15-112
Fundamentals of Programming

Week 3 - Lecture 3:
Sets and dictionaries.

June 8, 2017

The Plan

> Efficient data structures: sets and dictionaries

What is a data structure?

A data structure allows you to store and maintain a collection of data.

It should support basic operations like:

- add an element to the data structure
- remove an element from the data structure
- find an element in the data structure

…

What is a data structure?

A list is a data structure.

It supports basic operations:

- append( ) $O(1)$
- remove( ) $O(N)$
- **in** operator, index( ) $O(N)$

...

One could potentially come up with a different structure which has different running times for basic operations.

Motivating example

Car license plates

-Order number is arbitrary, some numbers may not exists

-Want to find if a plate with a certain number already exists.

How long does it take to search if a license plate exists ?

$$O(N)$$

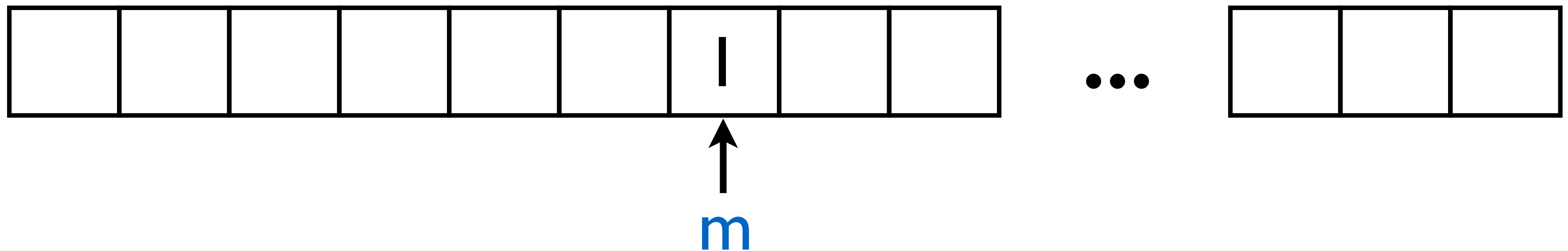
What if I know all the numbers are less than 10000 ?

Motivating example

Solution:

Create a list of size 10000.

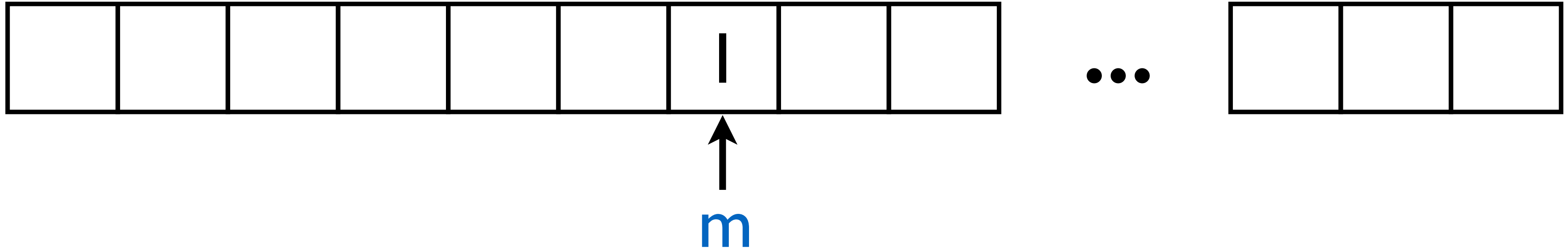
Put number m at index m.



What is the running time for searching for an element?

$O(1)$

Motivating example



The sweet idea:

Connecting value to index.

Motivating example

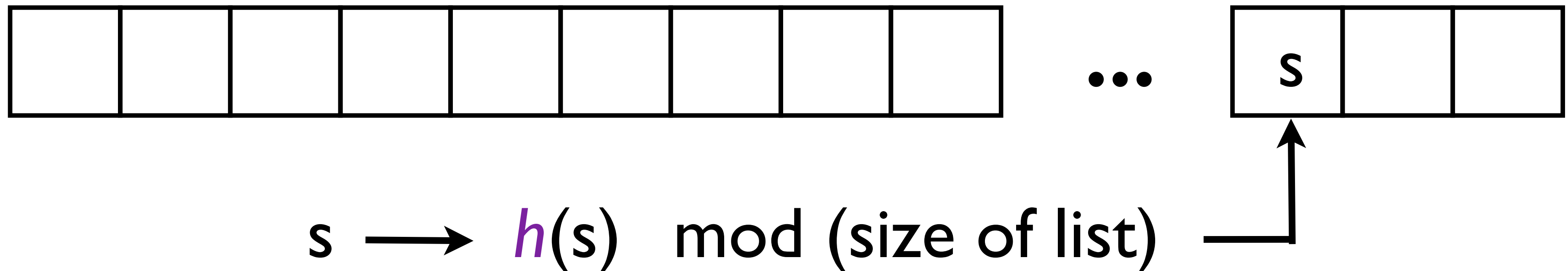
Questions

What if the numbers are not bounded by 10000 ?

