* hash table of K entries after n keys

if $\frac{n}{K}$ large $\Rightarrow$ hash won't work

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

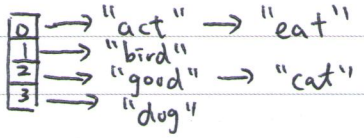
hash non-uniform $\Rightarrow$ $\frac{n}{K_{eff}}$ large

* idea: increase K when $\frac{n}{K}$ large
* naive
  1. set $K^{new} = 2K$
  2. change h(key) to range $\{0, \cdots, 2K-1\}$
  3. rebuild w/ $O(n)$ if insert is $O(1)$
  - cannot do often ($\frac{n}{K} > \theta$)
  - long waiting

* lazy approach
  1. set $K^{new} = 2K$ (use one more bit of $h(\cdot)$)
  2. change h(key)
  3. rebuild only the overflow entry $O(K) + O(\frac{n}{K})$

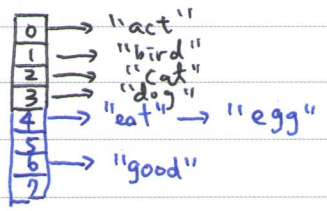
e.g. hashing w/ chaining of length 2

h(key) = (key[0] - 'a') % K

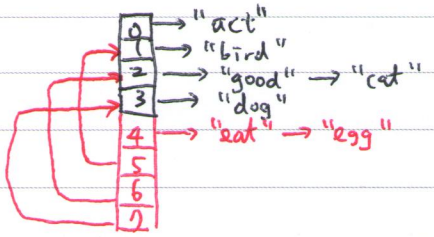


insert "egg"

naive



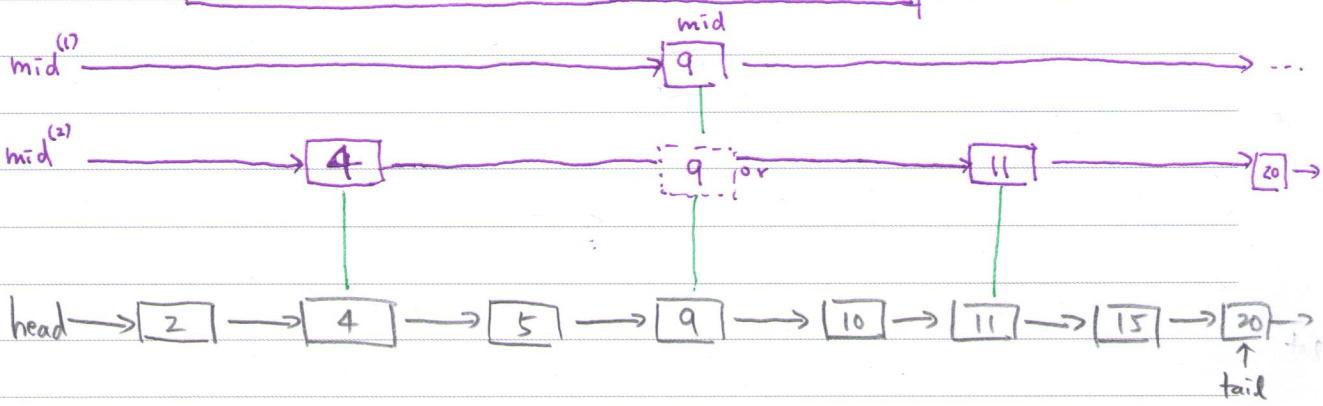
lazy (directory extension)



* hashing : extending array to do map / dictionary

* how to extend list to do map / dictionary
  - unordered : fast insertion , slow search
  - ordered : slow insertion , slow search

* why slow search ? sequential
  can we do binary search on ordered linked lists ?
  YES if mid node can be found quickly



* skip list = list + ... + list of quad + list of mid

search for 10
* (head , tail) = (2, 20) * mid = 9
* (head , tail) = (9, 20) * mid = 11
* (head , tail) = (9, 11) * mid = 10 found !

search for 14
(2, 20)
(9, 20)
(11, 20)
(11, 15) fail !

* but how to insert fast ? "cannot work if too strict"
  probabilistic : a node "survives" to the upper list w/ prob 1/2

* time of insert : w/ high prob. $O(\log n)$

search : $O(h) = O(\log n)$

expected space : $O(n) + O(\frac{n}{2}) + O(\frac{n}{4}) + \cdots = O(n)$