CSCI-SHU 210 Data Structures

Recitation12 Worksheet Maps (Dictionary) and Hash Tables

You have a series of tasks in front of you. Complete them! Everyone should code on their own computer, but you are encouraged to talk to others, and seek help from each other and from the Professor/TA/LAs.

Our goals for this week:

- What is Map ADT
  - Support (key, value) pairs.
- Understand hash methods.
  - Value ---------> index
- Understand collision handling methods.
  - Linear probing h(x), h(x) + 1, h(x) + 2, h(x) + 3 …
  - Quadratic probing h(x), h(x) + 1, h(x) + 4, h(x) + 9 …
  - Separate chaining make a list at each cell
  - Double Hashing h(x), h(x) + h'(x), h(x) + 2h'(x), h(x) + 3h'(x) …
- For each collision handling, can apply insert/delete by hand.
- Understand when HashTables data structures are appropriate.
  - Look up, insert, delete by value expected O(1)
  - No sorted order
- Understand what is "**expected O(1)**"
  - The more memory you use, the closer to O(1)
- Can use HashTables/(Dictionaries) to solve problems.

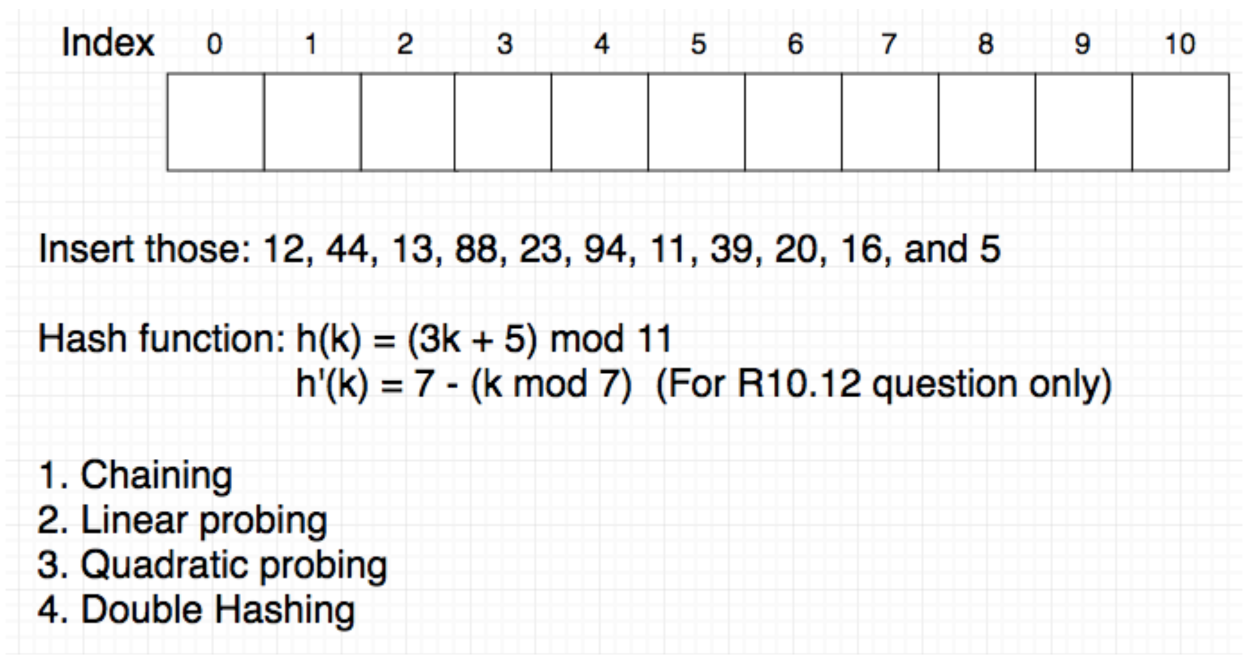
Part A: Drawing

. **R-10.9** Draw the 11-entry hash table that results from using the hash function, *h*(*i*) = (3*i*+5) mod 11, to hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, assuming collisions are handled by chaining.

