

Double hashing

Use primary and secondary hash functions $\text{hash1}(\text{key})$ and $\text{hash2}(\text{key})$, respectively.

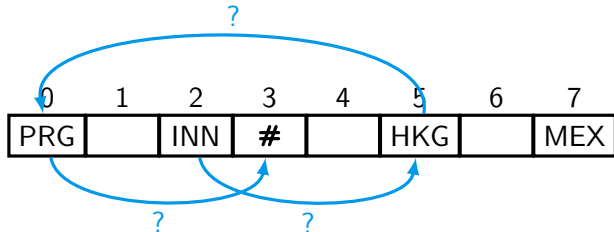
Insertion: We try the primary position $\text{hash1}(\text{key})$ first and, if it fails, we try fallback positions:

1. $(\text{hash1}(\text{key}) + 1 * \text{hash2}(\text{key})) \bmod T$
2. $(\text{hash1}(\text{key}) + 2 * \text{hash2}(\text{key})) \bmod T$
3. $(\text{hash1}(\text{key}) + 3 * \text{hash2}(\text{key})) \bmod T$
4. ... (until we find an available space)

T is the table size

Example

If $\text{key} = \text{TPE}$,
 $\text{hash1}(\text{key}) = 2$,
 $\text{hash2}(\text{key}) = 3$:



Double hashing is an improvement of linear probing. The only difference is that every `key` has a different sequence of “fallback” positions given by the secondary hash function.

Except for how we calculate the fallback positions, all the operations (`insert` , `delete` and `lookup`) work the same way; we use tombstones to mark deleted keys, when looking up we skip over those tombstones etc.

Linear probing's fallback positions are:

$$(\text{hash}(\text{key}) + i) \bmod T \text{ for } i = 1, 2, 3, \dots$$

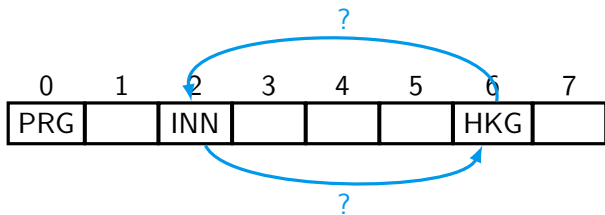
whereas double hashing's fallback positions are:

$$(\text{hash1}(\text{key}) + i * \text{hash2}(\text{key})) \bmod T \text{ for } i = 1, 2, 3, \dots$$

Avoiding short cycles

We can have short cycles!

Consider inserting a `key` such that `hash1(key) = 2` and `hash2(key) = 4` into a table of size 8:



The table size T and `hash2(key)` have to be coprime!

Two solutions:

- (a) T is a prime number.
- (b) $T = 2^k$ and `hash2(key)` is always an odd number.
(preferred)

Maths break:

- Two numbers a and b are said to be *coprime* if no number, other than 1, divides both a and b
- *Prime numbers* are the numbers which are divisible only by 1 and themselves.