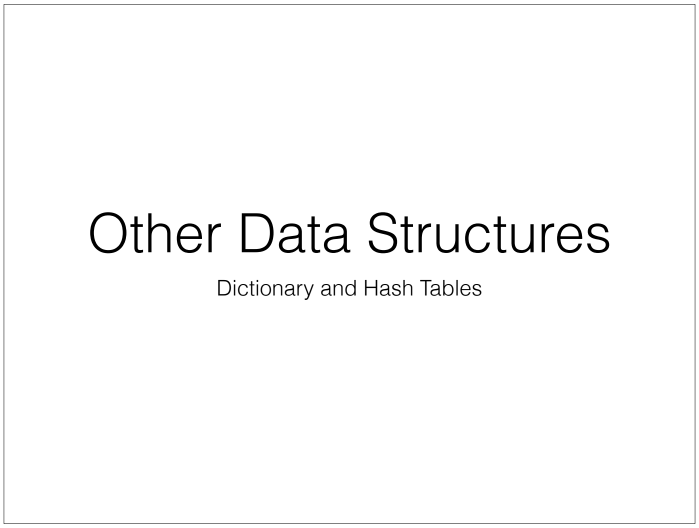
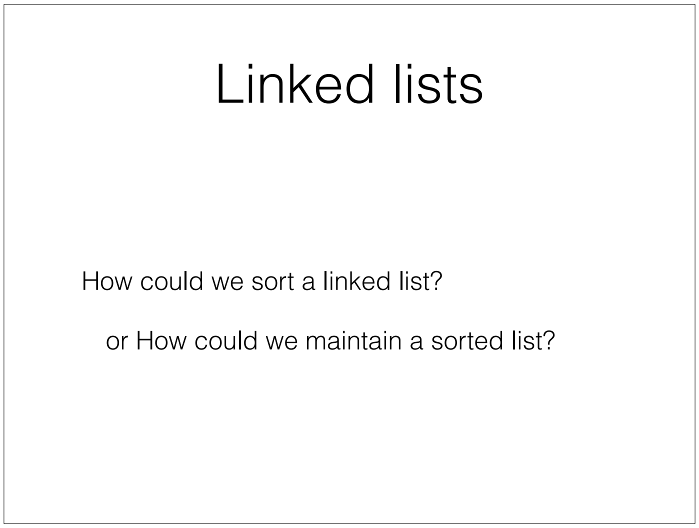


Other Data Structures

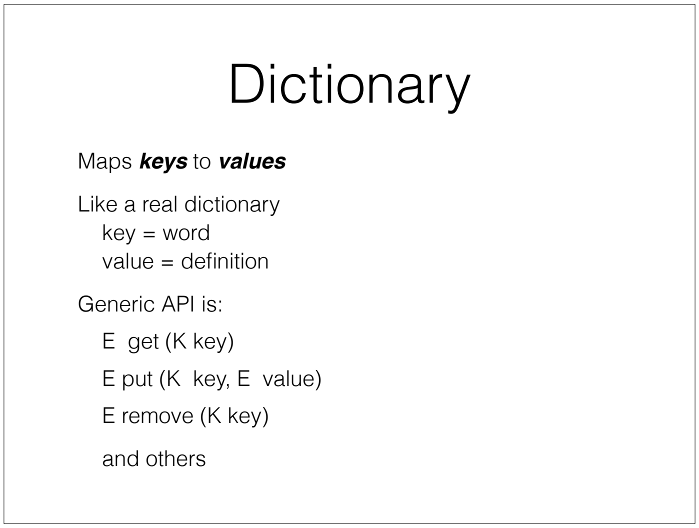
Dictionary and Hash Tables



Linked lists

How could we sort a linked list?

or How could we maintain a sorted list?



