Hash Tables II

AGENDA

- Intro to Hash Tables Review
- Collisions
- Load Factor & Resizing
- More Whiteboarding Problems (maybe)

Intro to Hash Tables Review

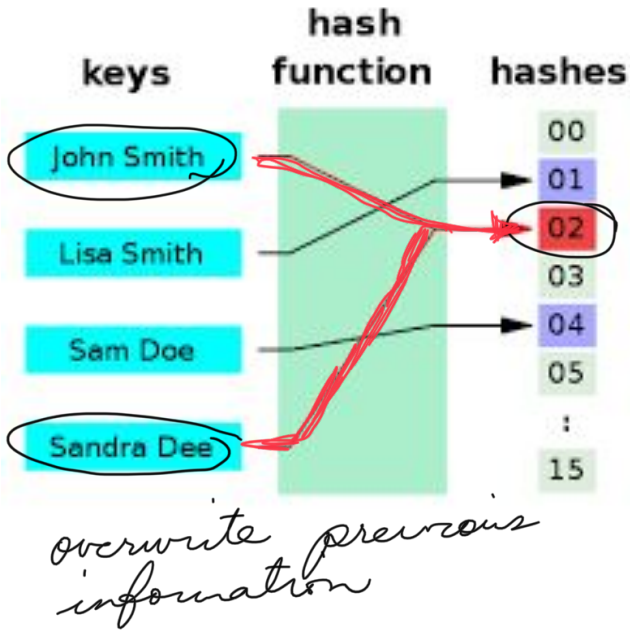
- deterministic
- fast
- sufficiently randomized output language

Collisions

COLLISIONS

- What happens if you try to store values that have the same index?
- Based on our last example you would overwrite the values!
- How can we solve this?

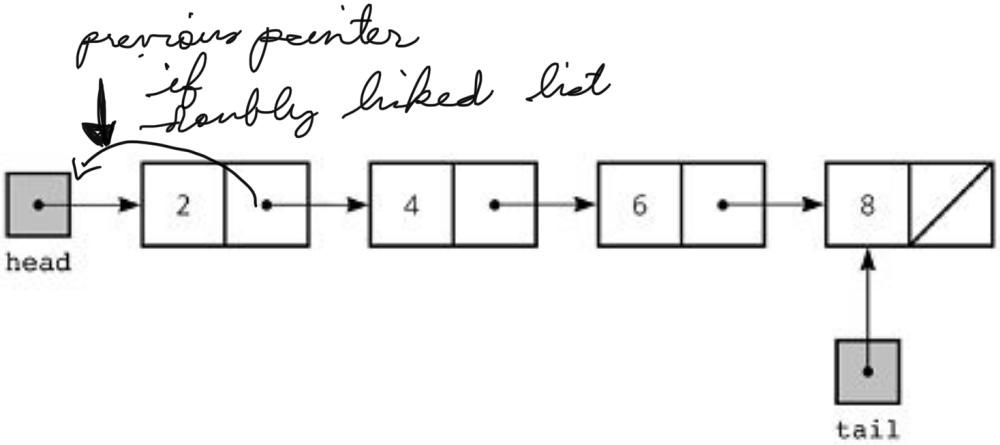
without collision function if two keys hash to the same index 2nd will override previous information



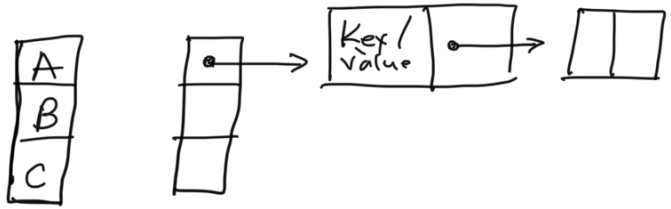
Collisions Demo I

LINKED LIST REVIEW

- Comprised of nodes
- Nodes contain: *value*, *next pointer*, *previous* (if doubly linked-list)
- To fully implement a hash table, we're combining three things: *hashing*, *arrays*, *linked lists*

