Hash Tables

Thursday, January 16, 2025 9:20 AM

A collection of slots/buckets, each with an address (like index)

Ex.
- 10, cat
- 20, dog
- 15, bird
- 2, snake

Table size: m

h(k) = k mod m

λ load factor = n/m

0	
1	
2	(20, dog), (2, snake)
3	(15, bird)
4	(10, cat)
5	

Why do we keep the int key?
- There will be lots of values modded with 6 that produce the same result

There is a constant amount of work for any k value

When looking for what animal is associated with '2':
1. we mod 2 with 6
2. go to that location
3. then search for the key '2'
    - This is a linear search

NumPy array of python lists of key value pairs

How big should we start our table?
- Short answer: quite large
- Table size -> 100k
- Use λ load factor to determine if it's too big
    - λ < 0.9
    - Constantly check for this
- Maybe increase table size to 1M
    - You have to make and re-insert everything to the table
- The more spaces we have, the quicker the linear searches will be (if you have a good hashing function)

Assumption: Largest chain 5 KV pairs
- "Essentially" constant time if you can give a probabilistic guarantee that the chains don’t exceed a certain length

When inserting, make sure that the item is not already in that chain/list
Since dispersion is complex, use an already implemented hashing function

(finance, 7.json)
(money, 10.json)
(bank, 15.json)
(market, 7.json)

Put 'finance' through a hash function
This will give you a 32-bit int
- No need to store this because we can just put the word that we are searching for through the hash function again

(381 finance, 7.json)
(767 money, 10.json)
(951 bank, 15.json) … mod with 10, check if already in there, insert
(767 money, 7.json) … here we would find money, so just add 7.json to the Json list

381 mod 10 = 1

0	
1	(381 finance, 7.json), (951 bank, 15.json)
2	
3	
4	

5	
6	
7	(767 money, (10.json, 7.json))
8	
9	