Sets/Maps

Python had dictionaries: what are they useful for

Baindom

Map<K,V> {

boolean put(K key, V value)
boolean getC Object key) <- will return false if types don't match

boolean isEmpty()
boolean remove(Object key)
Set<K> keySet() <- opt

}

The country linked bag from exam was infact a Map

Set like a map, but has no value

~~How can we implement it?~~

Set and Map can be implemented the same way, set has less information.
Could use an ArrayList for both.

word count

factorial

# Maps

Map<K, V>

~~void put~~

✓ put(K, V) ← returns old value
✓ get(K)

Ex: Telephone Book or Contact List — name → phone #
Cache — input → output

Unordered List
put O(1) (at end)
get O(n) sequential search

Sorted Arraylist
put O(n) need to shift
get O(log n) binary search

Linked Sorted List
put O(n) need to find
get O(n) can't binary search a linked list

## Hashing

Idea: Use an array, but rather than sorting, the index for key k is given by a function h(k). which maps keys to ints

~~h(k) is~~ h is the hash function
h(k) is the hash code

Ex: Map names to offices

Sam → 318
Dave → 312

Convert each name to an index into an array of size 5?

Brainstorm

Idea: Sum whatif we hire Aden, Dean and Deng

First letter Dave
David

h(k) % table size

Problem: We end up with collisions ← more on this Friday

Goal: Find a good hash function

~~Int Hash~~ hash ~~values~~ codes should be unique (even mod table size)
hash codes should be easy to compute

Strings

Ex $h(s) = h(s_1 s_2 s_3 \ldots s_n) = s_1 + p s_2 + p^2 s_3 + \ldots p^{n-1} s_n$

Java uses 31

Lets look at it

Define our hash code

Other topics

Passwords

Salting
Verify against the hash result

Proof of work

Spam Prevention
Bitcoin

K X

h(K+X) < n (i.e. begins with certain number of zeros)

Bloom Filters
~~Probabilistic Data~~
Set

might contain

Recap

h(K) : a function from Keys to integers

~~Strategy : then~~

~~Strategy~~

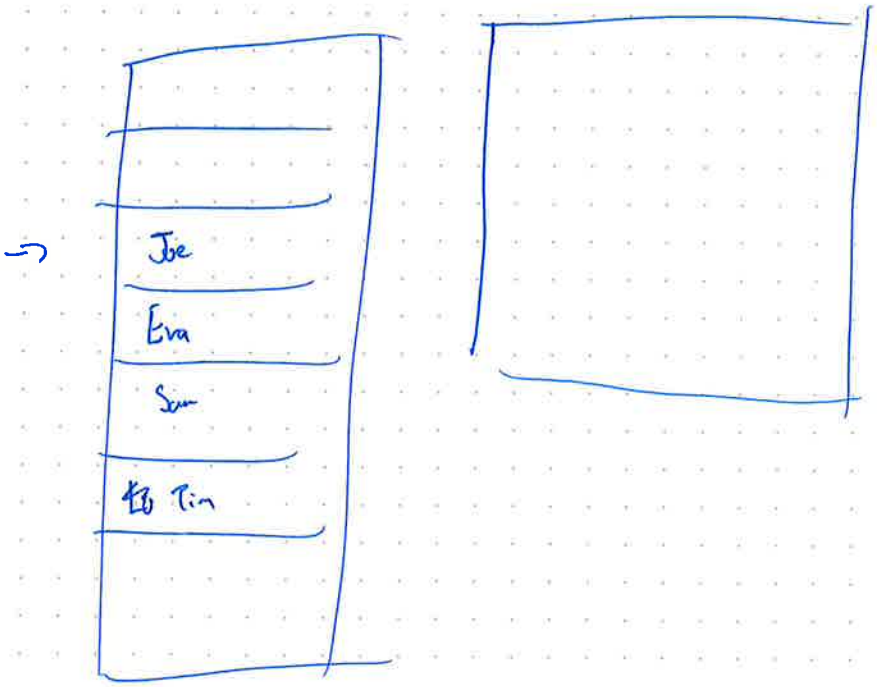
HashMap

hash function

1 Convert keys to integers using h(k)
2. Store key (and value) at location given by h(k) in an array, if that location is empty

```
Entry(K,V)[] table;

put(K,V)
    i = h(k)
    if (table[i] is empty or table[i].key = k
        table[i] = new Entry(K,V);
```



## Analysis

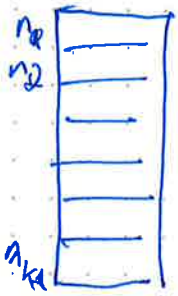
$n$ = # of entries
$K$ = size of table

$\alpha$ = load factor $= \frac{n}{K}$

Open addressing $\alpha \leq 1$

Chaining — $\alpha$ can be as large as you want

Unsuccessful search




$$\frac{n_0 + n_1 + \ldots + n_{K-1}}{K} = \frac{n}{K} = \alpha$$

$\Theta(\alpha)$

- $\alpha$: average time
- $\Theta$: worst cost

Successful search you go through half the chain, so ~~$\Theta(\frac{\alpha}{2})$~~

closer to $\Theta(\frac{\alpha}{2}) \to \Theta(\alpha)$

Open addressing

Harder analysis — what do you expect?