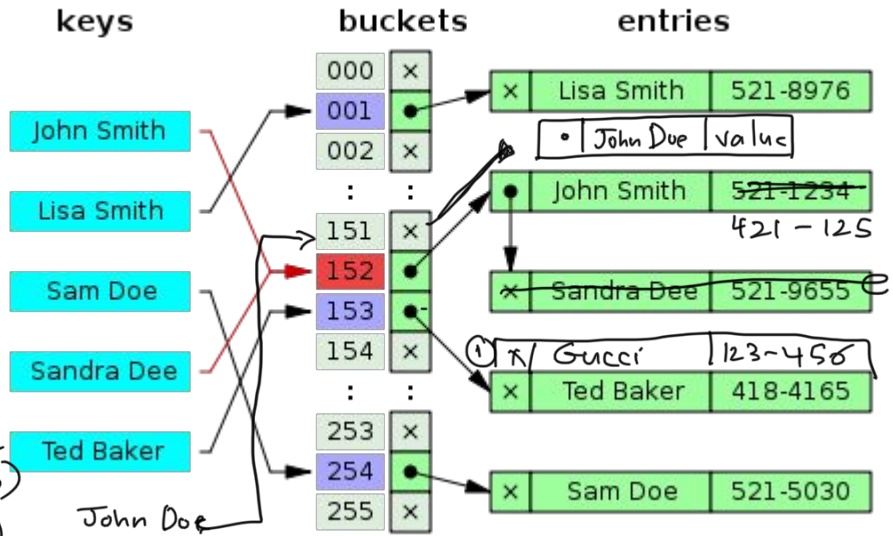


COLLISION RESOLUTION VIA CHAINING



- Operations of a hash table get slower if there are a lot of collisions
- To solve collisions, chain values together by using linked lists
- If a value already exists at that index, add the new item to the linked list

store both key and value because each row can hold more than one value



Examples
get ("John Smith")
store ("Gucci", 123-456)
delete ("Sandra Dee")
store ("John Smith" 421-125).

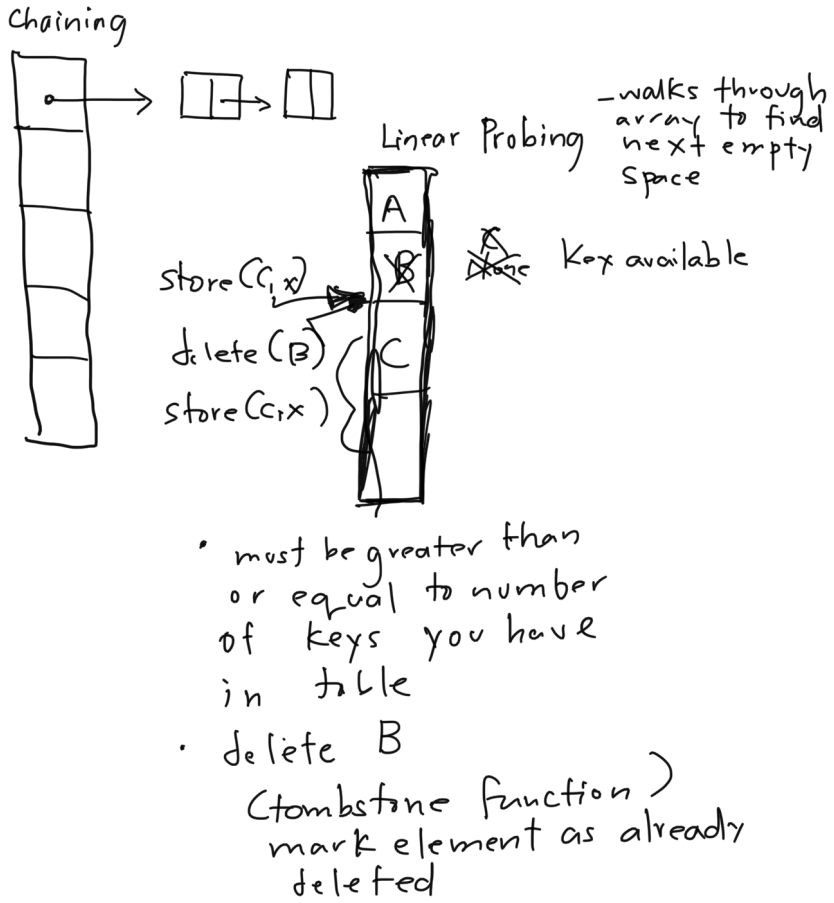
Gucci
collides with Ted Baker
initialize new Node
add to head/tail

Each node is dynamically added as collisions occur.

Collisions Demo II

COLLISION RESOLUTION VIA LINEAR PROBING

- Keys are stored in the table, instead of storing them in a linked list
- Size of table is >= number of keys
- If a collision occurs, walk the table until you find a free spot
- Tombstone an element when it's deleted



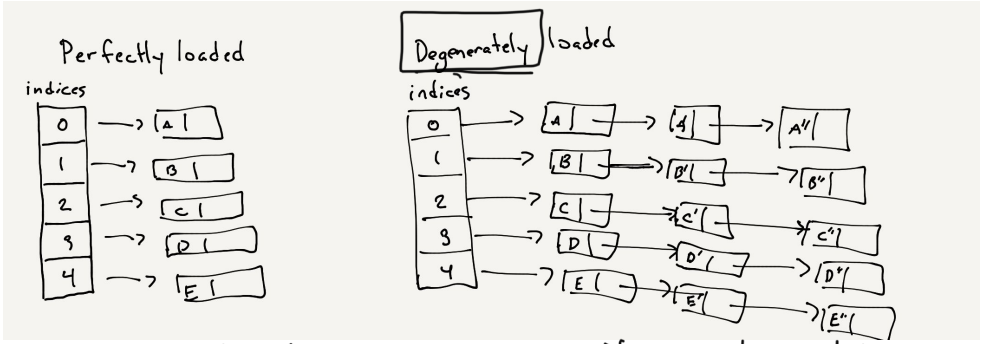
CHAINING VS. LINEAR PROBING

- Linear Probing has the following benefits:
  - *Low memory overhead* - just need an array and a hash function
  - *Excellent caching performance* - by taking advantage of arrays and cache locality
- Linear Probing has the following drawbacks:
  - *Gets very slow* as load factor approaches 1 (why do you think that is?)
  - You can *never have more elements than the size of your array*
- For the guided project you should use *chaining* as the collision resolution method

Load Factor & Resizing

LOAD FACTOR & RESIZING

- The performance of a hash table worsens as there are more collisions
  - Can you think of why?

