Ashutosh Rai
CS 335

Lab 7: Exploring Hashing

Part 1

Writeup 1

The most common and basic implementation of hash tables used in Python is the Python dictionary, which is a data type built into Python. The dictionaries are indexed by keys, which can be any immutable type such as strings, numbers or tuples. Mutable objects such as lists cannot be used as keys as they are modifiable. Dictionaries can be thought as an unordered set of key-value pairs where the keys have to be unique within a dictionary. Some core functionalities offered by the Python dictionary are storing of the key-value pair, extracting value via key and deletion of the key-value pair.

Python dictionaries use open addressing to resolve the hash collisions, which occurs when hash values for two or more keys: value pairs are the same. The dictionary is implemented as a contiguous block of memory, which allows O(1) lookup time due to indexing. Each slot in the table can store only one entry, which consists of <hash, key, value>, implemented as a C struct. When a new dictionary is initialized, 8 slots in the table are created. The dictionary is resized when it is two-thirds full.

Design 1

A good use case for a hash table would be to store the id number and the name of students at a big school. The operation that is optimized in this case would be fast look-up. This makes sense as in a bog school, you can expect to have thousands and thousands of students. For various purpose such as finding students info and other official reasons, looking up students name with their id number would be useful. Sometimes, it might be a repetitive part of a task where the names of several thousand students might have to be extracted just through their id numbers. As the lookup time is crucial in these cases, a hash table is a good implementation for storing the student id number as keys and the student names as values.

Implementation 1

The code is attached

Part 2

Design 2a

To build my own hash table in Python, I would do the following:

- Create a list of lists. This will be our hash table. The hash value will correspond to the index of the outer list. The purpose of the having lists inside a list is to take care of hash collisions when the hash values are the same.
- Define a hash function. The hash function should take the key value and return a hash value after the operation of the mathematical operations required by the hash function. The hash value which should be within the range of the indexes of the outer list.
- Define a function that would allow you to insert the key-value pair into the hash table. The function could take the name of the hash table, key, and value as the arguments.

Implement 2

The code is attached

Design 2b

While working on the initial design, I forgot to design the functions to lookup the value based on key, and delete the key-value pair based on key.

Hence, the additions I had to make were:

- Function to lookup the value based on key: The function takes the key as an argument and finds the hash value from the key. Then the inner list is extracted from the outer list of the table and traversed for the key we are looking for. If the key is found, the value is returned. If the key is not present then appropriate message regarding the absence of the key-value pair is printed.
- Function to delete the key-value pair based on key: The function takes the key as an argument and finds the hash value from the key. Then the inner list is extracted from the outer list of the table and traversed for the key we are trying to delete. If the key is found, the key-value pair is removed from the inner list. If the key is not present then appropriate message regarding the absence of the key-value pair is printed.

Part 3

Design/Implement 3

Brief test design overview:

I generated a list of random unique IDs and a list of random names, both of size 100000. Then I proceeded to insert all of the corresponding key (student ID) - value (name) pair into both the Python dictionary and my hash table. I timed the total time taken to do this for both of them. Similarly I also timed the time taken to search all the key-value pairs and then delete the key-value pairs.

The code is attached.

Writeup 3

Screenshot of the results:

```
------ Test Results for 100000 pairs (Python Dictionary vs. My Hash Table) -------

Insertion time for the Python dictionary: 0.06133151054382324 seconds
Insertion time for the my hash table: 0.21695256233215332 seconds

Searching time for the Python dictionary: 0.03952383995056152 seconds
Searching time for the my hash table: 0.12429594993591309 seconds

Deletion time for the Python dictionary: 0.04057192802429199 seconds
Deletion time for the my hash table: 0.1657421588897705 seconds
```

As one can see, for all three operations of insertion, searching and deletion, it took my hash table longer than the Python dictionary. An interesting observation I made was that it generally took my hash table about 4 times as much time as it took the Python dictionary for all three operations.

I am not surprised by the results as my hash table, including my hash function, was something I put together in a relatively shorter amount of time compared to the hash table used for Python dictionaries. As my hash function is very basic, the Python dictionary hash function probably is much better in terms of generating hash values. On top of that, as Python dictionary uses open addressing, so that improves the Python dictionary's hashing as well.