Dictionary

Maps ***keys*** to ***values***

Like a real dictionary
key = word
value = definition

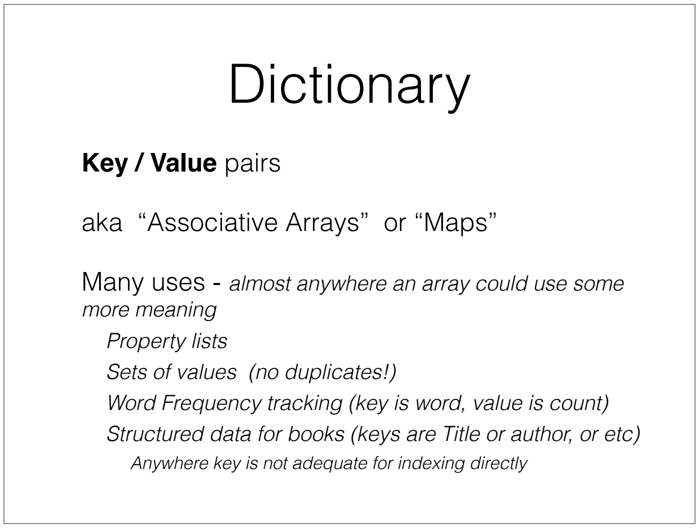
Generic API is:

E get (K key)

E put (K key, E value)

E remove (K key)

and others



Dictionary

Key / Value pairs

aka "Associative Arrays" or "Maps"

Many uses - *almost anywhere an array could use some more meaning*

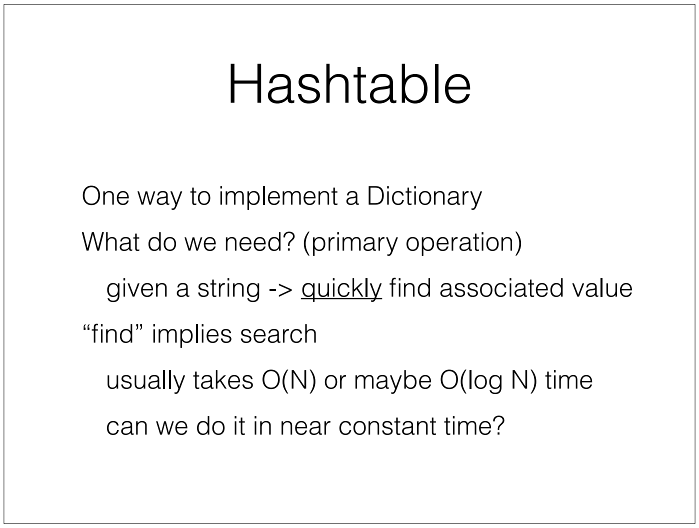
Property lists

Sets of values (no duplicates!)

Word Frequency tracking (key is word, value is count)

Structured data for books (keys are Title or author, or etc)

Anywhere key is not adequate for indexing directly



Hashtable

One way to implement a Dictionary

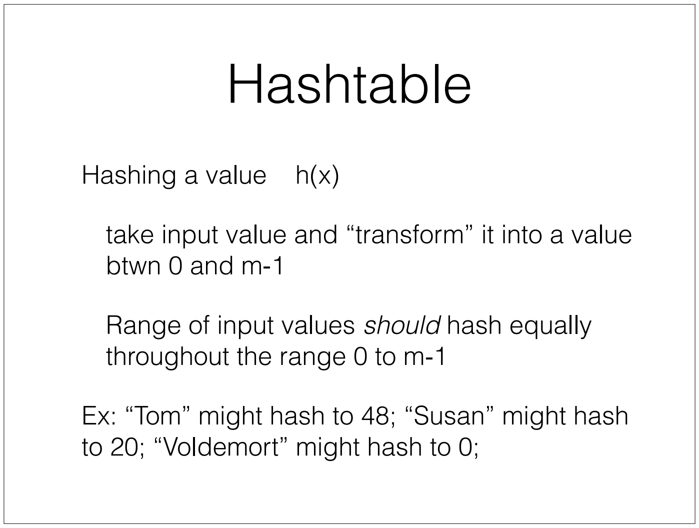
What do we need? (primary operation)

given a string -> quickly find associated value

"find" implies search

usually takes O(N) or maybe O(log N) time

can we do it in near constant time?



Hashtable

Hashing a value h(x)

take input value and "transform" it into a value btwn 0 and m-1

Range of input values *should* hash equally throughout the range 0 to m-1

Ex: "Tom" might hash to 48; "Susan" might hash to 20; "Voldemort" might hash to 0;

Hashtable

Once you have a hash function: h(x)

How do we use that:

- Create *hashTable* as an array of size m
- As the value with key x is stored at location: h(x)
- So: Tom’s info is stored at   hashTable[48]
  - Voldemort’s is stored at   hashTable[0];

What is the time to find Tom’s info?