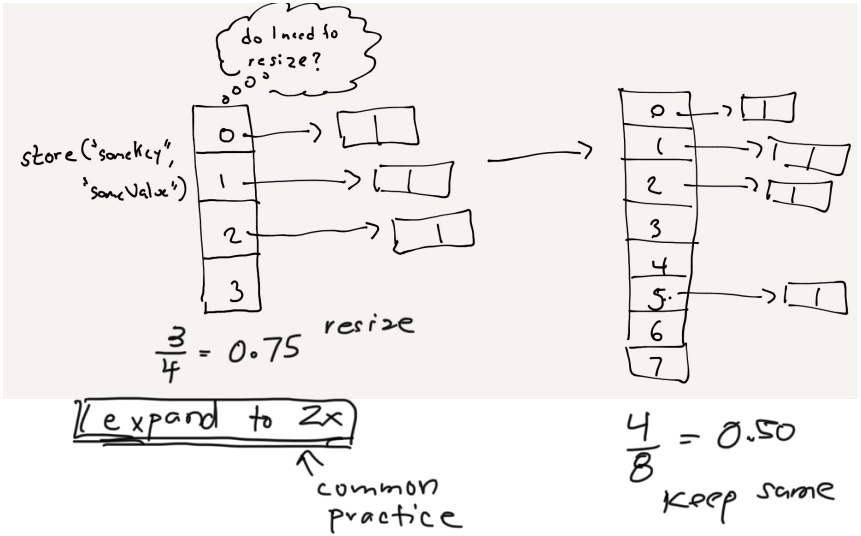


exactly loaded w/ 1 item

- so many items chained in linked list
- bad hashing function so every thing goes to 1 bucket

LOAD FACTOR & RESIZING

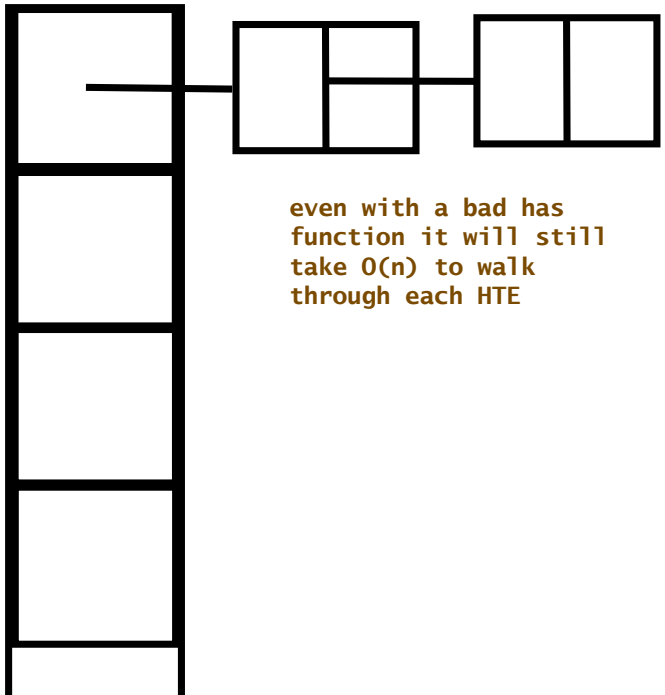
- To prevent this from happening, we need to know when to increase the size of our table
- Use *load factor* to determine when to expand/shrink
- *Load factor = num elements / num slots*
  - Expand table if load factor > 0.7
  - Shrink if < 0.2



Load Factor & Resizing Demo

HASH TABLES TIME/SPACE COMPLEXITY

- What's the time/space complexity of get/store/delete in a degenerately loaded hash table? Perfectly loaded hash table?



even with a bad has function it will still take O(n) to walk through each HTE

	Average	Worst
Get	O(1)	O(n)
Store	O(1)	O(n)
Delete	O(1)	O(n)

HASH TABLES TIME/SPACE COMPLEXITY

- Make sure you can explain why!

	Average	Worst
Get	O(1)	O(n)
Store	O(1)	O(n)
Delete	O(1)	O(n)

TODAY’S ASSIGNMENT

- Implement a hash table with collision resolution via chaining and resizing based on load-factor

More Whiteboarding Problems

HASH SET

- Hash tables are used in common data structures (sets, dictionaries, etc.)
- Leetcode Link

Hash Set Demo