Double hashing

Use primary and secondary hash functions `hash1(key)` and `hash2(key)`, respectively.

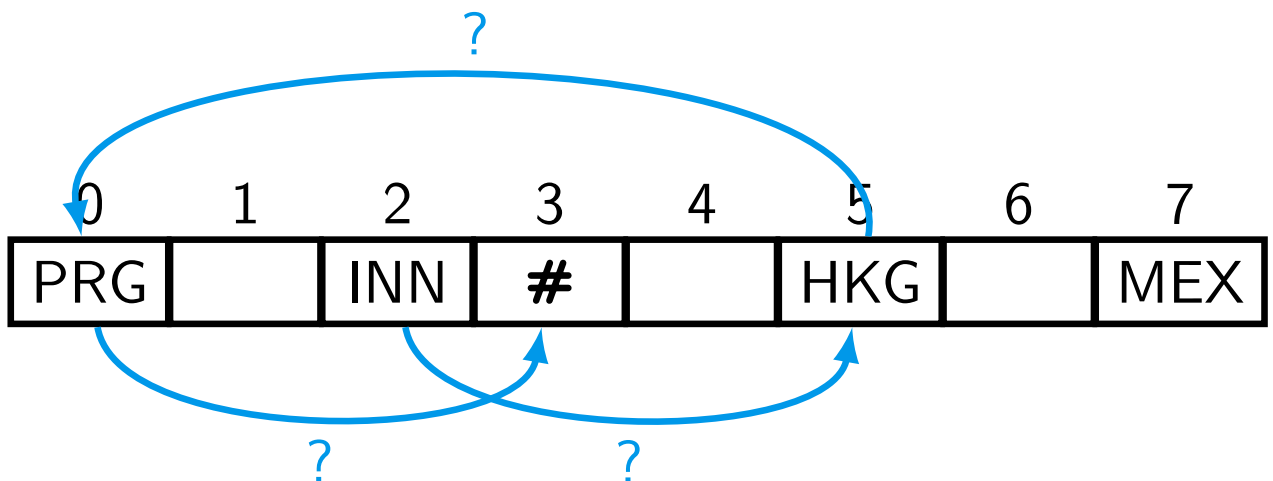
Insertion: We try the primary position `hash1(key)` first and, if it fails, we try fallback positions:

1. `(hash1(key) + 1*hash2(key)) mod T`
2. `(hash1(key) + 2*hash2(key)) mod T`
3. `(hash1(key) + 3*hash2(key)) mod T`
4. ... (until we find an available space)

`T` is the table size

Example

If `key = TPE`, `hash1(key) = 2`, `hash2(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:

`(hash(key) + i) mod T` for `i = 1, 2, 3, ...`

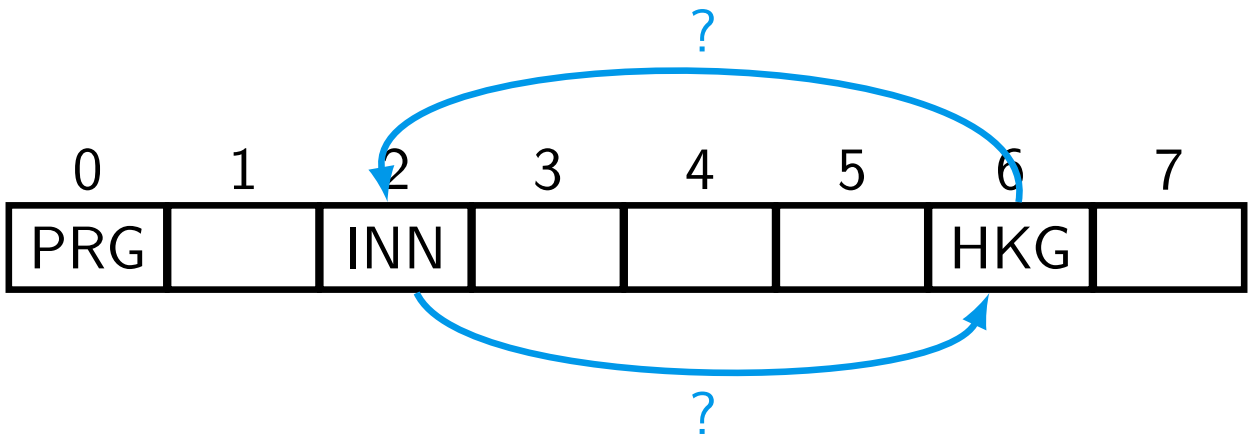
whereas double hashing's fallback positions are:

`(hash1(key) + i*hash2(key)) mod T` for `i = 1, 2, 3, ...`

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.