Chapter 13, Sections 13.2 – 13.5

We will discuss:

- Hash tables, hashing
- `dict` in Python
- `map` and `unordered_map` in C++
- Collision resolutions

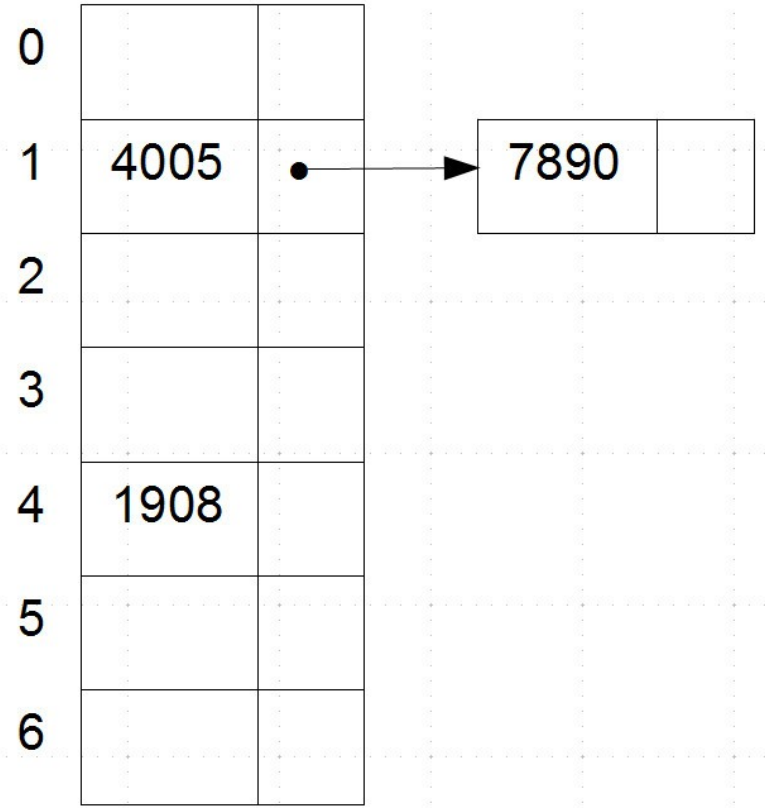
Collision Resolution – separate chaining

Consider a hash table with 7 positions that holds some personal records. The key to each record is the **last 4 digits of customer's SSN. The hash function f is given by:**

f(k) = k%7,

produces the index of the slot in the array for the key k.

The method of collision resolution is separate chaining. Draw the boxes and arrows in the following diagram to give the state of the hash table after the following keys are used to insert records in the order:
4005, 1908, 7890, 1928, 0035, 1076, 0187, 1098, 7777, 1108, 0089, 1625.
You can see the first three insertions have already been made as a hint.



Collision Resolution – double hashing

Below is an array with 15 positions, which is used as a hash table to keep some IDs. The key to each record is the 3-digit customer's ID.
The hash function h gives the index of the slot in the array for the key k: h(k) = k%15.
The method of collision resolution is double hashing. Hence, if collision happens, we repeatedly compute (h(key) + ih2(key)) % 15, for i from 1 to 15, and h2(key) = key%7 until an empty position is found (for adding an item to the hash table), or the key is matched (for retrieving an item from a hash table, given a key).
Add the following keys to the hash table and show your calculations:
17, 42, 189, 24, 120, 78, 114, 146, and 282. The first four insertions are already made:

24		17							189			42		