Hash Tables

Typically, the insertion and find algorithms will attempt to locate a valid location and terminate after m tries.

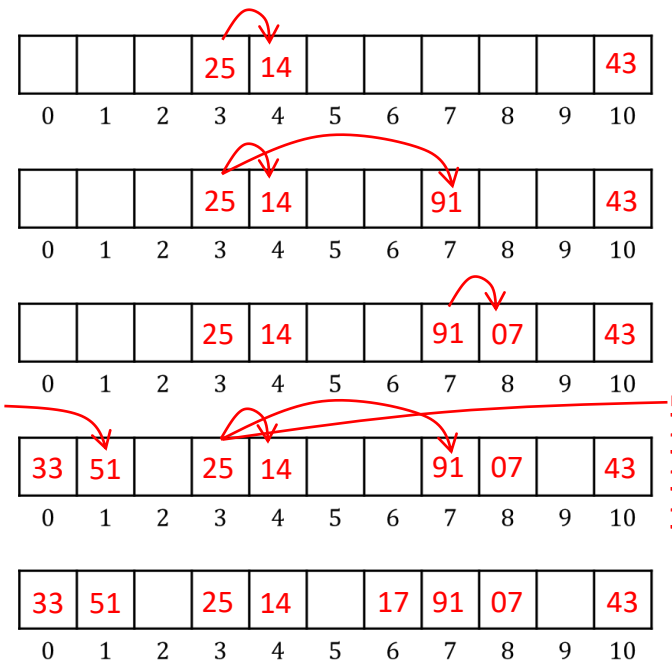
With linear probing this denotes the fact that every single one of the m locations in the table was examined:

- insertion: this signifies there is no open location to insert into
- find: this signifies no entry matches the target key

Quadratic Probing

Insert the following keys into the initially empty hash table below, with $m = 11$. Assume the hash function is $h(k) = k \bmod m$

43, 25, 14, 91, 07, 33, 58, 17, 03



$58 \% 11 = 3$
$3 + 1 = 4$
$3 + 4 = 7$
$3 + 9 = 12, 12 \% 11 = 1$

Insertion fails on 03

$03 \% 11 = 3$
$3 + 1 = 4$
$3 + 4 = 7$
$3 + 9 = 12, 12 \% 11 = 1$
$3 + 16 = 19\ 19 \% 11 = 8$
$3 + 25 = 28, 28\%11 = 7$
$3 + 36 = 39, 39\%11 = 6$
$3 + 49 = 52, 52\%11 = 8$
$3 + 64 = 67, 67\%11 = 1$
$3 + 81 = 84, 84\%11 = 7$
$3 + 100 = 103, 103\%11 = 4$
$3 + 121 = 124, 124\%11 = 3$

Quadratic Probing Pros and Cons

Pros: Fixed primary clustering (associated with linear probing)

Cons: Only guaranteed to find an open location when $\alpha \leq 0.5$
Secondary clustering:

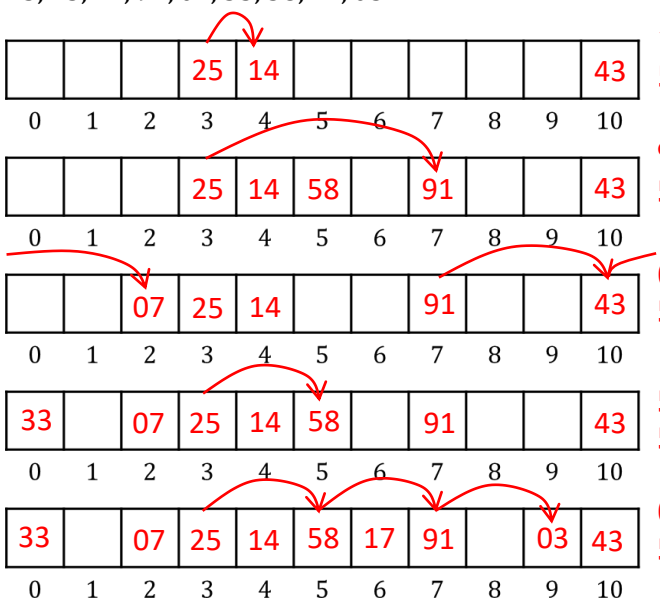
- items that hash to same index all go up in the same probe sequence
- keep jumping over empty spots and continue colliding on full ones

---

Double Hashing

Insert the following keys into the hash table below, The hash function is $h(k) = k \bmod m$. The second hash function is $h_2(k) = 5 - (k \bmod 5)$

43, 25, 14, 91, 07, 33, 58, 17, 03



$14 \% 11 = 3$
$5 - (14\%5) = 1$

$91 \% 11 = 3$
$5 - (91\%5) = 4$

$07 \% 11 = 7$
$5 - (7\%5) = 3$

$58 \% 11 = 3$
$5 - (58\%5) = 2$

$03 \% 11 = 3$
$5 - (3\%5) = 2$

Pros:

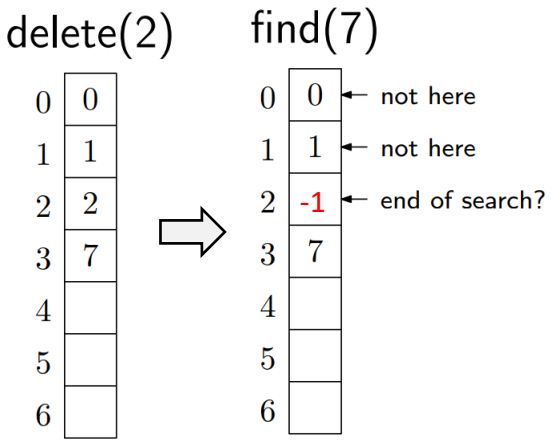
- If $\alpha < 1$ it will always find an open location
- No primary or secondary clustering
- keys that map to same initial index will have a different # of jumps due to second hash function

Cons:

- One extra calculation for collision handling (the second hash function), but it should be quick to compute.

Deletion in Open Addressing

For this example, examine the table below, with the hash function $h(k) = k \bmod 7$, and that uses linear probing:



tombstone:
- keep track of the fact that a 2 has been deleted
- if the keys are also positive integers, then could put -1

When searching, treat a tombstone as an occupied location.

Key observation: A key may be found *after* the deleted element (which is now a tombstone) in the probe sequence.

With tombstones, when probing occurs and a tombstone is located, we will keep probing until we either find the key later on in the probe sequence, or we find the first null location, which would mean the key is not in the hash table.

Questions:

For insertion, can we overwrite the tombstone?

No, what if the key is already contained in the hash table (later in the probe sequence after the tombstone), and so the associated value should be updated instead of inserted. If we overwrite the tombstone we could end up with the same key twice!

Instead, first try and find the key, if it is not found, *then* we can overwrite the tombstone with the new entry!

Will the hash table fill up with tombstones?

Potentially. And this is a bad thing since a lot of tombstones slows down the hash table operations (similar to how an increasingly large load factor does).

What we could do is wait for a time when the usage levels are very low for the hash table, and copy and re-insert all of the elements into a new storage array without any tombstones.