

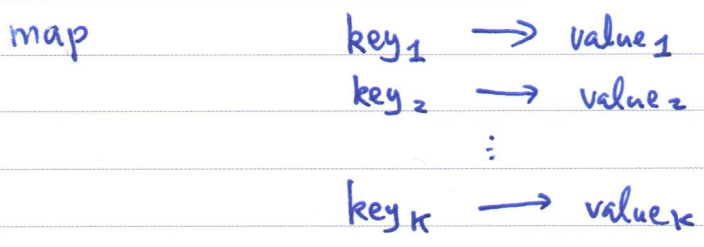
Subject:

* priority queue

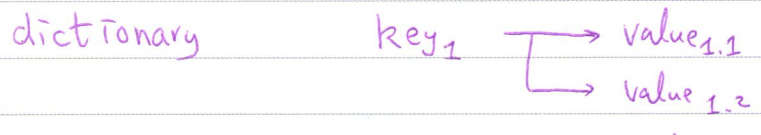
want A. insert(key, value)
A. retrieve Max()
fast

* what if

want A. insert(key, value)
A. retrieve(key)
fast?



unordered/
ordered



			search insert	retrieve
* sparse array	w/ key		O(1)	O(n)
	(on dense)	ordered key	O(n)	O(log n)
linked list	w/ key		O(1)	O(n)
		ordered key	O(n)	O(n)

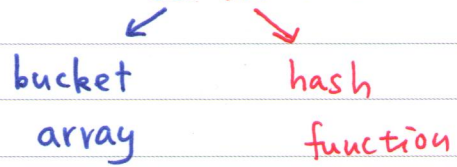
* if keys are integers

O(1) insert O(1) retrieve
by dense array (that wastes space)

* how about using a "simple" (almost O(1)) function

h: key → integer

A.insert(key, value) \Rightarrow A[h(key)] = value
 v = A.retrieve(key) \Rightarrow v = A[h(key)]



* if {keys} \longleftrightarrow {0, 1, 2, ..., k-1}

one-to-one

"perfect hashing"

* what if not?

e.g. #keys > k



\Rightarrow two keys of the same h(.)

\Rightarrow collision!

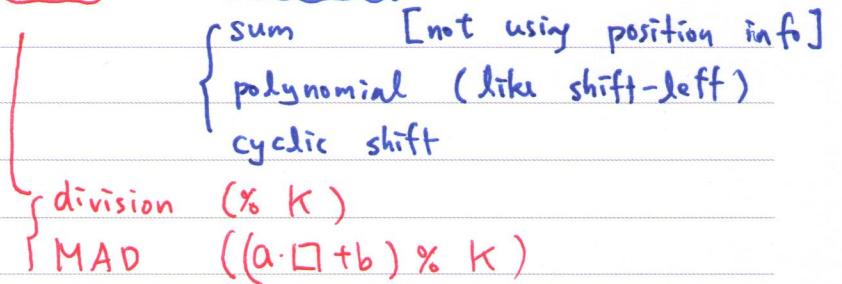
* e.g. h(str) = str[0] - 'a'; 26 possibilities
 h("act") = h("apple")

$h(str) = \sum_i (str[i] - 'a')$ 2^{32} ? possibilities
 not uniform

$h(str) = \sum_i (str[i] - 'a') \% k$ k possibilities
 "stop" "pots" "spot"
 \Rightarrow easy collision

* how to hash

$h(key) = \text{compress}(\text{Hashcode}(key))$



Subject :

* how to resolve collision during insertion?

- ① fixed (unordered) array per bucket
[can still overflow]
- ② linked list per bucket
chaining : don't want long chains
- ③ other data structure [usually called secondary] per bucket
[more complicated]
- ④ use other empty buckets
open addressing

* open addressing

- $a[key] = value$
 $x = a[key]$
- ① insert (key, value) to $h_0(key) = h(key)$ check
 - ② if fail, insert to $h_1(key)$ check
 - ③ if fail, insert to $h_2(key)$ check
 - ⋮
 - ④ if fail, insert to $h_m(key)$ check
 - ⑤ declare failure not found

* (A) linear probing :

$$h_i(key) = (h_{i-1}(key) + 1) \% K$$

$$= (h_0(key) + i) \% K$$

primary clustering

m = K-1

(B) quadratic probing :

$$h_i(key) = (h_0(key) + i^2) \% K$$

secondary clustering

m = ?

(C) double hashing :

$$h_i(key) = (h_0(key) + i \cdot \tilde{h}(key)) \% K$$

$\tilde{h}(key) > 0$

m = ?