Dictionary Problem (How a (Python) dictionary is built)

Cormen Sects 11.1, 11.2 and 11.4

No analysis

Given a set $S \subseteq U$ of keys

(U is called universe, ... $U = \{0, \ldots, m-1\}$

e.g. $U = \{0, \ldots, 2^{32}-1\}$)

each key may be associated with a value

(Python dictonary vs set)

support the following operations:

- Insert (S, k) adds key k to set S
- Delete (S, k) removes key k from S
- Search (S, k) returns True if $k \in S$ False otherwise

(often called look up operation)

or returns the value associated with key k, if $k \in S$

or NIL otherwise

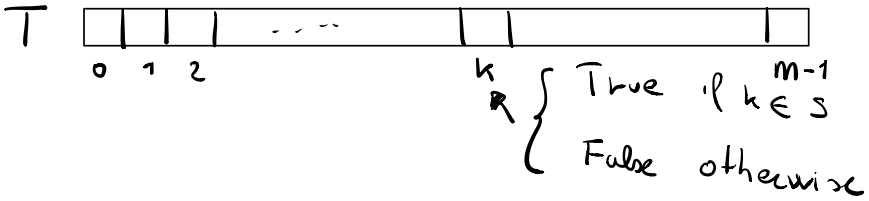
Search in Python

- "pippo" in dictionary { True / false
- dictionary["pippo"] returns the value

Direct-address table

Simplest possible solution

$S \subseteq U = \{0, 1, \ldots, m-1\}$ and m is small



Search (T, k)
    return T[k]

Insert (T, k)
    T[k] = True

Delete (T, k)
    T[k] = False

Complexity $O(1)$ worst time

It may use too much space!!!
if $m >> n = |S|$

Hash Tables

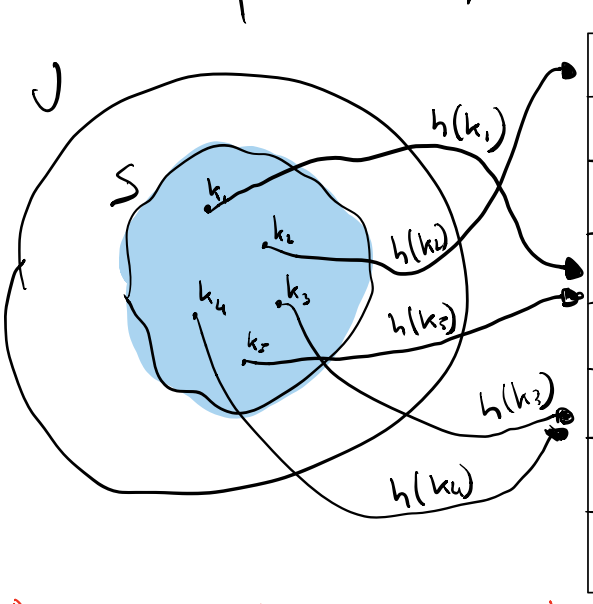
Idea: save space using a smaller table T
we use a table of size $\Theta(n)$
instead of $\Theta(m)$ as before.

Issue: key k cannot be stored in position $T[k]$ as with direct-access
why? because position k could not exist!

Solution: Use a (so called) hash function

$$h: U \longrightarrow \{0, 1, 2, \ldots, |T|-1\}$$

to compute the slot (or position) of the key k in the table T



$|T| = 8$

$|S| = 5$

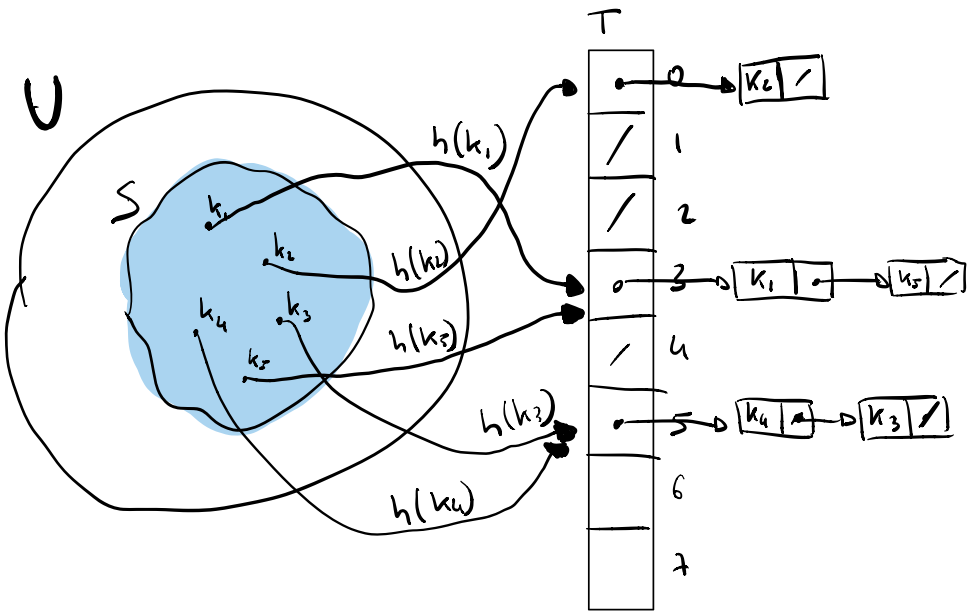
Problem:
Collisions!!!
$k_1 \neq k_2 \in S$
for which
$h(k_1) = h(k_2)$

