#ASLI ENGINEERING

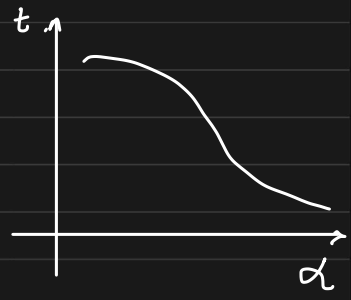
Implementing Resize of a Hash Table

SWIPE

BY

ARPIT BHAYANI

Resizing Chained Hash Tables or tables with Open Addressing



Resizing is important to maintain a consistent performance.

Performance degrades as load Factor increases

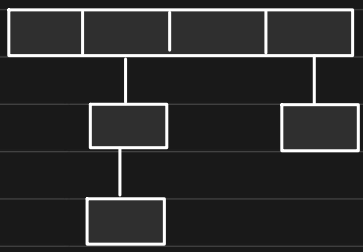
↳ longer lookups

↳ multiple collisions

Resizing Chained Hash Tables

To decide when to resize we need to know the load factor α and hence keep a track of

keys in table and size of the table



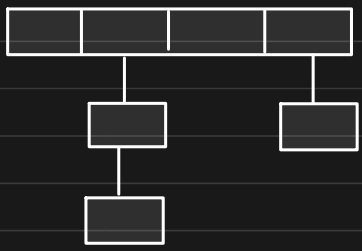
Resize is triggered upon insert

```
insert_key(k, table) {
    ____________________
    ____________________

    α = count_keys / length_table
    if (α >= 0.5) {
        resize(table, length_table x 2);
    }
}
```

ARPIT BHAYANI

Table is shrunk upon delete key



```
delete_key ( table, key ) {
    ---------------------
    ---------------------
    α = count_keys / length_table
    if (α < 0.125) {
        resize ( table , length_table / 2);
    }
```

Resize is all about

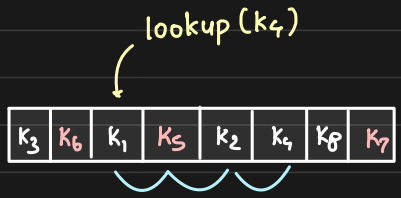
- allocating a new array of desired size
- insert all keys in new array
- delete old array once movement is complete

Two ways to prepare new array

1. Re-insert all existing keys in the new one — simple but expensive
2. Instead of re-allocating linked list nodes, just adjust the pointers — slightly efficient

ARPIT BHAYANI

Resize open addressing hash tables



In open addressing, we always soft delete the elements, so that we can reach the keys further in the collision chain

Hence, when a key is deleted from hash table, we CANNOT reduce the key counter

because the deleted keys also slow down key lookups

To handle this, we need

1. keys counter → to keep track of active keys in table

   insert → keys counter ++

   delete → keys counter --

2. used counter → to keep track of slots actually occupied

   insert → used counter ++

   delete → nothing

load factor = $\alpha = \frac{\#\text{ used}^{*}}{\#\text{ slots}}$ ← * #used instead of #keys

ARPIT BHAYANI

Resize is triggered upon insert

When the table is resized, we only rehash the active keys and skip the deleted keys.

```
insert_key(k, table) {
    ---------------------------------
    α = count_used / length_table
    if (α >= 0.5) {
        resize(table, length_table x 2);
    }
}
```

load factor $\alpha = \frac{\text{used}}{\text{length}}$

Table is shrunk upon delete key

When the table is shrunk, we only rehash the active keys and skip the deleted keys.

```
delete_key(table, key) {
    ---------------------------------
    α = count_used / length_table
    if (α < 0.125) {
        resize(table, length_table / 2);
    }
}
```

load factor $\alpha = \frac{\text{used}}{\text{length}}$

ARPIT BHAYANI