THE PROBLEM.

Most of the data we care about come in logical pairs of a (key, value)

- (homeAddress, Person)
- (UserID, FacebookProfile)
- (TradingBookID, BookMetadata)
- (courseCode, (zID, marks))

- Cryptography (Identification)
- Systems (VirtualMemory)
- Bioinformatics (DNA search)

How can we allow for efficient inserts and lookups given a "key"?

Q: How could we represent this?

	LINKED LIST	TREES. (O(log n))	GRAPHS?	HEAPS.
lookup?	O(n)	O(log n)	O(V+E)	O(log n)
insert?	O(1)	O(log n)	O(1)	O(log n)

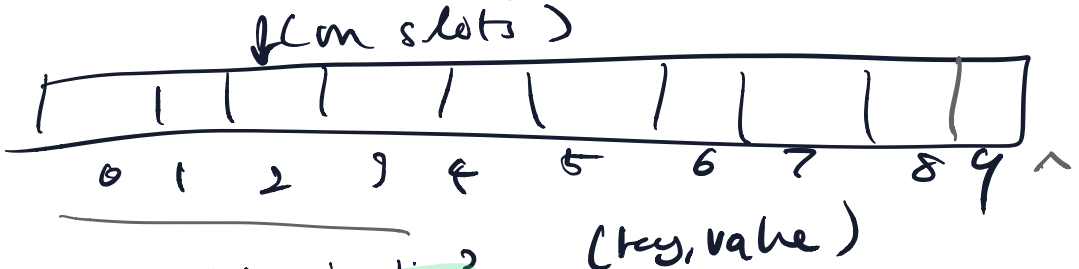
Goal:

LOOKUP: $\Theta(1)$ (amortized)

INSERT: $\Theta(1)$ (amortized)

Idea: Say we have an array of m slots.

(1)



(key, value)

(2) deterministic?

We have a function $h: K \to \mathbb{Z}_m$ that is computed in O(1) time. (ID)

(3) For (key, value) to insert we go by rule.

$A[h(key)] = value$.

④ For lookups, given a key where do I look?

O(1) insert

O(1) lookup.

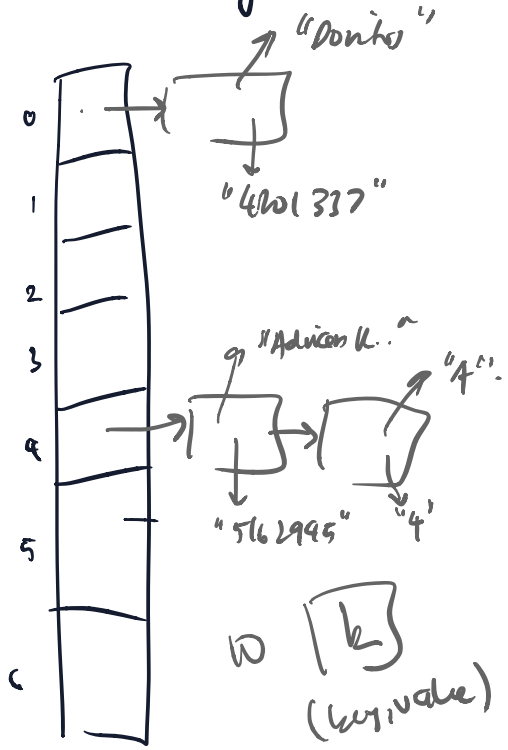
PROBLEM!

$|K| = |\mathbb{Z}_i|$

$|K| > |\mathbb{Z}_i|$

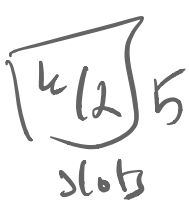
"injecting"

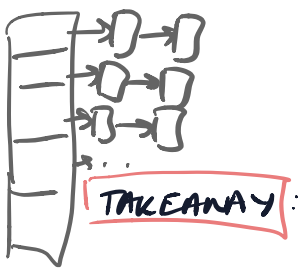
Handling collisions : (key $\in \mathbb{Z}$ ← UID, char* value). (k, V)



e.g. STUDENT RECORDS

ZID	NAME
$h(5162945) = 4$	"ADRIAN R. MARTINEZ"
$h(4201337) = 0$	"DORITOS X. MTNDEW"
$h(4) = 4$	"A"





TAKEAWAY: Set of keys does not have same cardinality as the range of our hash h.

