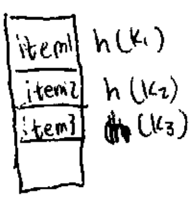
— Hash Table

- we have a universe of n keys and want to store them to a Hash Table with m slots
- we create a hash value $h(k_i)$ for each key k_i
- Running Time: the time to compute hash value + one time step to look up in the hash table.

— Collision

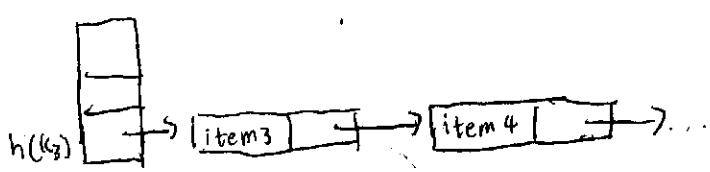
- when two keys map to the same hash value → leading to store to the same slot.
- for ex.



Now if $h(k_4) = h(k_1)$, we have to store item 4 in the same slot as item 1

- Collision resolution strategy:

Chaining: store collided item in linked list. ex.



if $h(k_3) = h(k_4)$

pro: Can store as many keys as possible.

cons: if we have a long collision look up time may not be constant.

* Collision resolution strategy (cont.d)

linear probing: resolve collision by sequentially searching the hash table for free location.

Ex. $h(x,i) = (H(x) + i) \pmod{m}$

where $H(x)$ is an ordinary hash function, and i is the i^{th} stepsize (all previous $i-1$ slots occupied)

Quadratic probing: similar to linear probing but search for open slots as a quadratic function

Ex. $h(x,i) = (H(x) + i^2) \pmod{m}$

pro of linear/quadratic probing: fast insert, lookup.

cons: we are limited in the number of elements can be stored, and we can not empty the slot when deleting. (we have to put a dummy element in deleted slots)

* Simple uniform hashing assumption.
  * each key $k \in K$ is equally likely to be hashed to any slot of table T, independent of where other keys are hashed.
  * load factor = $\alpha = \frac{n}{m}$ = average number of keys per slot.

Example: key a_1 and a_2 would have $\frac{1}{m}$ chance of collision

under simple uniform hashing assumption.

so under SUHA, both a_1 and a_2 don't get hashed

to slot 1 is $\left(\frac{m-1}{m}\right)^2$.

- Rolling Hash.

· useful in string matching. consider 2 strings A and B,

we want to find all matching subsequence of length k.

$$h(A[i:i+k-1]) = A[i]\cdot 26^{k-1} + A[i+1]\cdot 26^{k-2} + \ldots + A[i+k-1]$$

so $h(A[i+1:i+k]) = A[i+k] + 26\cdot h(A[i:i+k-1]) - A[i]\cdot 26$

so for string length k, total running time to calculate all

hash value is $O(k) + O(n-k) = O(n)$

· when attempting to find the longest common substring,

use binary search. Hash all substring length k of A into a

hashtable and look up all substring length k of B.

· Another example would be looking up words in

a dictionary of DNA bank and trying to find

patterns.