ADT Dictionary

A Dictionary ADT is one that supports the following operations:

Init(), IsEmpty()

Insert(x, K) — x is element with key K.

Search(K) — returns the element with key K, perhaps if missing, returns closest element with key < K

Delete(K)

Key from a totally ordered set

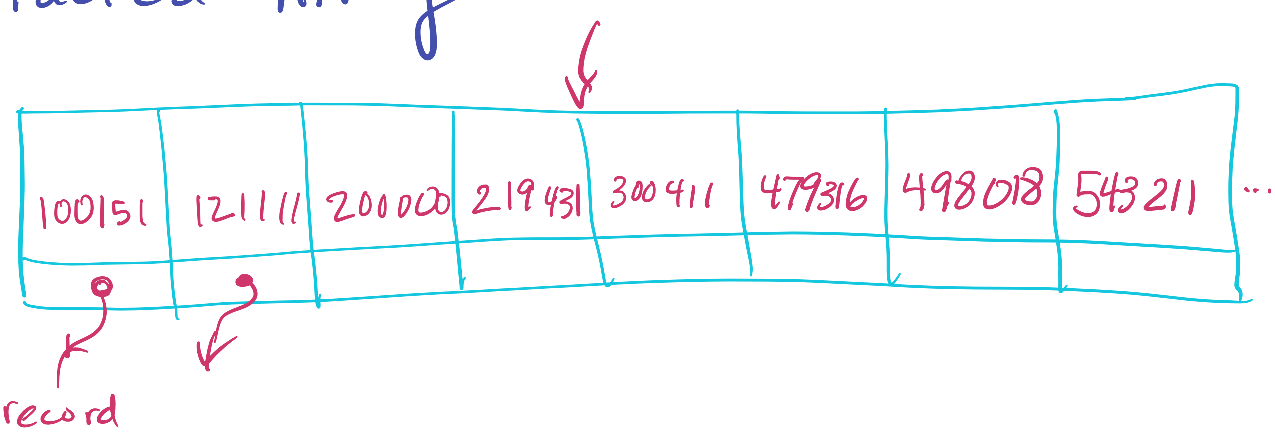
Differs from a PQ, wherein we always want to only extract the min.

	Insert	Search	Delete	Init	Space
unsorted list	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
sorted list	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
Array - packed	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
Array Loose	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$?	$\Theta(n)$?	$\Theta(\lvert U\rvert)$
* Binary Search Trees	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$

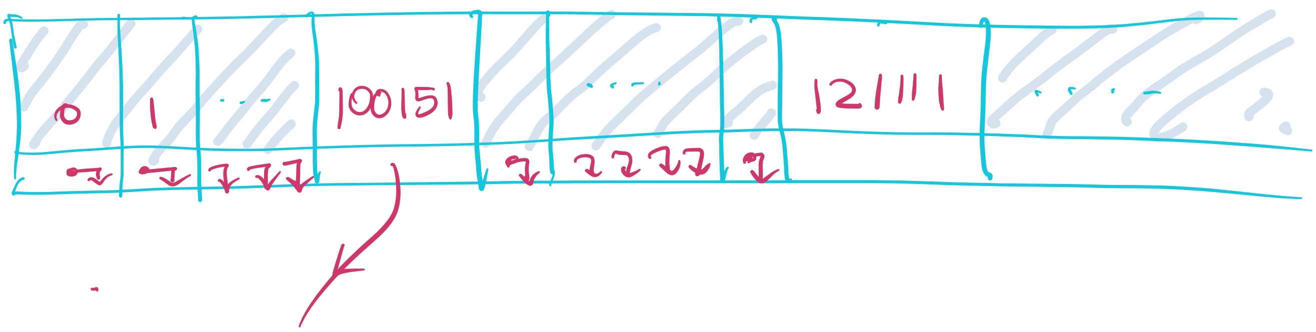
*

U = universe of key

Packed Array:



Loose Array = "Direct Address Table"

