#ASLI ENGINEERING

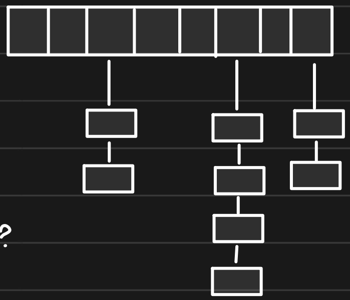
Understanding performance of a Hash Table

SWIPE

BY

ARPIT BHAYANI

Performance of a Hash Table



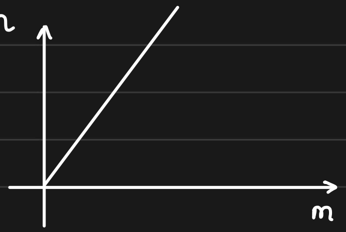
Hash Table gives constant time performance when there are zero collisions

But thats impossible...

So, how can we quantify how "full" the table is ?

This ↑ is called the load Factor

load Factor



load factor $\alpha = \dfrac{n}{m}$

n → elements

m → slots in the hash table

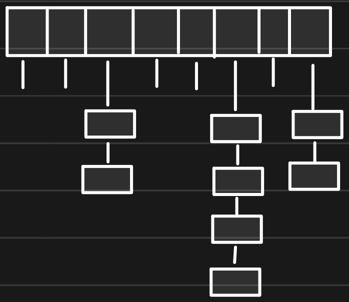
There is a good corelation b/w α and time

load Factor with different resolution strategies

1. with chaining, we can never fill the table
   ↳ we can always add a new node to list
2. with open addressing, we would eventually run out of space and writes would fail

In both cases perf would go down much before operations start to degrade

Chained Hashing



load factor = average number of elements stored per linked list

$$\alpha = (0+0+2+0+0+4+0+2)/8 = 1$$

Time to resolve collision = $O(1+\alpha)$

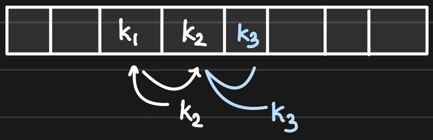
adding node — traversing the list

* Upper bound. we can definitely optimize it

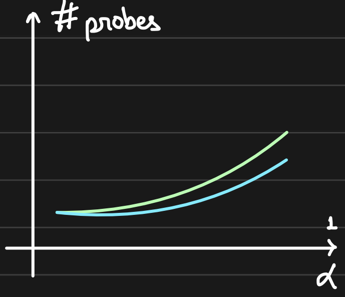
Open Addressing

Doing analysis with open addressing is challenging

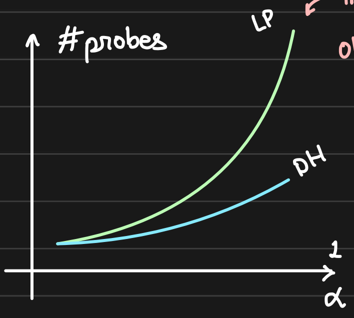
as... Collisions Interfere !!



Collisions on one slot will interfere with probe beginning on other



Hit



Misses

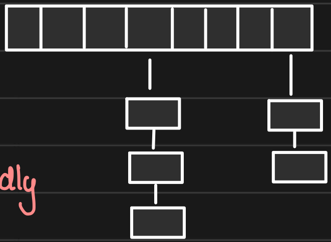
infinite loop in key lookups otherwise O(m)

ARPIT BHAYANI

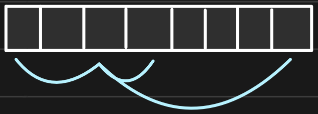
Which is the best strategy then?

Depending on the probing strategy, the cost of each probe changes, and it matters.

1. Probing is costly with chained hashing
   - ↳ linear traversal of linked list
   - ↳ random memory accesses ← Not cache friendly



2. Double hashing requires us to evaluate 2 hash functions
   - ↳ CPU intensive and time consuming
   - ↳ random jump in array ← Not cache friendly



* Optimal strategy depends on the use-case.

Hence, tune and evaluate!!

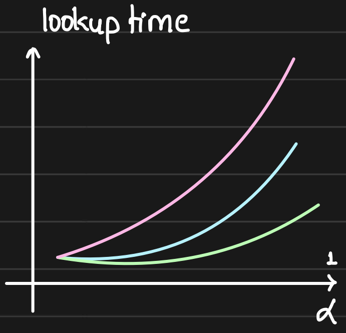
How to compare and benchmark?

If we are re-implementing Hash Tables, it is important to know how good we did, and hence analyze and benchmark.

ARPIT BHAYANI

lookup time as a function of load

For your strategy identify how lookup time changes as a function of load factor against all strategies



1. create a hash table of size 1024
2. insert n elements varying 32 to 900
3. lookup 1000 random keys (high miss ratio)

We should see:

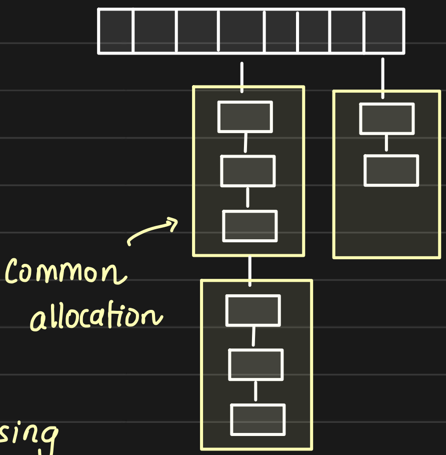
1. performance for open addressing degrades as $\alpha \to 1$
2. chained approach degrades gracefully
3. linear probing would be slower than double hashing
4. Probes of Double Hashing will be shorter

Note: we cannot conclude, chained hashing >> open addressing because chained hashing is not very cache friendly

* when tables are short we do not see impact of caching

ARPIT BHAYANI

Bettering cache performance in chained hashing



To leverage cache in chained hashing, we can allocate pool for each slot so that the nodes of list are close to each other in memory

* linked list of arrays

Chained Hashing outperforms Open Addressing when tables are smaller.

Open addressing outperforms chained hashing when tables are large enough

→ critical for your application

If Hash Table performance matters a lot,

experiment with different strategies, parameters, and algorithms