Hashing

Dr. Baldassano
chrisb@princeton.edu
Yu's Elite Education

Last week recap

- Linked lists
  - Advantages/Disadvantages over arrays
  - Singly vs. doubly-linked
- Queues
  - FIFO
- Stacks
  - FILO

Homework: Find list median

Associative arrays

- Regular array/list: maps nonnegative integer keys onto values, e.g. states[3] = ‘NJ’
- Associative arrays: map any key onto values, e.g. capitals[‘NJ’] = ‘Trenton’
  - Want to be able to add, delete, and modify each key/value pairing
  - Called dictionaries in Python
  - Can also be used without values (sets)

Associative arrays

- How to implement associative arrays?
- Can just store a list of key-value pairs

Index	Key	Value
0	NJ	Trenton
1	NY	Albany
2	PA	Harrisburg

- Big-O for add, delete, find, modify?
  - Add: O(1)
  - Find, modify, delete: O(N)
  - Memory: O(N)

Speeding up find and delete

- Goal: Find items in O(1) time, without increasing memory requirement too much
- Strategy: Convert every key to a nonnegative index, then use a regular array
- Example:
  - Initialize capitals to be empty length-10 array
  - When adding ‘NJ’, plug ‘NJ’ into a function f(‘NJ’) = 4
  - Store ‘Trenton’ in capitals[4]

Hash function

- The function converting keys into indices is called the *hash* function
- Input: Keys (may be any type)
- Output: Nonnegative index where we should store the value of that key
- **Ideally, we want all keys to be mapped to different indices**

Example: Website logins

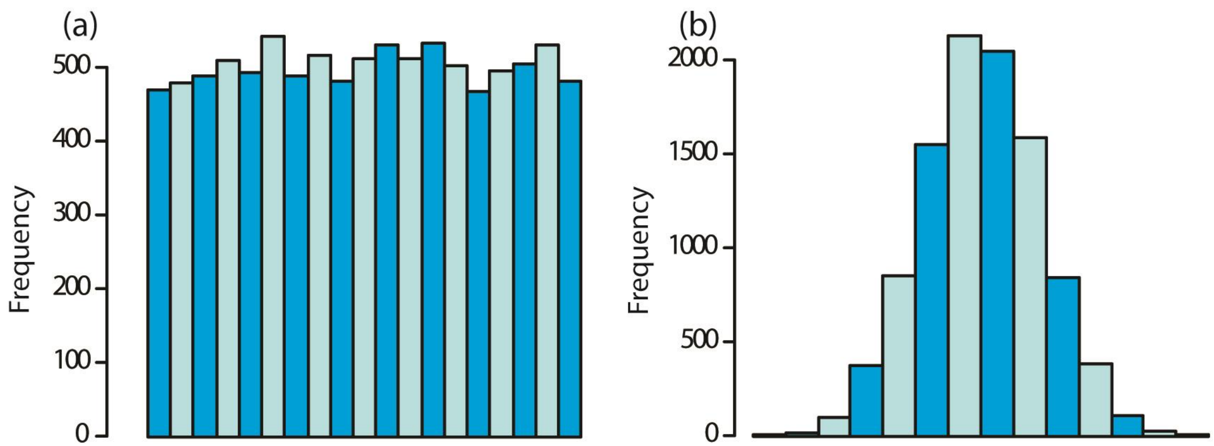
- Four users sign up for website:
  - Praneel, Areeq, Jimmy, Allen
  - f(Praneel) = 5, f(Areeq) = 2, f(Jimmy) = 0, f(Allen) = 3
- Store their passwords in hash table

Big-O for hash table

- If hash function successfully maps all keys to different indices, then:
  - Add: O(1)
  - Find: O(1)
  - Modify: O(1)
  - Delete: O(1)
- What's the catch??
- Will need to use extra memory: for good hash function, memory should still be O(N)

Designing a hash function

- When picking a hash function f, we want it to distribute keys uniformly over the array