Separate chaining

$h_1(x) = x \bmod 7$ where $m = 7$

⇒ there must exist a $k_1, k_2 \in \mathbb{Z}$ s.t.

$h(k_1) = h(k_2)$

aka. collisions!

Q1. What is the worst case when inserting k (key, value) pairs into our hash table? Imagine our table has size k/2 and we measure search cost by NUMBER of key comparisons.

If we assume hvs are evenly distributed what's average cost?

DOUBLE HASHING.

n = num keys stored
m = num slots in table

$\Omega\left(\frac{n}{\lg n}\right)$

load factor $\alpha = n/m$ = # keys per slot. lower bound of $\Omega(\alpha)$.

Idea: Have two hash functions h_1, h_2 and if we reach a collision add on $h_2(k)$.

Pseudocode:

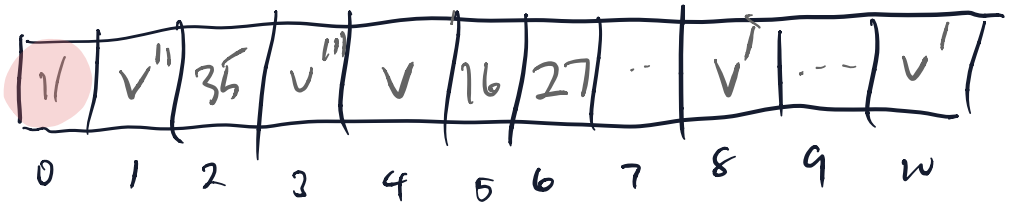
```
base = h1(k), shift = h2(k)
while (no collisions on A[base]):
    base += shift.
insert (k, v) in A at base.
```

why?

Q2. consider inserting the sequence

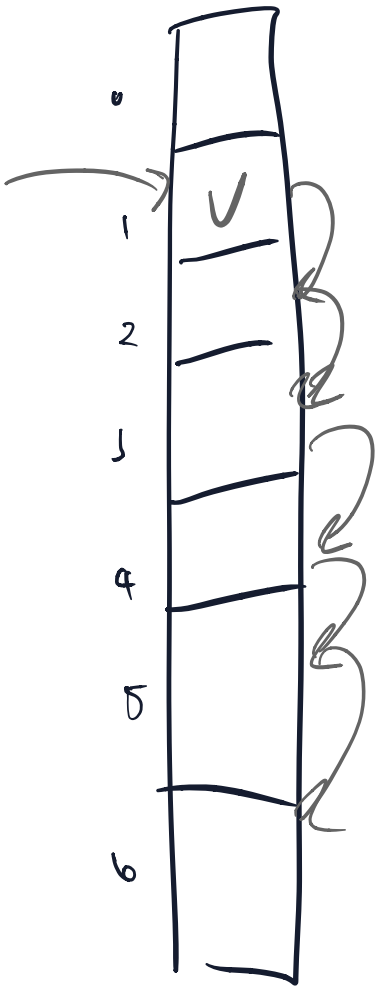
~~11~~ ~~16~~ ~~27~~ ~~35~~ 22 20 15 24 29 19 13

into an empty table with $h_1(x) = x \% 11$, $h_2(x) = (x \% 5) + 1$



LINEAR HASHING.

Idea: use $h(k)$ to define starting index, then +1 the index and to as our point to insert our value.



$h(x) = x \bmod 6$

① insert (8, "Hello")

② insert (4, "hey").

③ insert (14, "whatsup").

$h_2(x) = 1$

PROBLEMS

Why am I learning all three, when I only need one?

1. Easy / Pragmatic

2. Using 2 complicated hash functions ↓↓ number collisions

$h_1 \sim h_2$ ↑ distribution ↑ cache efficiency.