is the same as pop. I’m making an assumption, but I believe pop starts from the right side of the list (the highest indexes first), while the delete option starts from the smallest indexes first.

Copy Manipulation:

There are four different copy methods present between lists and dictionaries (2 each). This is the .copy() method and then the list(parameter) or dict(parameter). There are small but noticeable differences in times present between these different methods, although the dictionaries tended to take less time (again, still a surprise considering the increased sizes of the dictionaries having an attribute associated with each. I’m assuming that not having an index has a greater effect on these methods though!). All of these seem to be O(n), and I can not think of a reason why they would be anything else due to the simple fact that they are copying each entry from the dictionary or list to the other. The only thing I can say that surprises me is that they are so much quicker than adding items to the list or dictionary in the first place in equal quantities.

Length Call:

Calling for the length of dictionaries or lists is O(1), uncaring of the specific size of the list or dictionary itself. I can only assume that when you make a list or dictionary, that there is an unseen parameter that includes the size of the list and adds to it when items are added to the list, making it so that when the length is called, only that number has to be returned.

Clear Methods:

Lastly, the clear methods for lists and dictionaries are interesting in that they are O(n), due to the fact that they seem to have to delete the items individually. Initially I assumed they would have been O(1), and just disassociate with the information present, but that would make an on-going issue over time due to the information simply not being deleted, resulting in either memory or security issues.