Hashing

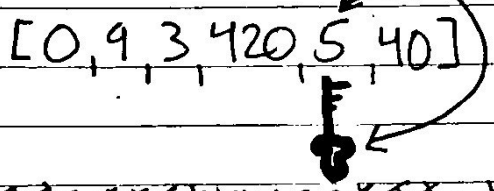
The Idea

- The most basic but core idea:

```
lst = [ ]
insert(12)
lst = [None ..... None, 12]
index -> 0,1,2,3,4,5,6,7,8,9,10,11,12
insert(1)
lst = [None, 1, ..., 12]
index -> 0, 1, ..., 12
```

Use Key to map item so we can access the item by the "index" O(1)

- Key, element = "my num", 5



Hash Functions

```
def hashf(Key):
    hash val = 0
    for ix, c in enumerate(Key, start=2):
        hv += ix * ord(c)
    return hash val
```

Notes:

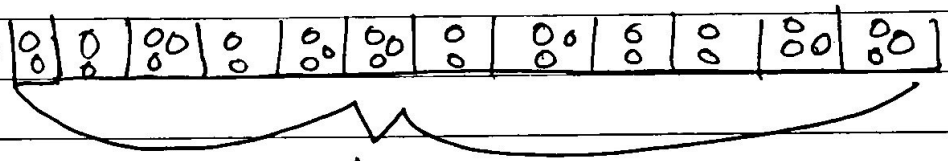
- Maybe use a nice prime near a power of 2
- Maybe start at 0 since using the whole key gives more variation

• We want hash functions to evenly distribute the keys

Bins & Balls

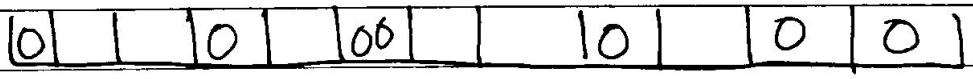
- If rand toss rand ball equaly likelyto land in any bin
- m bins n balls tossed, expect $\frac{n}{m}$ balls in each bin after awhile.
- Expect two balls in the same bin after $\sim \sqrt{\pi \frac{n}{2}}$ tosses
- Every bin has at least one ball after $\sim n \ln n$ tosses
- After n tosses the most loaded bin has $\Theta\left(\frac{\ln(n)}{\ln(\ln(n))}\right)$ balls.

After 30 tosses $\frac{n}{m}$ in each

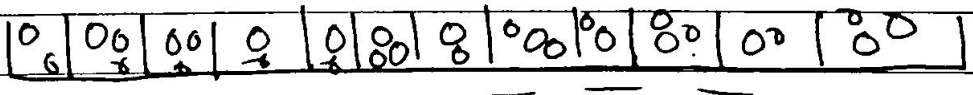


12 = m bins, n = 30

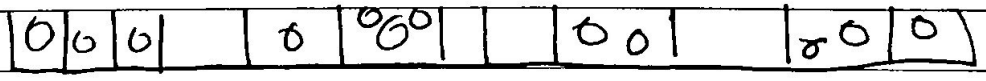
After 6.8 tosses $\sim \sqrt{\pi \frac{30}{2}}$



After 29.8 tosses $\sim 12 \ln 12$

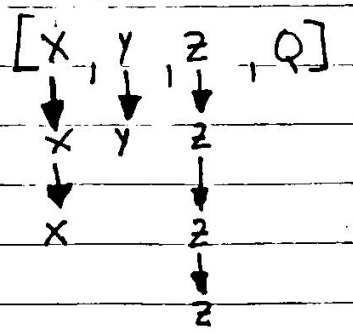


After 12 tosses $\sim \frac{\ln(12)}{\ln(\ln(12))} = 2.73$



Seperate Chaining

q, z, y, x = any random element don't matter



* Doesn't have to use linked list.
* keep size of list to n/5 in order to say access is O(1)

Linear Probe

[9, 3, 10, Nil, Nil, Nil, Nil]

insert(420)

say the hash indexes here

[9, 3, 10, 420, Nil, Nil, Nil]