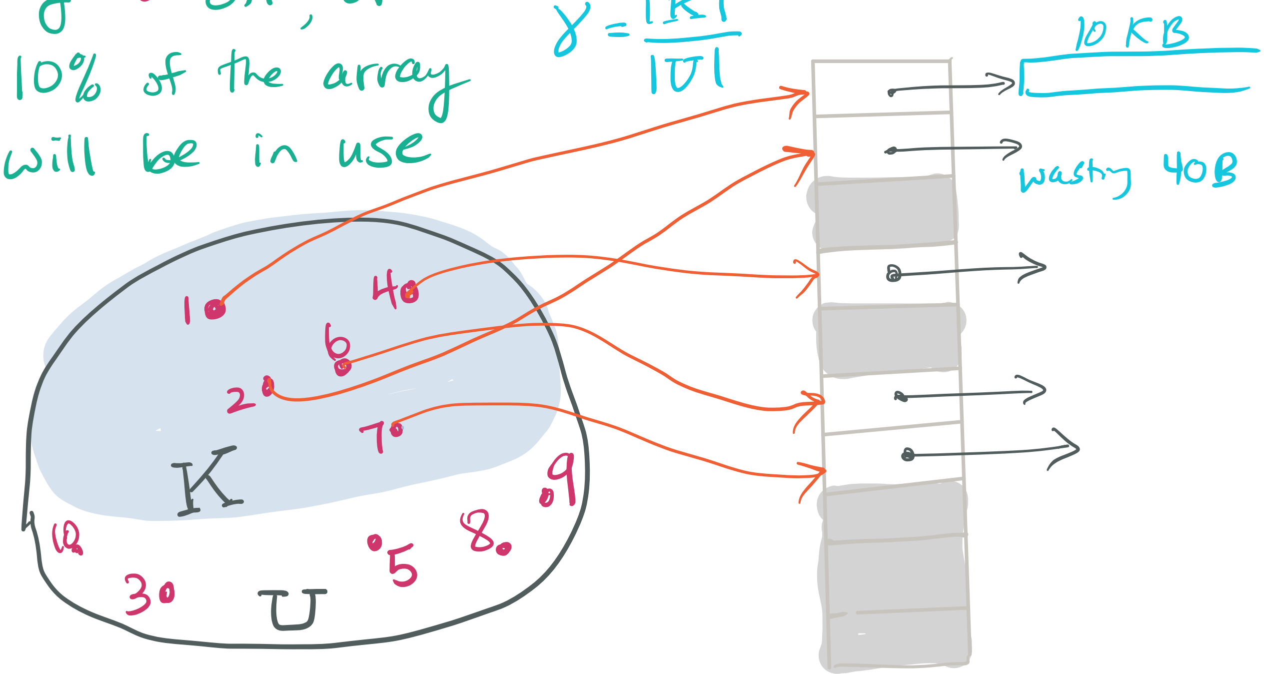


The Loose Array (or simply "Array", indexed by the Keys) is a fantastic Solution when γ is large when "Keys in use" is expected to be $\gamma \cdot U$

universe of key values

eg $\gamma = 0.1$, or 10% of the array will be in use

$\gamma = \frac{|K|}{|U|}$



Direct-Address implementation

DirectAddressSearch(K)

return $T[K]$

DirectAddressInsert(x, K)

$T[K]$ = a pointer to x

DirectAddressDelete(K)

$T[K]$ = NULL

The main problem with DirectAddress is that many applications do NOT have high γ (high key density)

Can we achieve DirectAddress-like

behaviour when γ is low ?