Need strategies to deal with collisions!!!

Collisions resolution with chaining

Table T is not an array of keys but it's an array of lists.

Keys colliding in the same position are collected in the corresponding list



Insert (S, k)

Insert k at head (or tail) of list T[h(k)]

Delete (S, k)

remove k from T[h(k)]

Search (S, k)

scan T[h(k)] looking for key k

Complexity

- Insert $O(1)$ time worst case
- Delete and Search $\Theta(|T[h(k)]|)$ time

In the worst case $|T[h(k)]| = \Theta(n)$

i.e. we scan the whole set

We need good hash functions which spread keys on whole table so that on average there are few collisions

Goal of hash function: behaves as random as possible

Load factor: $\alpha = \frac{|S|}{|T|}$ ← size of set / ← size of table

$|S| = n \qquad |T| = 2n$

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

we expect $\alpha = \frac{1}{2}$ keys per slot

There are (a lot of) good hash functions (called universal)

$$h(k) = \big(\big((ak + b) \bmod p\big)\big) \bmod |T|$$

where p is a (randomly chosen) prime

a and b are random values in $\{1, \ldots, p-1\}$

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

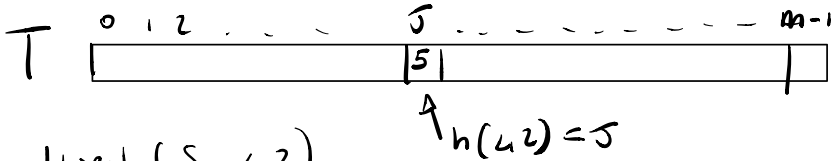
p, a, b are chosen at the beginning

Open Addressing

- Keys occupy the table itself
  so we save space for auxiliary info (e.g., pointers)
  and use larger table.

- Each entry of T contains
  - a key of S
  - NIL
  - DELETED

Insertion

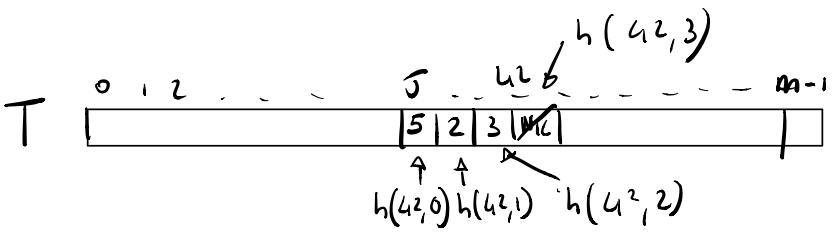


Insert (S, 42)

we check (probe) a sequence of entries of T
until we find an empty one (e.g., NIL)

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

↑ probe



```
Insert (T, k)
1   i = 0
2   repeat
3       j = h(k, i)
4       if T[j] == NIL or T[j] == DELETED
5           T[j] = k
6           return j
7       else i = i + 1
8   until i == m (= |T|)
9   error "hash table overflow"
```

Searching

Look at key k by using k's probe sequence
Stop either if we find k or we find NIL

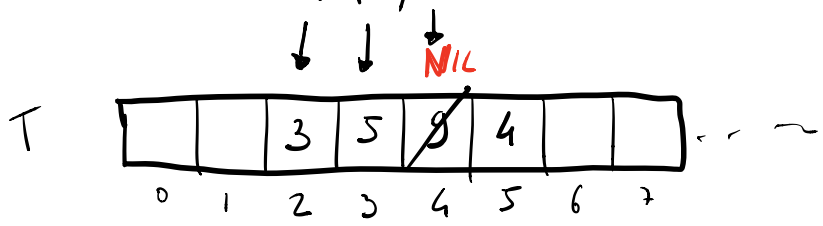
```
Search (T, k)
1   i = 0
2   reapeat
3       j = h(k, i)
4       if T[j] == k
5           return j
6       i = i + 1
7   until T[j] == NIL or i == m
8   return NIL
```

Deletion

Deletion is the most difficult op

When we delete a key from slot i

we cannot simply set T[i] = NIL. Why?