. **R-10.10** What is the result of the previous exercise, assuming collisions are handled by linear probing?

. **R-10.11** Show the result of Exercise R-10.9, assuming collisions are handled by quadratic probing, up to the point where the method fails.

. **R-10.12** Show the result of Exercise R-10.9, when collisions are handled by double hashing using the secondary hash function h'(k) = 7 – (k mod 7)?



Insert those: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5

Hash function: h(k) = (3k + 5) mod 11
h'(k) = 7 - (k mod 7) (For R10.12 question only)

1. Chaining
2. Linear probing
3. Quadratic probing
4. Double Hashing

Figure 1: Problem outline for Part A: Drawing.

R-10.9 chaining

Index	0	1	2	3	4	5	6	7	8	9	10
	13	94				44			12	16	20
		39				88			23	5	
						11					

R-10.10 Linear Probing

Index	0	1	2	3	4	5	6	7	8	9	10
	13	94	39	16	5	44	88	11	12	23	20

R-10.11 Quadratic Probing

Index	0	1	2	3	4	5	6	7	8	9	10
	13	94	39	11		44	88	16	12	23	20

Insert those: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5

Hash function: h(k) = (3k + 5) mod 11

$h_0(5) = 20 \% 11 = 9$

$h_1(5) = (9+1) \% 11 = 10$

$h_2(5) = (9+4) \% 11 = 2$

$h_3(5) = (9+9) \% 11 = 7$

$h_4(5) = (9+16) \% 11 = 3$

$h_5(5) = (9+25) \% 11 = 1$

$\boxed{\text{5 cannot be inserted.}}$

$h_6(5) = (9+36) \% 11 = 1$

$h_7(5) = (9+49) \% 11 = 3$

$\vdots \quad 7$

2

Double hashing

Index	0	1	2	3	4	5	6	7	8	9	10
	13	94	23	88	39	44	11	5	12	16	20

Insert those: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5

Hash function: $h(k) = (3k + 5) \bmod 11$
$h'(k) = 7 - (k \bmod 7)$ (For R10.12 question only)

$(h(k) + f(i)) \% N$, $\Rightarrow$ when $h(k)$ position is occupied

$f(i) = i \cdot h'(k)$, $i \geq 1$

Part B: Understanding textbook code

Python build in:

• MutableMapping	Abstract class. This marks that we are going to implement five methods. (__delitem__, __getitem__, __iter__, __len__, __setitem__)

Five files:

• Map_base.py	Abstract Map ADT, class Item is defined here.
• Unsorted_table_map.py	O(N) Map ADT, using python list.
• Hash_map_base.py	O(1) Map ADT, using hashing.
• Chain_hash_map.py	Handle collisions using separate chaining
• Probe_hash_map.py	Handle collisions using linear probing