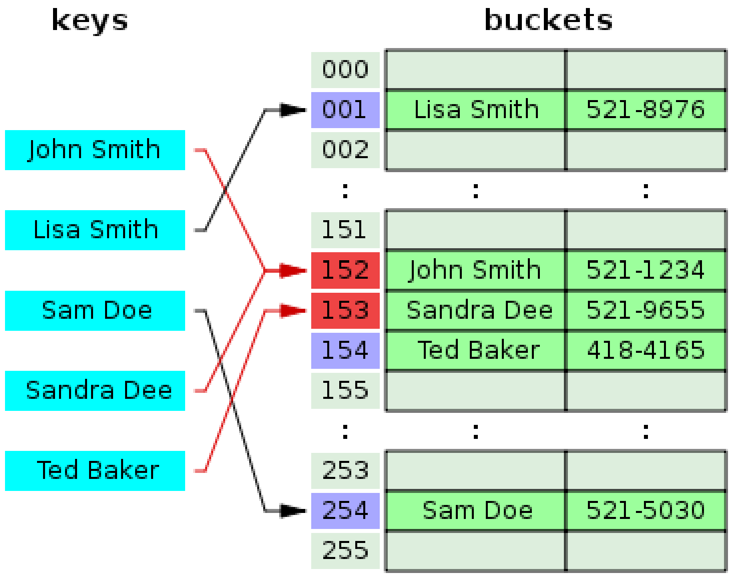


Collisions

- As long as our array is smaller than the total number of possible keys (e.g. all possible usernames), there will always be *collisions*
  - Collision occurs when f(key1) = f(key2)
- Collisions more likely to happen if:
  - Array is not big enough
  - Hash function isn't uniform
- What do we do when we go to add a key/value but another key is already there?

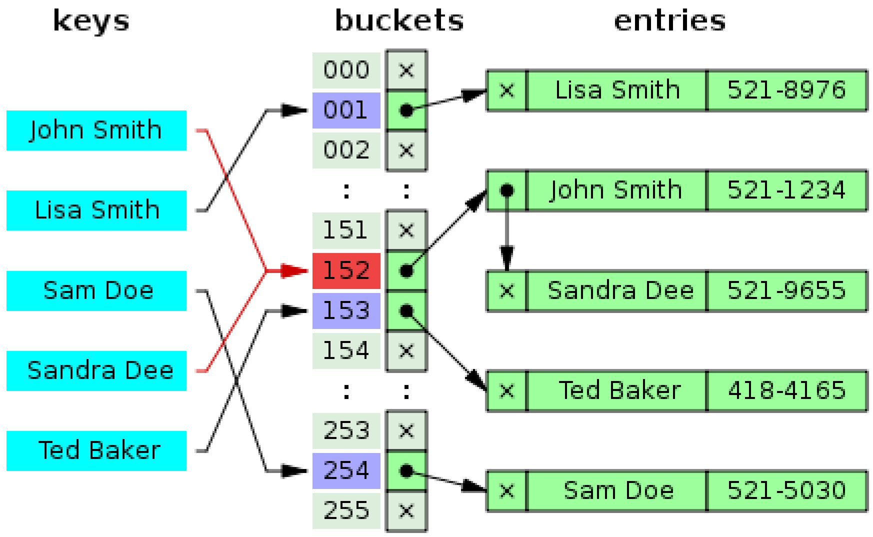
Dealing with collisions: Open Addressing

- If there is a key/value in your spot, just use the next open spot



Dealing with collisions: Chaining

- Chaining: Each array bucket contains a list of all key/values mapped to that bucket
- Usually used a linked list



Dealing with collisions

- Open addressing easiest when keys are small and have at least twice as many memory slots as keys
- Otherwise, chaining better:
  - Only stores pointers in array
  - Can have more keys than memory slots
  - Can handle variable-sized data

