**CS469 Data Structures and Algorithms**

**HOS05 Hash Tables**

04/26/2023 Reviewed by Christopher Sharp

01/29/2024 Reviewed by Anh Nguyen

09/29/2024 Reviewed by Shahid Khan

School of Technology and Computing (STC) @City University of Seattle (CityU)

**Before You Start**

* The document’s examples are written in Python. Please finish the Python tutorial in Module0 folder before you start the assignment.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Understand Hash Tables
* Implement Hash Tables in python

**Resources**

* Bhargava, Y, A. (2016). Grokking algorithms GitHub repository. Retrieved from: <https://github.com/egonSchiele/grokking_algorithms>
* Data Structures and Algorithms in Python:

<https://ebookcentral.proquest.com/lib/cityuseattle/reader.action?docID=4946360>

* Grokking algorithms: an illustrated guide for programmers and other curious people: <https://learning.oreilly.com/library/view/grokking-algorithms/9781617292231/OEBPS/Text/kindle_split_001.html>

# Hash Table

The hash table is a data structure that is directly accessed based on the key value. In other words, it accesses the record by mapping the key code value to a location in the table to speed up the search. This mapping function is called a hash function, and the array storing records is called a hash table.

Given a table M, there is a function f(key). For any given key value key, if the address of the record in the table containing the key can be obtained after substituting the function into the function, then the table M is called a hash table, and the function f(key) is a hash function.

Each position in the hash table is called a Slot. These Slots are numbered starting from 0. At the beginning, the hash table is empty, and all Slots are initialized to None. The figure below shows an empty hash table with a length of 11.

图示

描述已自动生成

The mapping corresponding to the element in the hash table and its position is called a hash function. Given an element, the position in the hash table can be obtained through the hash table. Suppose we have the following elements: 54, 26, 93, 17, 77, 31. Through the remainder method, the remainder obtained by dividing the element by the length of the hash table is used as the hash value.

表格

描述已自动生成

After calculating the hash value corresponding to each element, we can insert the element into the hash table.

图示

描述已自动生成

## Collision

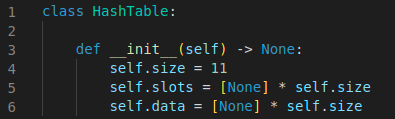
Ideally, the hash function will assign each key to a unique bucket, but most hash table designs use **imperfect** hash functions, and different keywords may get the same hash address, that is, k1≠k2 , And f(k1)=f(k2). This phenomenon is called Collision. Keywords with the same function value are called synonyms for the hash function.

# Implement the Map abstract data type in the Python dictionary

Let us implement it in Python.

**Create a file called hashTable.py.**

Type the following code:



We use two lists to create a HashTable class that implements the Map abstract data type.

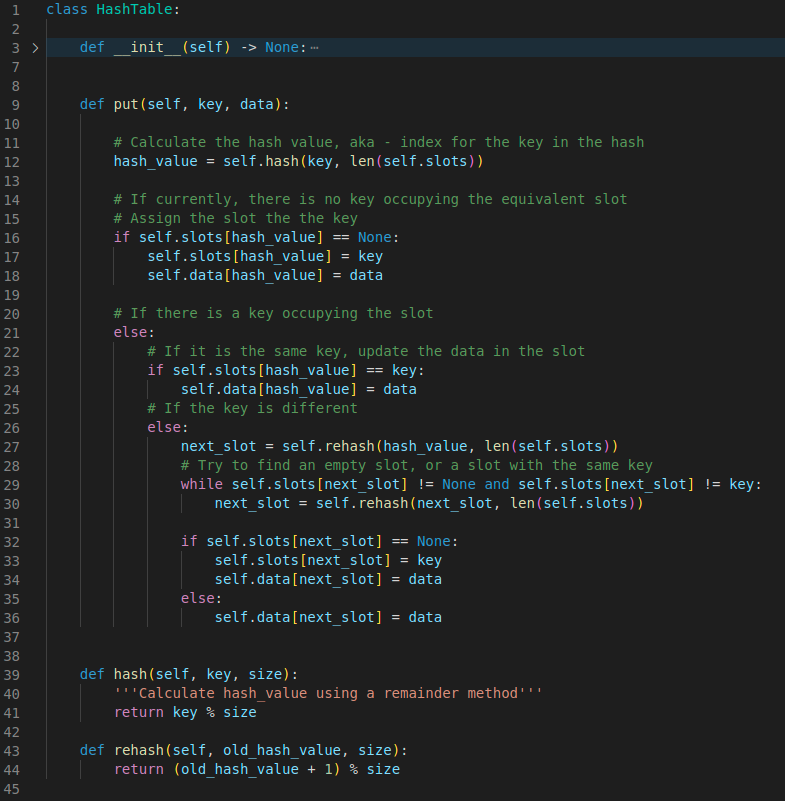
* slots: hold the key items
* data: hold the data values.

When we look up a key, the corresponding position in the data list will hold the associated data value. We will treat the key list as a hash table using the ideas presented earlier.

Note that the initial size for the hash table has been chosen to be 11. Although this is arbitrary, it is important that the size be a prime number so that the collision resolution algorithm can be as efficient as possible.

