Dictionary Problem

How a Python dictionary is built

Cormen Sections 11.1, 11.2, and 11.4

No analysis

Given a set $S \subseteq U$ of keys (U is called universe $n = |S|$ (usually $n <<< m$) $U = \{0, \ldots, m-1\}$)

each key may be associated with a value

(Python dict vs set)

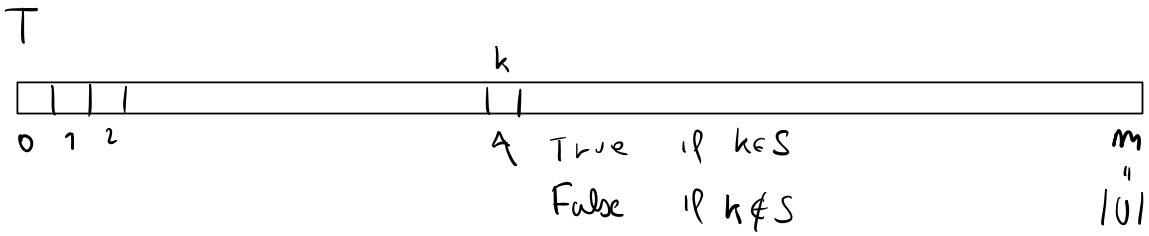
support the following operations:

- Insert (S, k) adds k to S
- Delete (S, k) deletes k from S
- Lookup (S, k) return True if $k \in S$, False otherwise
  (Search (S, k)) (return value associated with k if $k \in S$, N/C otherwise)

Python		dict	set
• Insert (S, k)	is	$S[k] = \ldots$	S.add(k)
• Delete (S, k)	is	del $S[k]$	S.remove(k)
• Lookup (S, k)	is	k in S	
• Search (S, k)	is	$S[k]$ returns the value	

Direct-address table

Simplest possible solution




```
Insert (T, k)
    T[k] = True        // O(1)
Delete (T, k
    T[k] = False       // O(1)
Lookup (T, k)
    return T[k]        // O(1)
```

in worst case

It may use too much space!!

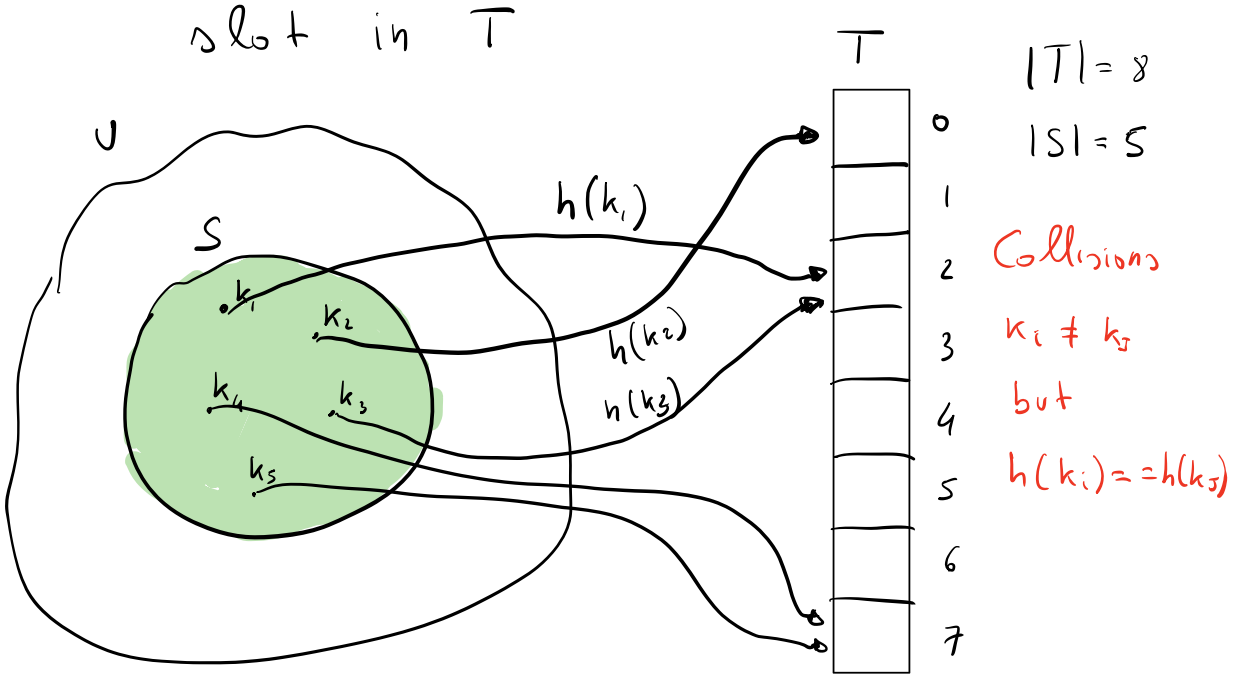
space is $\Theta(|U|)$ bits

Hash Table

Idea: Save space using a smaller table T
We use a table of size $\Theta(n)$
instead of $\Theta(m)$ as before