Hash Tables

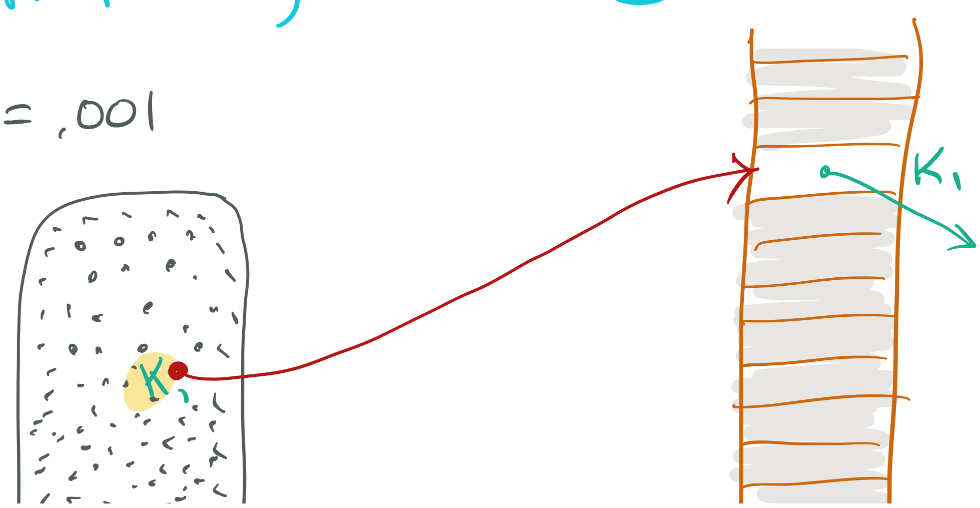
Suppose you have a function h

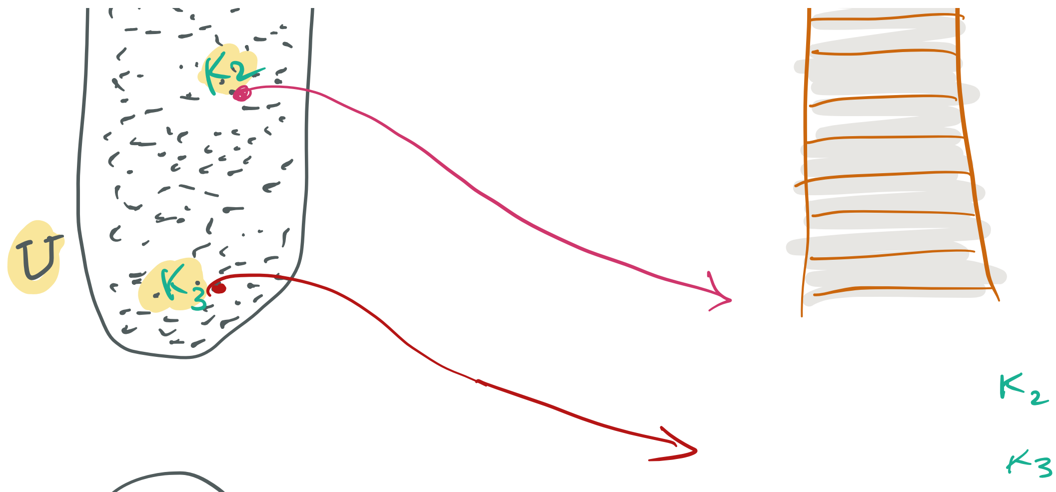
$$h : U \to \{0, \dots, m-1\}$$

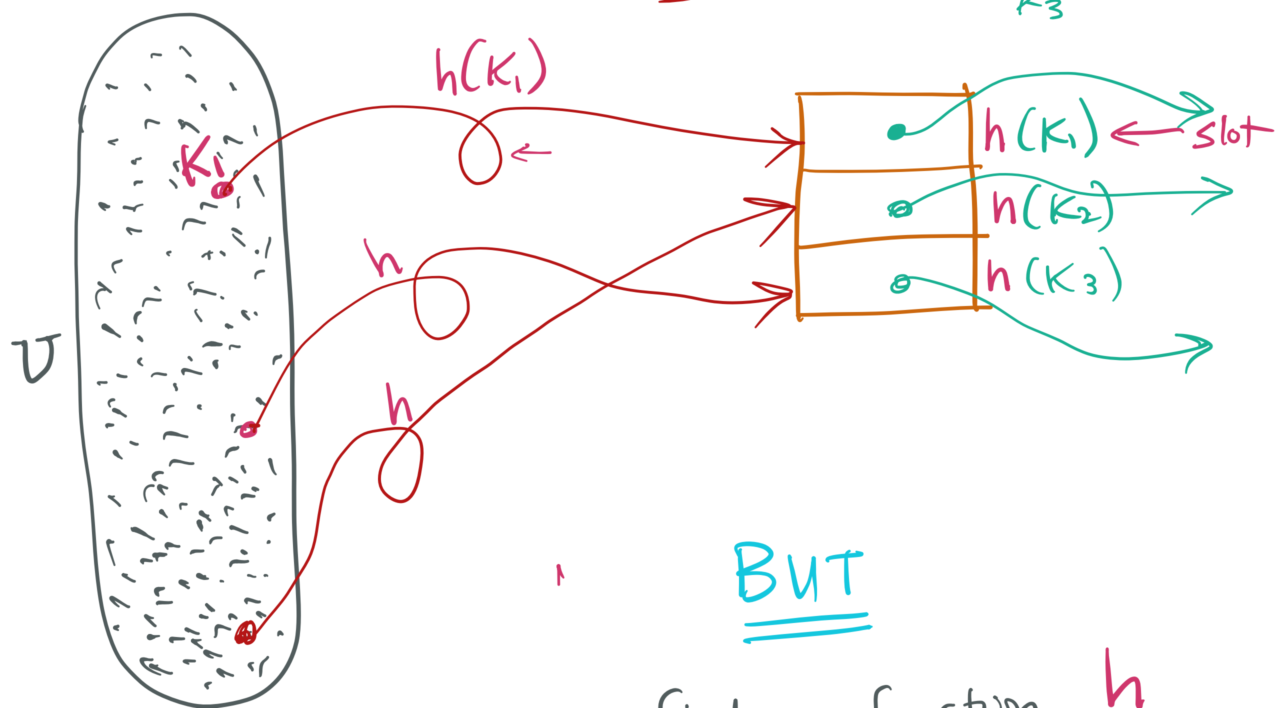
ie. f maps the universe of keys to a much smaller set of values...