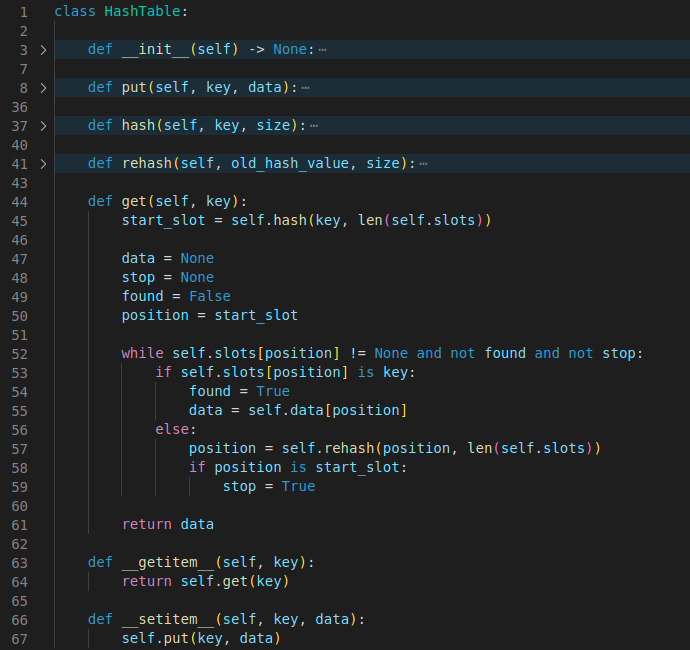
Under the HashTable class, add the following functions:

* hash() implements the simple remainder method. The collision resolution technique is linear probing with a “plus 1” rehash function.
* put() assumes that there will eventually be an empty slot unless the key is already present in the self.slots. It computes the original hash value and:
  + If that slot is not empty, iterates the rehash function until an empty slot occurs.
  + If a nonempty slot already contains the key, the old data value is replaced with the new data value.

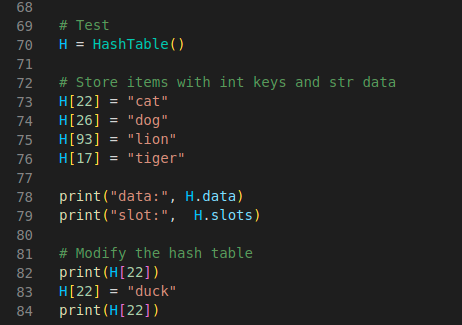


Next, under HashTable class, add the following functions:

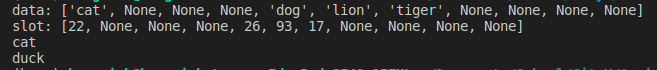
* get() function begins by computing the initial hash value. If the value is not in the initial slot, rehash is used to locate the next possible position.
  + Notice that line 58 guarantees that the search will terminate by checking to make sure that we have not returned to the initial slot. If that happens, we have exhausted all possible slots and the item must not be present.
* The final methods of the HashTable class provide additional dictionary functionality. We overload the \_\_getitem\_\_() and \_\_setitem\_\_() methods to allow access using “[]”. This means that once a HashTable has been created, the familiar index operator will be available.



The following session shows the actual operation of this class of HashTable. First, we will create a hash table and store some items with integer keys and string data values.



Your output should look like this:



# Analysis

We stated earlier that in the best-case hashing would provide a O(1), constant time search technique. However, due to collisions, the number of comparisons is typically not so simple. Even though a complete analysis of hashing is beyond the scope of this text, we can state some well-known results that approximate the number of comparisons necessary to search for an item.

The most important piece of information we need to analyze the use of a hash table is the load factor, 𝜆. Conceptually, if 𝜆 is small, then there is a lower chance of collisions, meaning that items are more likely to be in the slots where they belong. If 𝜆 is large, meaning that the table is filling up, then there are more and more collisions. This means that collision resolution is more difficult, requiring more comparisons to find an empty slot. With chaining, increased collisions means an increased number of items on each chain.

As before, we will have a result of both a successful and an unsuccessful search. For a successful search using open addressing with linear probing, the average number of comparisons is approximately 卡通人物

中度可信度描述已自动生成and an unsuccessful search gives 卡通人物

中度可信度描述已自动生成If we are using chaining, the average number of comparisons is 图示, 示意图

描述已自动生成 for the successful case, and simply 𝜆 comparisons if the search is unsuccessful.

**Q:** **If the user wants to input a string as the key, which function in the above code should we modify? What method would you use to rewrite this function?**

**Please state your answer in a PDF/text file in the repository folder.**

# Optional Resource

If you want to learn more about the Hash Tables and solve some code challenges. You can try some Leetcode Coding challenges: <https://leetcode.com/problems/integer-to-roman/>

Because this part is optional, the TA won’t be responsible for answering questions for the challenges on Leetcode. Instead, you can visit the question’s discussion board to find hints and solutions.

# Push Your Work to Github

Open terminal and make sure you’re in the repository folder. (i.e: hos05\_courseName\_GitHubUserName)

**Type the following command to upload your work**:

>>> git add .

>>> git commit -m "Submission for HOS05 - YourName"

>>> git push origin master