Python hashing example

Olympiad Problem

- Censoring: http://www.usaco.org/index.php?page=viewproblem2&cpid=533

Hashes in cryptography

- Often we want to prove that two pieces of data match, *without* looking at them
- Very common example: passwords
  - Websites/computers want to check if the password you're entering now matches the one you signed up with
  - BUT don't want to just store a copy of password, otherwise a hacker/insider could mass-copy passwords out of the database

Hashes in cryptography

- Solution: only store a *hash* of the password
- A good hash function will rarely have collisions – so if password hashes match, passwords almost certainly match!
- Cryptographic hashes also designed to make sure they are hard to reverse (hard to get from hash to password)

Attacking a hash

- Let's say we lost our password and only have the hash (or we're doing something evil)
- How can we get password just given the hash?
- Could just try every possible password, and store all hashes
  - This will take a looong time, but we only ever have to do it once
  - Then we can use this table many times for this hash function
  - Called "Rainbow Tables"

Python hash attack example

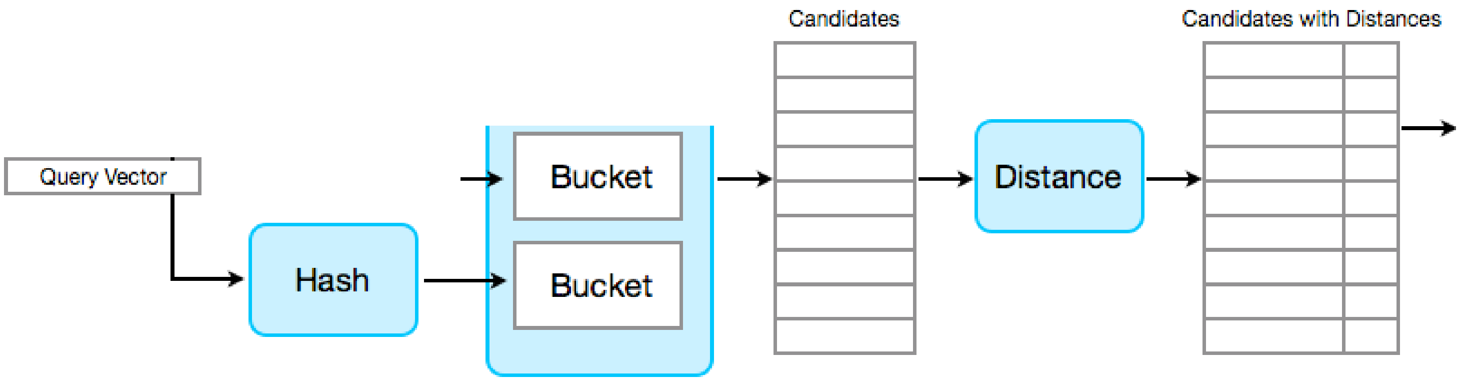
“Opposite” kind of hashing

- For hash tables, we want to avoid collisions
- When might we *want* collisions?
- Main use: detecting nearest neighbors
  - If we map similar values to the same bucket, hashing will find close neighbors
- Called *locality-sensitive hashing*

Examples of LSH

- Finding near-duplicates
  - Detecting plagiarism
  - Finding other sizes of image
  - Avoid duplicate search results
- Recommender systems
  - Find similar customers, see what they bought
  - Find similar movies to your favorites

Finding closest neighbors with LSH



LSH for detecting breaking news on Twitter

Incoming Tweet	Closest previous Tweet	Similarity Score
@Real_Liam_Payne i wanna be your female pal	i. wanna be your best friend so follow me :)	0.385
RT @damnitstrue: Life is for living, not for stressing.	RT Life is for living, not for stressing.	0.99
East Timor quake leaves Darwin shaking: An earthquake off the coast of East Timor	Everybody leaves eventually	0.129

Assignment: Anagrams

- Given a dictionary http://www.codeabbey.com/data/words.txt divide it into groups of anagrams
- Example: tea, asleep, plus, ate, please
  - [tea, ate], [asleep,please], [plus]