# CI583: Data Structures and Operating Systems
## Handling collisions in hashtables

In the last video we looked at the basic ideas behind hash tables.

We saw that hash functions will produce "collisions" – two different keys that result in the same hash value.

The simplest way to deal with that is a "linear probe" – if we look at the array index $n$ is full, where $n$ is the hash value of the key we want to store, look in index $n + 1$, and keep looking til we find an empty spot.

As the hash table grows, so does the average probe length.

# Open addressing

Not only could several keys hash to the same index but even when we insert a value with a hash that hasn't been seen before, it might be occupied with the overflow from an earlier index.

This problem is called primary clustering, and it causes performance to degrade from $O(1)$ all the way down to $O(n)$ at worst.

We can reduce the problem by making sure the hash table does not become more than two thirds full at most.

# Rehashing

The load factor of a hash table is the ratio of values stored to array size. When using open addressing, this should not exceed 0.6.

When the load factor becomes a problem we can expand the array.

We might do that by doubling its size, but we need to call `insert` repeatedly to copy the data over, so this is quite expensive. -**Why?**

# Rehashing

The load factor of a hash table is the ratio of values stored to array size. When using open addressing, this should not exceed 0.6.

When the load factor becomes a problem we can expand the array.

We might do that by doubling its size, but we need to call `insert` repeatedly to copy the data over, so this is quite expensive. -**Why?**

The reason we can't just loop through the old array and copy values in the same positions in the new one is that the hash function depends on the size of the array.

# Open addressing

Clustering is a significant problem with linear probing. Once a cluster forms it tends to get bigger, and the bigger it gets the faster it grows, reducing performance all the time.

One of the ways we can combat this is to make sure that our hash function produces a uniform distribution, i.e. that no part of the array remains sparsely populated while clusters form elsewhere.

Another approach is to change the way the probe works.

# Open addressing

In quadratic probing we search more widely distributed locations than the area adjacent to the occupied location.

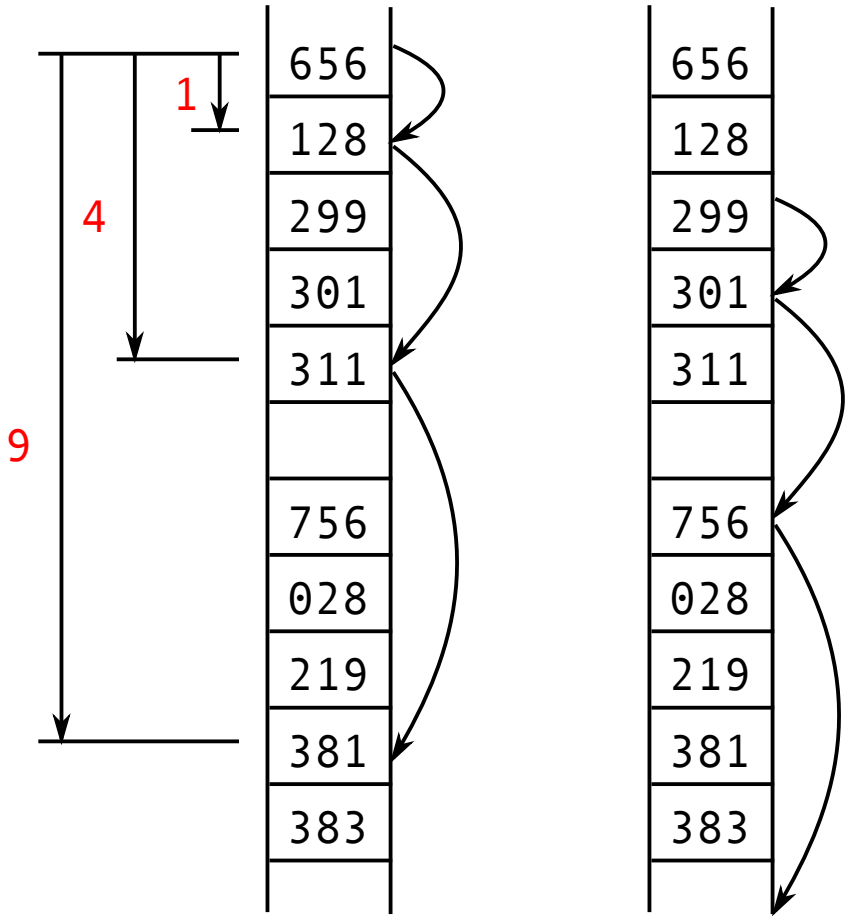
For each step in the probe, we jump forward by the square of the step number.