ideally, values that index into an appropriately-sized array.

$\gamma = .001$







BUT

Can we find a function h

that maps:

- exactly to $\{0, \ldots, m-1\}$ with $m = |K|$?
  - don't need this
- so as to have no two distinct keys used, K_1 and K_2, where

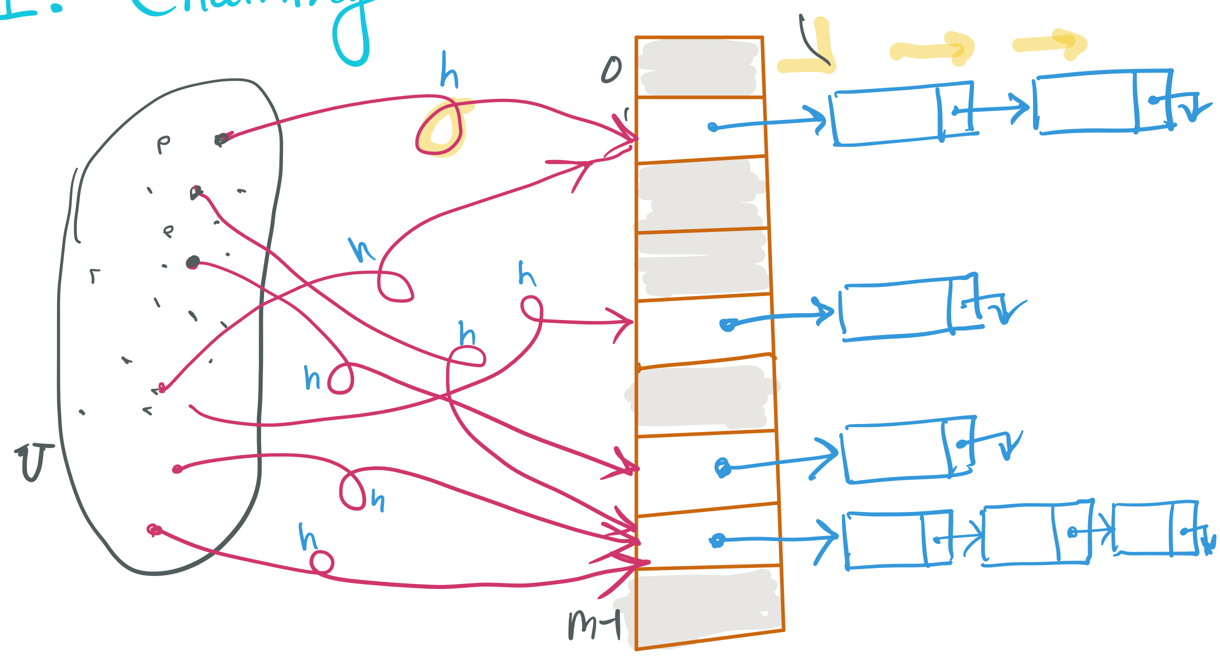
$h(K_1) = h(K_2)$?

↑ this is called a <u>collision</u>

Generally, we cannot avoid collisions, because we don't always know in advance what subset of $\mathcal{U}$ will be in use.

Two strategies for dealing with collisions.

I. Chaining



Analysis of hashing with chaining

In this context, we are actually interested in expected behaviour more than worst-case behaviour.

Why?