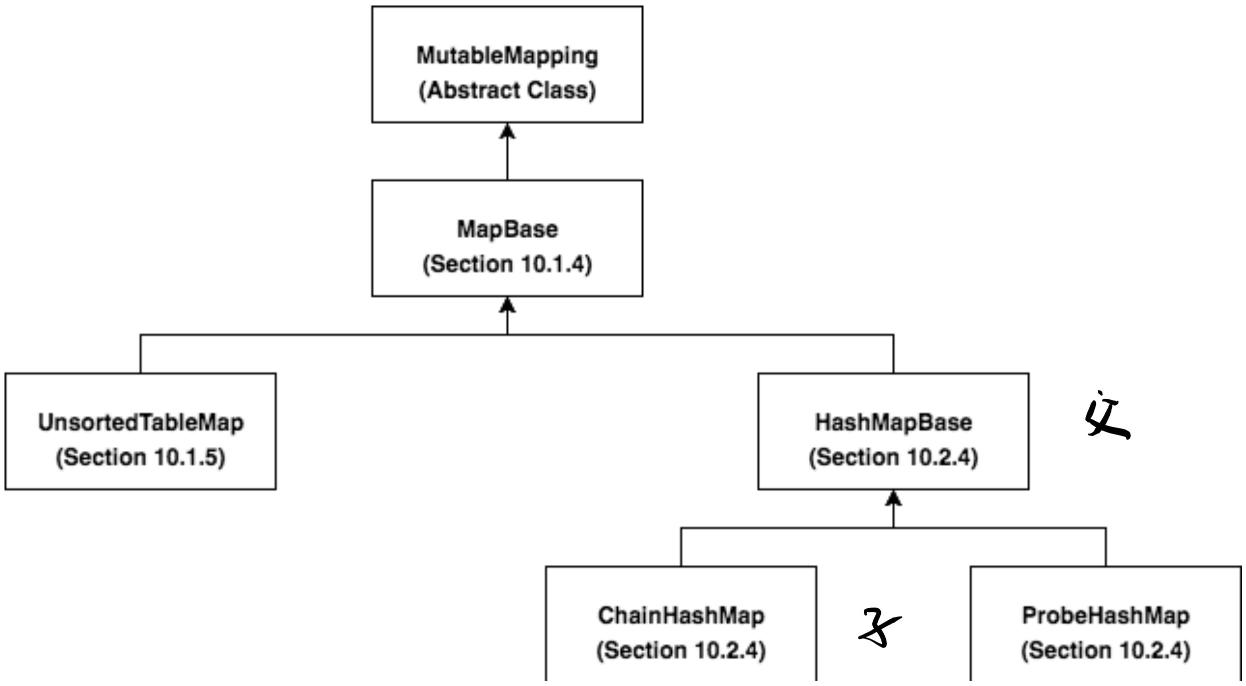
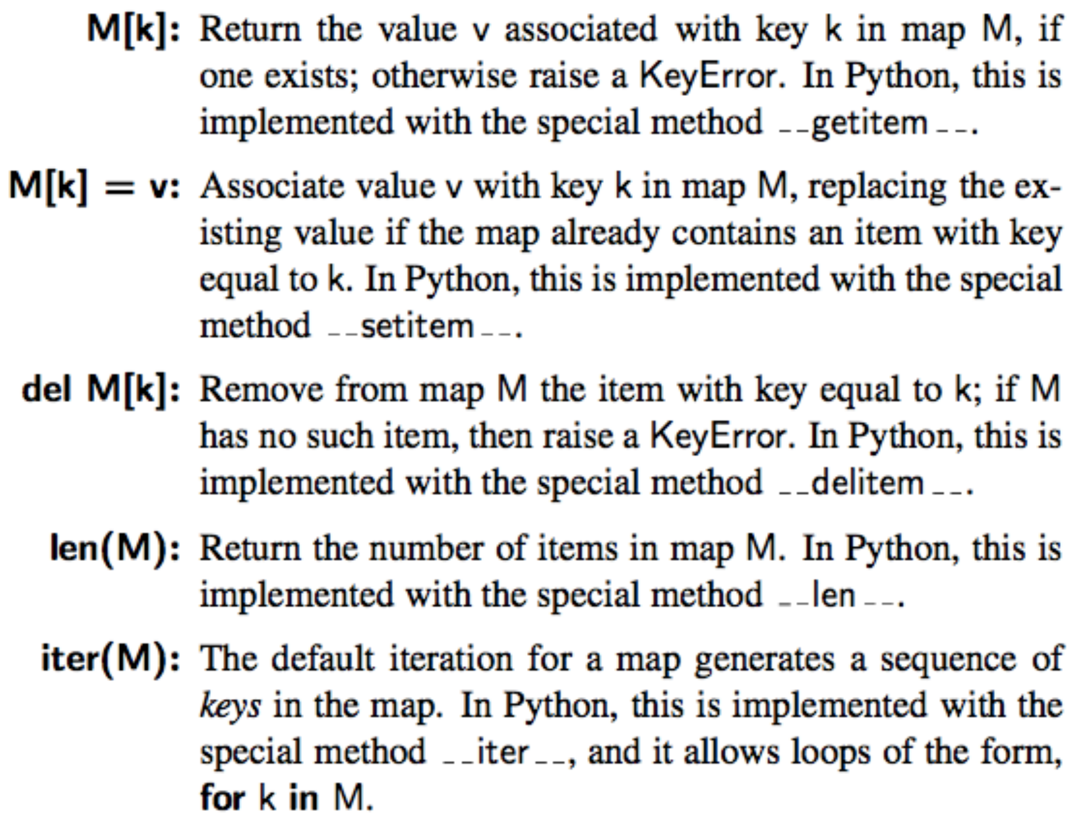