Issue key k cannot be stored in position $T[k]$ because position k may not exist

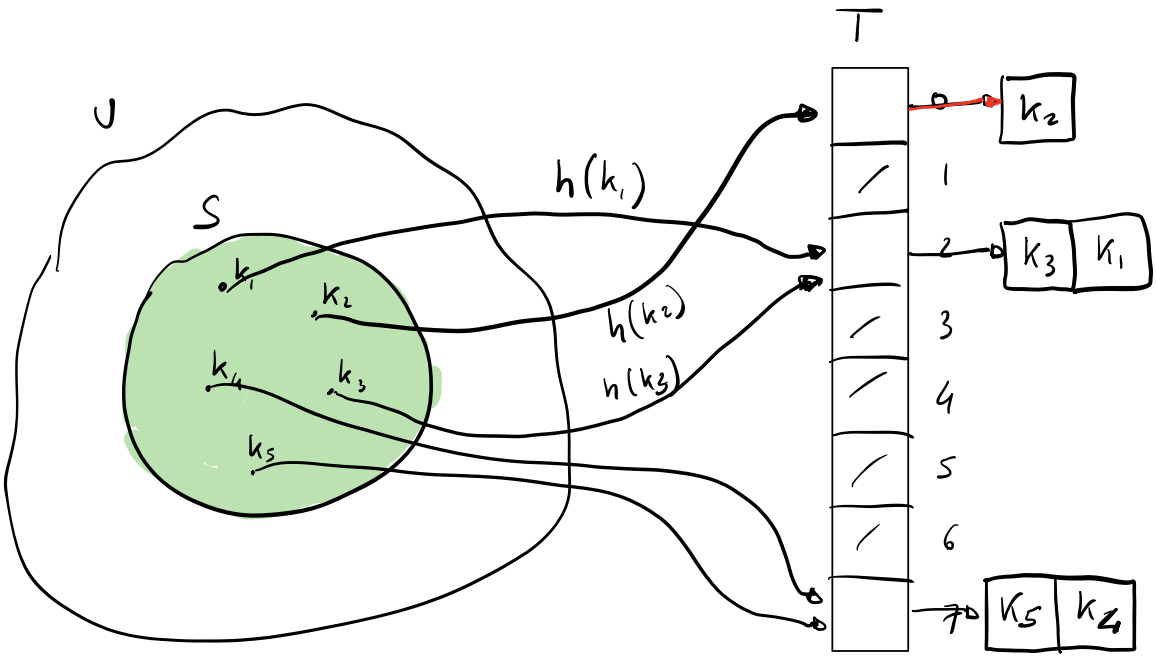
Solution Use a function $h : U \to \{0, 1, 2, \ldots |T|-1\}$
h is called hash function
we use h to map a key to its slot in T



Need a strategy to deal with collisions!

$h(k) = ((ak + b) \bmod p) \bmod |T|$ with a, b random and p a prime $\geq |T|$ (random)
h is fixed at the beginning

Hashing with chaining



Insert (S, k)
    p = h(k)
    T[p].append(k)

Delete(S, k)
    p = h(k)
    remove k from T[p]

Look up (S, k)
    p = h(k)
    return (k in T[p])

Complexity

- Insert $O(1)$ worst case
- Delete $\geq O(|T[p]|)$ worst case
- Look up

Good hash functions spread keys on T
so we have few collisions per entry
that's why they are randomized

Load factor $\alpha = \frac{|S|}{|T|}$ expected number of key per slot

if $|S| = n$ and $|T| = 2n$

then $\alpha = \frac{1}{2}$

so we expect $\frac{1}{2}$ key per slot

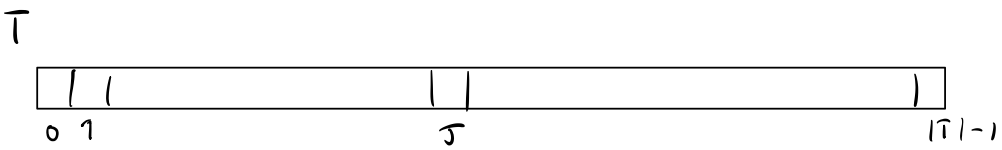
Lookup and Delete takes $O(1+\alpha)$ time
in expectation