- = time to calculate h(“Tom”)

HashTables

Challenges?

- What if two keys hash to same number? (this is called a collision)
- store in next location?
- store in location h(x) + count2
- each location could be a linked list!
- use different hash so: h(x) + h1 (x)   …

HashTables

No matter how much we try, since we have a limited sized table, it is likely that (eventually) two keys will hash to same value   h(x) = h(y)

To be efficient, HashTables must to be larger than needed (i.e. must have many empty spaces)

- “load factor”   *keep it < 50%*
- Tradeoff: Time -vs- Space

HashTables

Challenges?

- What if the hashTable fills up?
  - ***rehashing*** - build bigger, then re-add all values
  - O(n) time: since O(n) + O(1)*n
- How can we print all values?
- What about sorting the values?

Hash Function Example

Spread out the results

- public int hash (String s, int n)
  - v = sum of the character value of all characters
  - return (v mod n)

Does it work? “AB”   “BC” “A” “BA”

ASCII values: A:65 B:66 C:67

hash(“AB”, 100) => 131%100 = 31   hash(“BC”, 100) => 133%100 = 33

hash(“A”, 100) => 65%100 = 65   hash(“BA”, 100) => 131%100 = 31

Tend to cluster around multiples of (65-90) Not very spread out

For large tables 10,000?

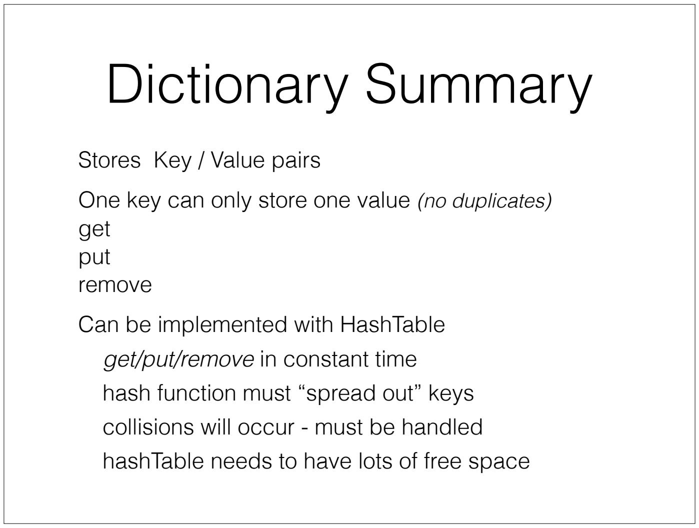
Hashing Objects

Equals

- all objects have an equals method
- HashTable can’t rely on memory address!
- equals must properly override method to work

Hash

- all objects are supposed to have a hash method too



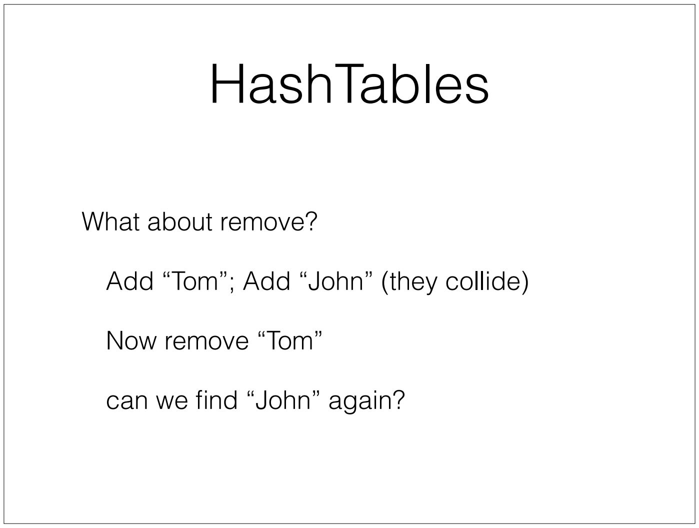
Dictionary Summary

Stores Key / Value pairs

One key can only store one value *(no duplicates)*
get
put
remove

Can be implemented with HashTable

- *get/put/remove* in constant time
- hash function must “spread out” keys
- collisions will occur - must be handled
- hashTable needs to have lots of free space



HashTables

What about remove?

- Add “Tom”; Add “John” (they collide)
- Now remove “Tom”
- can we find “John” again?