Figure 2: Class diagram for given five files, MutableMapping is defined in python standard library.

Part C: Modifying textbook code

Let’s recall the Map Abstract Data Type:



M[k]: Return the value v associated with key k in map M, if one exists; otherwise raise a KeyError. In Python, this is implemented with the special method __getitem__.

M[k] = v: Associate value v with key k in map M, replacing the existing value if the map already contains an item with key equal to k. In Python, this is implemented with the special method __setitem__.

del M[k]: Remove from map M the item with key equal to k; if M has no such item, then raise a KeyError. In Python, this is implemented with the special method __delitem__.

len(M): Return the number of items in map M. In Python, this is implemented with the special method __len__.

iter(M): The default iteration for a map generates a sequence of *keys* in the map. In Python, this is implemented with the special method __iter__, and it allows loops of the form, **for** k **in** M.

```
Your task 1: Modify hash_map_base.py


Our HashMapBase class maintains a load factor λ = 0.5.


Can you locate this code? Modify load factor λ to 0.66.
```

```
Your task 2: Modify Probe_hash_map.py. This file currently uses linear
probing to handle collisions.


After modification, your hashing should become quadratic probing instead
of linear probing.



Your task 3: Modify Hash_map_base.py.

In Hash_map_base.py, find out the hash function, what type of compression
 method is it using?
```

it's now using MAD (multiply, add, divide)

```
Change the compression method using Division method.

Recall Division method:

                       Index = k % N

Recall from our slides:

Hash method:
  h1: keys → integers

Compression method:
  h2: integers → [0, N - 1]
```

Part D: Coding & Problem solving

1. Find whether a list is subset of another list

Given two python lists: L1[0..m-1] and L2[0..n-1]. Find whether L2 is a subset of L1 or not. Both the python lists are not in sorted order. It may be assumed that elements in both lists are distinct.

```
Your task 4: implement function l1_contains_l2(l1, l2) in contains.py.


This function checks if every element in l2 exists in l1.

Use ProbeHashMap(Linear probing HashMap) to solve this problem.


Assume all elements in l1, l2 are unique.
```

2. Count word frequency.

```
Your task 5: Complete count_word_frequency.py.


This program counts words in a given text file, then output the most
frequent word and its frequency.
```

3. Union and Intersection of two Linked Lists

Given two python lists, create **union** and **intersection** lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn't matter.

Example:

```
Input:
   List1: [10, 15, 4, 10]
   lsit2: [8, 4, 2, 10]
Output:
   Intersection List: [4, 10]  (Order doesn't matter.)
   Union List: [2, 8, 20, 4, 15, 10]  (Order doesn't matter.)
```

```
Your task 6: implement function union(l1, l2) in union_intersection.py.


This function returns a new list, that is the union of l1 and l2.
Use Python dictionary to reduce runtime.
Assume all elements in l1, l2 are unique.
```

```
Your task 7:
implement function intersection(l1, l2) in union_intersection.py.


This function returns a new list, that is the intersection of l1 and l2.
Use Python dictionary to reduce runtime.
Assume all elements in l1, l2 are unique.
```