Assume we don't have clustering problems

$P(h(k)$ is full $) = \alpha$

Like flipping a coin with probability of heads $= \alpha$

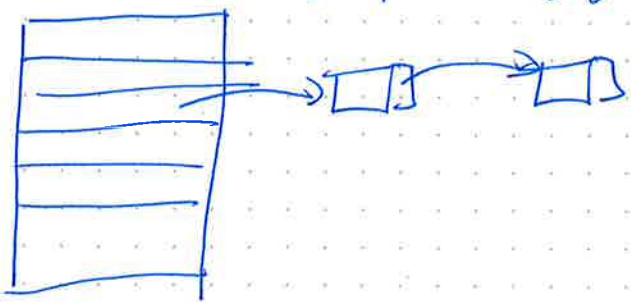
Expected number of steps is $\frac{1}{1-\alpha}$

$\Theta(\frac{1}{1-\alpha})$

With linear probing this looks more like $\Theta(\frac{1}{(1-\alpha)^2})$

~~Other~~ Chaining

Other solution is to pull up a seat anyways



Problem: There might be a long chain (i.e. people sitting at your table)

## Analysis

### Space

Just add ~~one pointer to array~~ fill in array for open addressing

no new allocations

### chaining

Constantly allocates new nodes.

all things being equal, this takes more space

### Deletion

Open Addressing hard

Chaining easy

### Table Size

~~No hard limit~~ Open Addressing hard limit

Chaining no hard limit

Both - Performance will deteriorate

Hashing

Collisions

Metaphor: Meeting friend for lunch in the dining hall

Strategy 1 - Open Addressing
Finding a predictable next spot to look
a look at nearby tables
if you search nearby tables 1 by 1 linear probing $h(k) + i$
Primary
Problem 1: Clustering

The chance of someone hitting your cluster of tables is high
so large clusters get larger more quickly

b. look at tables further and further away: $h(k) + i^2$

Problem: Secondary Clustering
Two keys with the same original hash will continue to collide.

In metaphor: people with similar tastes keep showing up

c. Double hashing $h_1(k) + i h_2(k)$

You and your friend agree to both a location to look and a interval to schedule future searches

d. Cuckoo hashing choose two locations, $h_1(k)$ and $h_2(k)$

If both locations are full, kick some one out, they continue kicking people out.

If you reach a loop, rehash everything with new hash functions
reassign locations ~~(pull fire alarm)~~
for everyone then ask everyone to be reseated

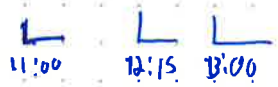
Other problem - Delete

Show up and find an empty table

Solution: leave tracy (Creates problem with cuckoo hashing)

Motivation: Appointment reservation system

Single office
Reservations for future appointments
Reserve request for appointment at time $t$
Add $t$ to set $R$ if no other appointments are within 30 minutes




$|R| = n$
$O(\log n)$ time ← target

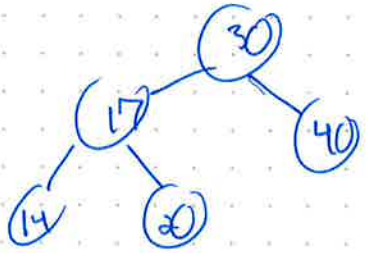
Do example

Unsorted list: Insert in $O(1)$ w/o check, but check take $O(n)$

Sorted array: Search in $O(\log n)$ - Find smallest $i$ such that $R[i] \geq t$
Do check in $O(1)$ - Compare $R[i]$ and $R[i-1]$ against $t$
Actual insertion in $O(n)$ - Need to shift

Hash Map Table: can only look up exact values - cannot do range query

Binary Search Trees




Node $x$: Key (and values)

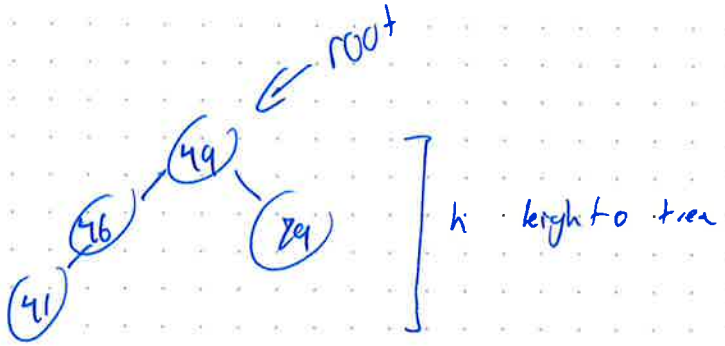
Pointers: parent
left
right ← Pics can draw!

Invariant: For all nodes $x$, if $y$ is in the left subtree of $x$ $y.key \leq x.key$

if $y$ is in right subtree of $x$ $y.key \geq x.key$

Look at example, left tree values are $< 30$
right tree values are $> 30$

Insert 49
29
46
41




$O(h)$ time

~~Ex~~ Look at code:

Traversals

Start with pre order

~~use BST to sort?~~

Print in sorted order

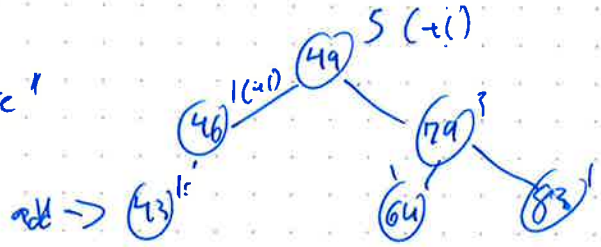
find Min() go to left until leaf $O(h)$

next Layer(t) $O(h)$

Augmentation

Rank(t) how many appointments occur before t

Store "size"




if t < key
return ~~rank~~ t left.rank(t)
t = k
return left.size of
t > k
return left.size + 1 + rank(t) right