Search (T, 4)

We have to use DELETED
- It is empty for Insert
- It is not empty for Search

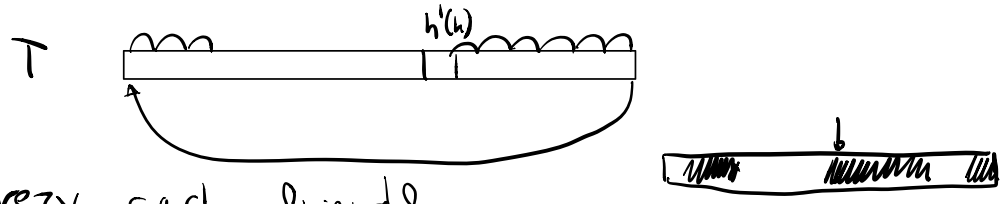
Complexity all ops in $O(1+\alpha)$ expected time with good enough hash functions

Probe sequences

$m = |T|$

Linear probing

$$h(k,i) = (h'(k) + i) \bmod m$$

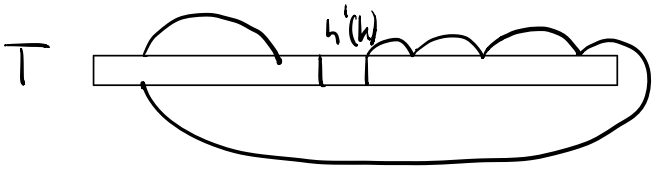


- very cache-friendly
- it create long runs of used entries

Quadratic Probing (used by Python's dictionary implementation)

$$h(k,i) = \left(h'(k) + c_1 i + c_2 i^2\right) \bmod m$$

c_1 and c_2 are random constants



Double hashing

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

```
Delete (T, k)
1   i = 0
2   repeat
3       j = h(k, i)
4       if T[j] == k
5           T[j] = DELETED; return j
6       i = i + 1
7   until T[j] == NIL or i == m
8   return NIL
```

Exercises

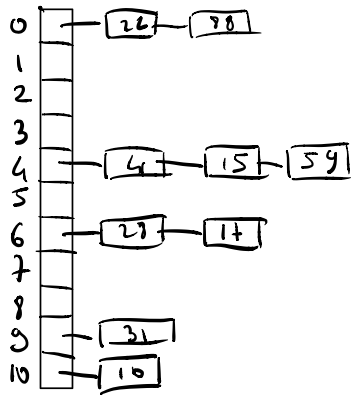
① Hashing with chaining

$|T| = 11 \qquad h(k) = k \bmod 11$

$S = \{10, 22, 31, 4, 15, 28, 17, 88, 59\}$

$h(S) = \{10, 0, 9, 4, 4, 6, 6, 0, 4\}$

T



(2) Open Addressing with linear probing

(a)

$|T| = 11$ $\quad h(k,i) = ((\underbrace{k \bmod 11}_{h'(k)} + i) \bmod 11$

$S = \{10, 22, 31, 4, 15, 28, 17, 88, 59\}$

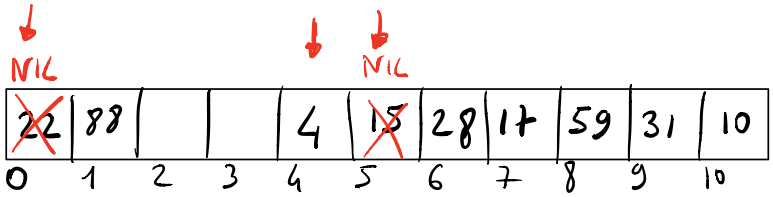
$h(S) = \{10, 0, 9, 4, 4, 6, 6, 0, 4\}$

0	1	2	3	4	5	6	7	8	9	10
22	88			4	15	28	17	59	31	10

$h(15,0) = ((15 \bmod 11) + 0) \bmod 11 = 4$

$h(15,1) = (15 \bmod 11) + 1 \bmod 11 = 5$

(b) Show an example of a Delete(k) and Search(k') so that Search would be incorrect without the use of DELETED.



Delete(22) Delete(15)

Search(88) Search(59) Search(28)

$h(28,0) = 6$