So, if we want to insert a value at $x$ but it is occupied, the probe looks at locations $x + 1$, $x + 4$, $x + 9$, $x + 16$, and so on.

These locations are modulus the array size, so that the probe wraps around and starts again near the beginning.

# Quadratic probing

Successful and unsuccessful quadratic probes (or the one on the right is a success, if looking for somewhere to insert):



# Double hashing

Quadratic probing does away with the problem of primary clustering, but clusters can still be produced since all keys with the same hash will probe the same locations – this is called secondary clustering.

A better solution, and the one most often used in practice, is called double hashing.

This allows us to generate probe sequences that depend on the key, rather than being the same for every key.

# Double hashing

The idea is to hash the key a second time, using a different hash function, and base the step size on the result.

The second hash function must not be the same as the original one (**why?**) and must output a step greater than zero. An example:

$$step = k - (hugeRange(key)\%k),$$

where $k$ is a constant and *hugeRange* is the first part of the original hash function.

# Double hashing

When we use double hashing, the array size must be a prime number.

Suppose the array size is non-prime, e.g. 15, and a key hashes to an initial position of 0 with step size 5. The probe will look at 0, 5, 10, 15, 0, and so on.

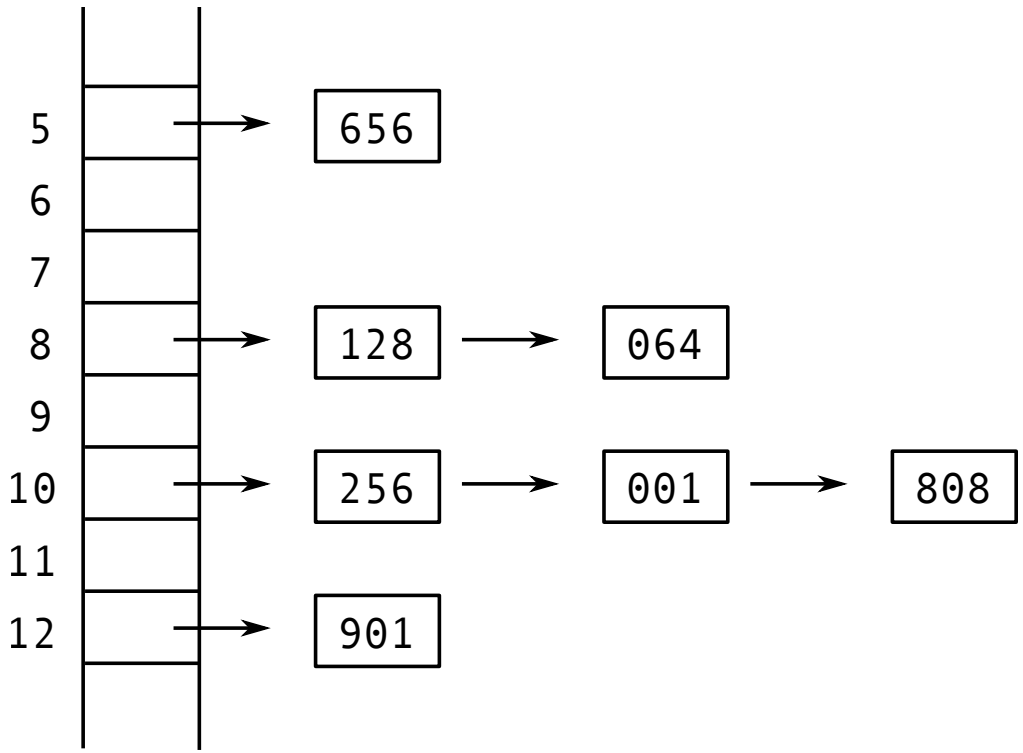
It could be that these locations are all occupied but there is space in positions 2, 3 and 4...

If the array size is a prime number then no step size will divide it evenly and every cell will eventually be inspected.

In the previous example, if we change the array size to 13, the probe looks at 0, 5, 10, 2, 7, 12, and so on.

# Separate chaining

As an alternative to open addressing, where we probe for empty locations, we can store collections of values at each index. This is called separate chaining.



# Separate chaining

Separate chaining is conceptually simpler than open addressing with double hashing but less efficient.

We can allow the load factor to exceed 1, as we simply add items to the list at each location, but we don't want the lists to become too long – searching them will have to be done in linear time.

The fact that we will rarely, if ever, need to rehash makes separate chaining a good choice if we don't know in advance how much data we want to store.