#ASLI ENGINEERING

Open Addressing in Hash Tables

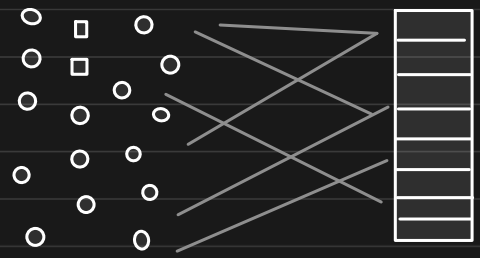
SWIPE

BY

ARPIT BHAYANI

Resolve conflicts using Open Addressing

Conflicts are inevitable!



$key_1 \longrightarrow f \longrightarrow h_1$

$key_2 \longrightarrow f \longrightarrow h_1$

Multiple keys can produce same hash key upon hashing

So, how can we store multiple keys in the same slot?

The two classical ways of achieving this

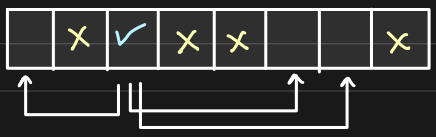
1. Chaining
* 2. Open addressing

Core Idea:

Instead of using some auxiliary data structure, use the empty slots.

Open addressing

When keys collide, find a way to hunt for an available slot in the array ... deterministically



Probing

Finding the next available slot is called probing.

probing strategy can be defined as

$j = p(k, i)$ that spits out the new index where key k can be placed in attempt i

$j \in [0, m)$ $i \in [0, m)$ $m \rightarrow$ size of the hash table

Hence, first insert $j = p(k,0)$, if that is occupied we try
$j = p(k,1)$, if that is occupied we try
$j = p(k,2)$, if that is occupied we try
⋮
$j = p(k, m-1)$, if that is occupied we try

Good probing function

The probing function should generate the permutation of numbers $[0, m-1]$ so as to cover the entire space eventually.

Implementing probing function

It is a simple mathematical or algorithmic function that deterministically tells us our next slot for a particular key

key attempt

$p(k_1, 0) \rightarrow 5$ — 1st attempt probing function gave 5

$p(k_1, 1) \rightarrow 7$ — 2nd attempt probing function gave 7

$p(k_1, 2) \rightarrow 2$ — 3rd attempt probing function gave 2

So, while looking for key k_1,

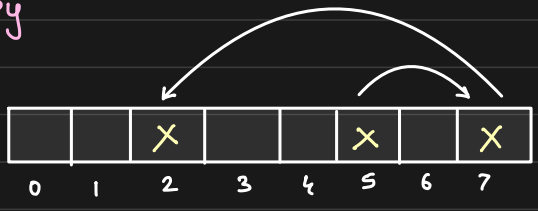
we first look at $p(k_1, 0) = 5$,

if we cannot find, $p(k_1, 1) = 7$,

if we cannot find, $p(k_2, 2) = 2$

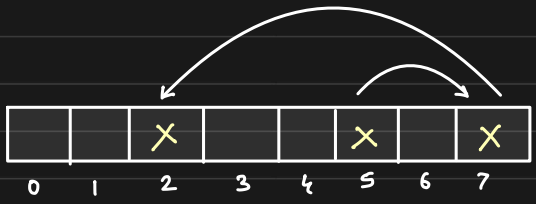
⋮

Hash Table Operations : Adding a key



Until we find a free slot,
keep probing and checking
at the first free slot, put the key

Hash Table Operations: lookup

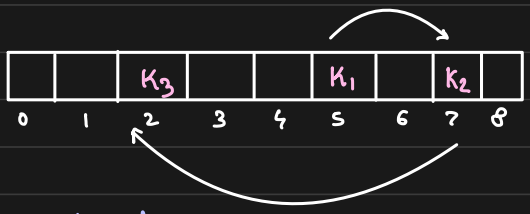


lookup is similar to adding.
Using probing function we
try to find key in slots
Iteration stops when we find the key
or we stumble upon an empty slot
or we exhaust iterating over all the slots

Hash Table Operations : Deleting a key

the deletion is a soft delete. We lookup the key using probing function and upon discovering, we mark the slot is deleted.

But why soft delete?



K_1, K_2, K_3 hashed at index 5

With open addressing, say we got them placed at 5, 7 and 2

[probing function]

Hard

Say, we delete key k_2 and now we are looking for K_3

if you delete k_2, then how would you ever reach k_3?

empty slot == stop iteration

So, you will never be able to reach k_3 during lookup

Hence, we need to differentiate b/w Free and Deleted

↓

Soft deletion is the way to go

limitation of Open addressing

Max number of keys = # slots in array