8. Hashing with Chaining

Link: https://www.youtube.com/watch?v=0M_kIqhwbFo

Dictionary: Abstract Data Type (ADT) maintain set of items each with a key

- Insert (item)
- Delete (item)
- search(key): return item with given key or report doesn't exist (key error in Python)

O(lg n) via AVL
Best search (logn) and best sort is (nlogn)

Use of dictionaries:

- Dictionaries are used in document distance problems where you need to find word count or word differences between two documents.
- Hash databases
- Spell Checkers
- Compilers and interpreters (old ones use dictionary to store variables and their values)
- Network router
- Substring search
- String commonalities
- Files and directories sync
- Cryptograhy

Simple approach:

- Direct access table
    - Store items in array indexed by key

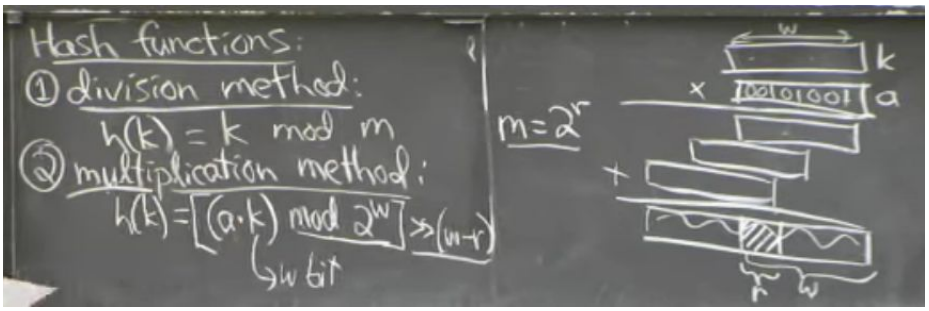
Downsides of Hash Tables and Solutions:

- Keys may not be nonneg. Integers
    - Solution
    - Pre-hashing (Python calls it hash)
        - Maps keys to nonneg. Integer
        - In theory, keys are finite and discrete
            - Immutable objects such as integers, not lists
            - (string of bits)
        - In Python, hash(x) is the prehash of x
        - Ideally, hash(x) = hash(y) ⇔ x=y
- Gigantic memory hog
    - Solution
    - Reduce universe of all keys (integer) down to reasonable size m for table
    - Idea: m = O(n) (n being the amount of keys in dictionary)
    - Place all keys into a key space and map their location to a hash table

    - However, doing so may lead to collisions as multiple keys attempt to take up the same slot in the hash table
        - Solution
        - Chaining: linked list of colliding elements in each slot of hash table
            - Item will be a pointer to a linked list
            - Worst case: O(n)
            - Simple uniform hashing (unrealistic and based on false assumption) - each key is equally likely to be hashed to any slot of the table, independent of where other keys hashing
            - Analysis of SUH: expected length of chain for n keys, m slots
                - n/m = α = load factor
                - $\Theta(1)$ if m = $\Theta(n)$
                - Running time = O(1+|chain|) = O(1+α)
                -

- __hash__ allows you to make a custom function
- By default, Python uses id which is the physical location of your object in memory

Hash functions

- Division Method: h(k) = k mod m
    - No good if k and m has common factors
    - If both are even, it will only use half the table skipping odd positions
    - M should always be prime that is not near a power of 2 or power of 10
- Multiplication Method: h(k) = (a * k) mod 2**w >> (w-r)
    - Variable a should be random, odd, and not close to a power of 2
    - 

- Universal hashing: h(k) = [(ak + b) mod p] mod m
    - a = Random
    - b = t {0...p-1}
    - p = Prime > |U| (bigger than your universe)
    - For worst case keys k1 != k2:
        - Probability {h(k1) = h(k2)} = 1/m