Open Adressing

- Previous solution: a lot of space overhead due to lists
- here keys are stored in T, no overhead

- Each entry of T stores
  - a key of S
  - None
  - a special value for DELETED

Insert



Insert (S, 42) $\quad h(42) = 5$

if $T[h(42)] == \text{None}$ or $T[h(42)] = \text{DELETED}$

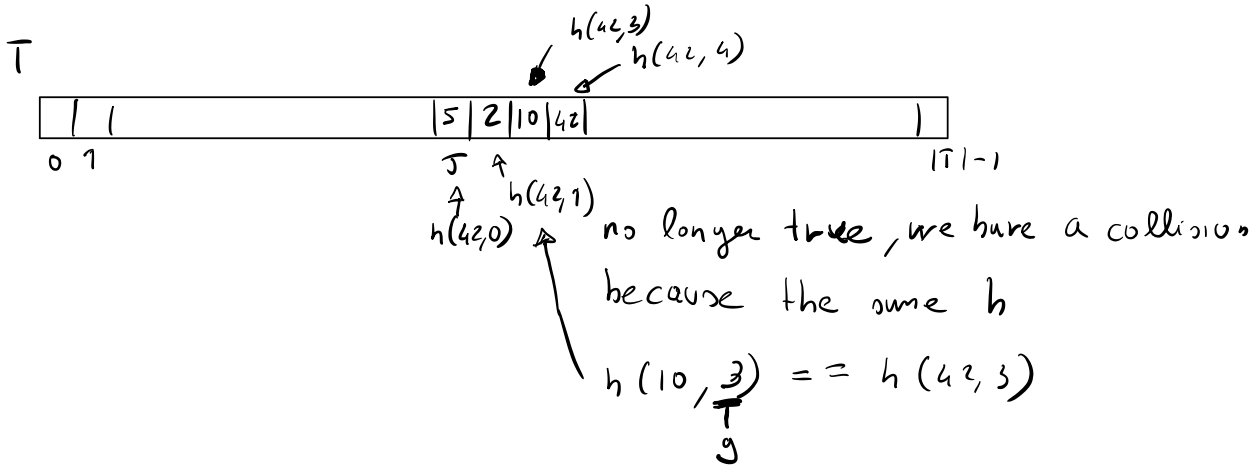
$T[h(42)] = 42$

if $T[h(42)] == \text{other key}$

we check (probe) a sequence of positions in T until we find an empty one

probe sequence $h(k,0), h(k,1), h(k,2) \ldots$

↖ # probe



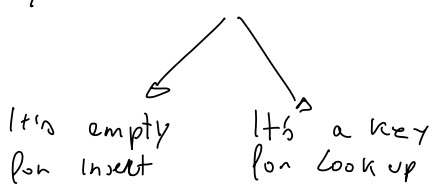

```
Insert ( T, k)
    i = 0
    repeat
        p = h (k, i)
        if  T[p] == None  or  T[p] == DELETED
                T[p] = k
                return  p
        i = i + 1
    until   i == m (=|T|)
    raise  "hash table overflow"
```

Searching

```
Lookup (T, k)
    i = 0
    repeat
        p = h(k, i)
        if T[p] == k
            return p (or True)
        i = i+1
    until T[p] == None or i == m
    return None (or False)
```

Deletion

If you delete a key at position p,
you cannot set T[p] to None !!!
we have to use special symbol DELETED



Complexity is $O(1 + \alpha)$ expected time for all ops
with good enough hash functions

Delete pseudo code by yourself

Probe sequences

Linear probing

$$h(k, i) = (h'(k) + i) \bmod |T|$$

- very cache-friendly 
- it creates long runs of used entries 

no matter what p is, I'm increasing the run by 1

Quadratic Probing (used by Python's dictionaries and sets)

$$h(k, i) = \left(h'(k) + \underbrace{c_1 i}_{\text{warmup be cache friendly}} + \underbrace{c_2 i^2}_{\text{run away if big run}}\right) \bmod |T|$$

c_1 and c_2 are random constants

Double hashing

$$h(k, i) = \left(h_1(k) + i\, h_2(k)\right) \bmod |T|$$