Lam Dang
For CS335: ADS
Lab 7: Hash Table.

Writeup 1b:
A popular use case of using Hash Table is keeping a record of flight seat on the plane, based on the person's name. We are going to create an interface that allow administrator to register, search and remove passenger with their flight seats. The key that is registered will be the name of the passenger, and the value mapped to the key is the seat number.

The reason it is a good idea to use hash table for this case is that:
- It use key-value pair to map the name of passenger with the seat number.
- It requires no order/hierarchy of any sort (unlike some of the tree data structure) which allow fast insertion and deletion.
- It is very straightforward to understand and implement.

Writeup 2b:
- Old: I used to store Key-Value pair in tuples.
- New: When I realized tuple is a immutable data type, I decided to create a new class that store key, value.
- Old: Check load balance AFTER inserting new item in the Hash Table
- New: Check load balance BEFORE inserting new item in the Hash Table.

Writeup 3:
My Speculation for why my Hash Table is slower than Python Hash Table:
- My Collision Handling technique is different from Python's (Bucket list vs Open Address)
- My hash function and compression is not as good as Python.
- My resize function takes longer time.

Book Reference:
Data Structure and Algorithm in Python

Web Links Reference:
https://mail.python.org/pipermail/python-list/2000-March/048085.html
https://stackoverflow.com/questions/327311/how-are-pythons-built-in-dictionaries-implemented
https://www.freecodecamp.org/news/exploring-python-internals-the-dictionary-a32c14e73efa/

CS335 : Lab 7 · Hash Table

Writeup 1a.

The most basic / common type of hash table in Python 3 (Mine is Python 3.7) is dictionary type.

Useful information about Python's Dictionary:

- Python use open addressing to resolve hash collision
- Each slot of table created by Python is one ~~slot only~~ entry only.
- Each entry is a combination of < hash, key, value >, implemented by C struct.
- Each new dict starts with 8 slots

Useful term:

- hashing : converting a string to a integer
- probing : search slots by slots to find an empty slot.

Implementation.

```
Insertion (key, value)
    index = hash ( key).
    go to that index slot
    IF empty : → Add → Done
    IF Not empty:
        → If the inserting key = Inserted key:
            → Change value ; DONE
        → Else:
            → Probing and add to ~~n~~ probe slot; DONE

Lookup (key).
    index = hash ( key).
    goto that index slot
    IF lookup key == slot key → return value ; DONE
    IF Not:
        Start probing.
            → If Found matching key → return value ; DONE
            → Else → Return None ; DONE

Delete ( key)
    Same as lookup , but delete object in slot rather than return
```

JAN FEB [✓ MAR] APR MAY JUN JUL AUG SEP OCT NOV DEC

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 [✓ 26] 27 28 29 30 31

Writeup 2A:

Class HashTable:

* Properties:

Table : Array.

Capacity :- Initialize the original capacity of the table.
- Double if extra space is needed

Number of entries :- Number of element added to a table

Number of available slot :- Number of slots unoccupied.

* Functions:

- Hash-function : Use hash function from Python
- Compression : - Compress hash of python into size N.
  - Problem: Hash value can be a large integer
  - Solution: Mod it with a large prime

- load_balancing: Check the ratio of entries /capacity

- resize : Double the capacity of current table if load_balance cross the threshold.

- get_item :- Retrieve the item in the table.
  - Return None if not found
- set_item :- Add an item to the table, in the hashed index
  - Create a new Bucket in the hashed index if necessary
  - Check if resize is needed before adding item

- del-item : - Delete an item in the table.
  - Raise Error if key not found.

- Shape_of_table :- Print out all the buckets

Class BucketList:
- List of Item: Array.
- (key, Value) : Tuple.
- Bucket_List_size : int

Function:
- get_item (key) : - Get the item in the bucket list if exist.
- set_item (key, value) : - Set the key, value pair into the bucket list
- dele_item (key) : - Delete the k-v pair in the Bucket based on Key.
- get_all (self) : - Return all element in Bucket.