- Worst case behaviour is pretty bad
  - acts just like unsorted linked list
- The behaviour is not just a function of the inputs, but also of the hash function we chose ....

  worst case is decoupled from dependence on inputs alone.

  There will always be a hash function that has good worst case behaviour on the same inputs.

Given hash table T with m slots
and n elements

? $\alpha = \frac{n}{m}$ is called the load factor

= average number of elements stored per chain.

Suppose we pick a hash function so that a randomly selected element of U is equally likely to hash to each of the m slots = "uniform distribution"

and this assumption is simple uniform hashing

Also assume computing $h(K)$ is $\in O(1)$.

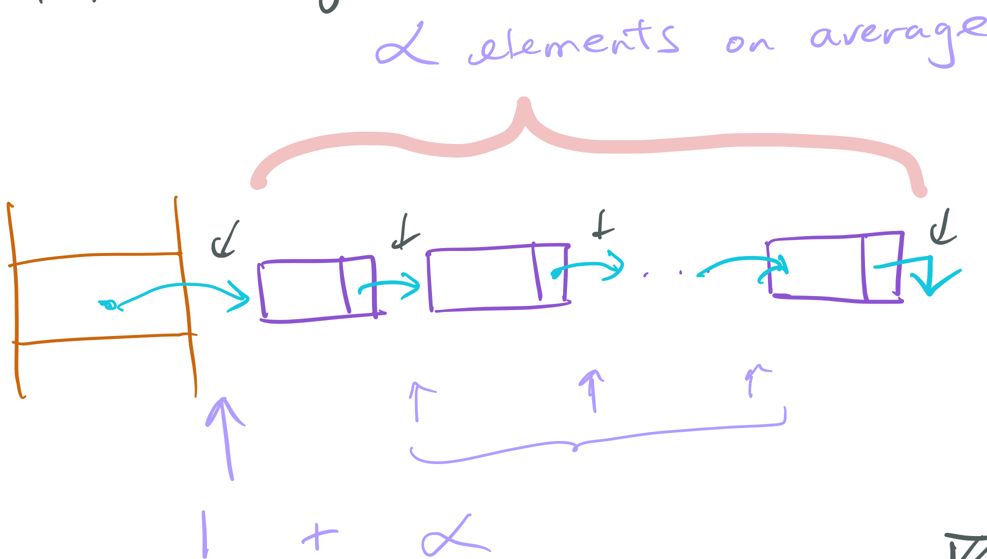
Theorem 12.1 (CLRS)

In a hash table with chaining, under simple uniform hashing, an unsuccessful search takes time

$\Theta(1+\alpha)$ on average.

+1 for hashing the key.

Proof



∎

Theorem 12.2 (CLRS)

In a hashtable with chaining, under assumption of simple uniform hashing, a successful search takes time $\in \Theta(1+\alpha)$

Proof.

Suppose we change Insert so that it traverses the list to the end and places the new item there.

This Insert has same running time as successful search.

Consider all the inserts, and all the element comparisons done during these (inefficient) inserts:

$$\sum_{i=1}^{n}\left(1+\frac{i-1}{m}\right) = n+\frac{1}{m}\sum_{i=1}^{n}(i-1)$$

#elements in table

When i was inserted

Divide by n to get average per insertion:

$$1 + \frac{1}{nm}\sum_{i=0}^{n-1} i = 1 + \frac{1}{nm}\left(\frac{(n-1)n}{2}\right)$$

$$= 1 + \frac{n^2 - n}{2nm}$$

$$= 1 + \alpha - \frac{1}{2m}$$

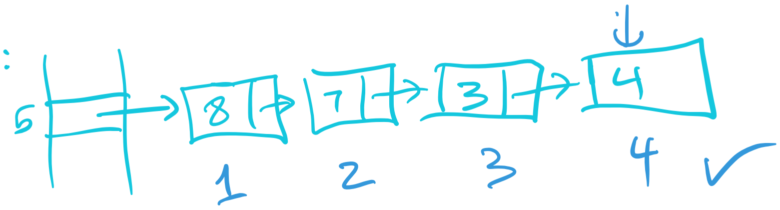
Number of comparisons in successful search = $1 +$ comparisons for insert

So is $\Theta\left(2 + \alpha - \frac{1}{2m}\right) = \Theta(1+\alpha)$ ∎

all hash to 5.

ins (4) ... ins (3) ... ins (7) ... ins (8)

good:



ins(8) .. ins(7) ins(3) ins(4),