What if the plates contain letters as well ?

Extending the sweet idea

Storing a collection of strings?



Start with a certain size list (e.g. 10000)

Pick a function *h* that maps strings to numbers.

Store s at index *h*(s) mod (size of list)

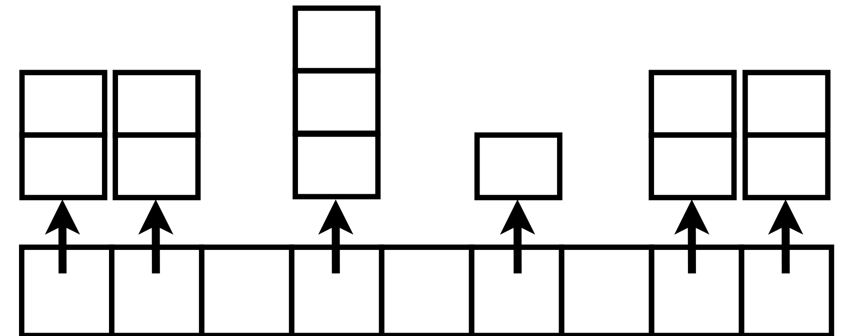
h is called a **hash function**.

Extending the sweet idea

Potential Problems

Collision: two strings map to the same index

List fills up



Fixes

The hash function should be "random"
so that the likelihood of collision is not high.

Store multiple values at one index (bucket)
(e.g. use 2d list)

When buckets get large (say more than 10),
resize and **rehash**: pick a larger list, rehash everything

Extending the sweet idea

What did we gain:

Basic operations add, remove, find/search **super fast**

(sometimes (infrequently) we need to resize/rehash)

What did we lose:

No mutable elements

No order

Repetitions are not good

Sets

Introducing sets

Sets:

- a non-sequential (unordered) collection of objects
- immutable elements
- no repetitions allowed
- look up by object's value
  - finding a value is super efficient

- supports basic operations like:

s.add(x), s.remove(x), s.union(t), s.intersection(t)
x **in** s

Creating a set

```
s = set()
s = set([2, 4, 8])                  # {8, 2, 4}
s = set([“hello”, 2, True, 3.14])   # {“hello”, True, 2, 3.14}
s = set([2, 2, 4, 8])               # {8, 2, 4}
s = set([2, 4, [8]])                # Error
```

(sets are mutable, but its elements must be immutable.)

```
s = set(“hello”)                    # {‘e’, ‘h’, ‘l’, ‘o’}
s = set((2, 4, 8))                  # {8, 2, 4}
s = set(range(10))                  # {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Set methods

Returns a new set (non-destructive):

s.copy()

s.union(t), s.intersection(t),
s.difference(t), s.symmetric_difference(t)

Modifies s (destructive):

s.pop(), s.clear()

s.add(x), s.remove(x), s.discard(x)

s.update(t), s.intersection_update(t),
s.difference_update(t), s.symmetric_difference_update(t)

Other:

s.issubset(t), s.issuperset(t)

The advantage over lists

1

```
print(5000 in s)        # Super fast
print(-1 not in s)      # Super fast
s.remove(100)           # Super fast
```

Essentially $O(1)$

Example: checking for duplicates

Given a list, want to check if there is any element appearing more than once.

Dictionaries (Maps)

Dictionaries / maps

Lists:

- a sequential collection of objects
- can do look up by index (the position in the collection)

Dictionaries:

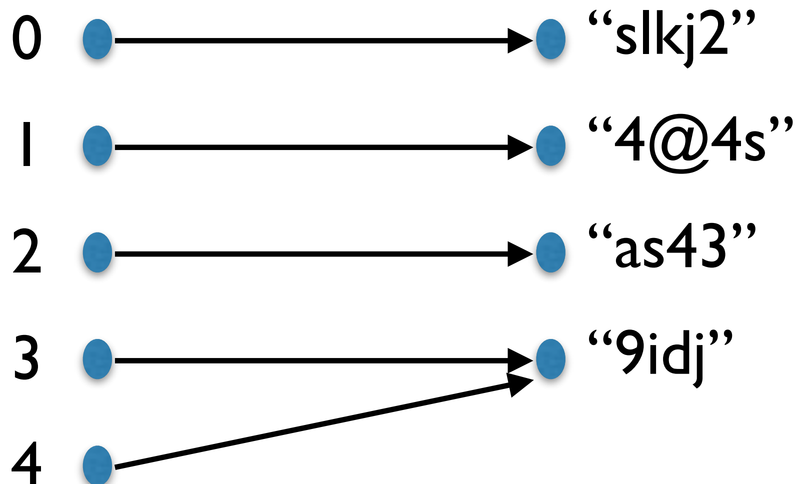
- a non-sequential (unordered) collection of objects
- a more flexible look up by keys
-

Dictionaries / maps

5 users, store user passwords

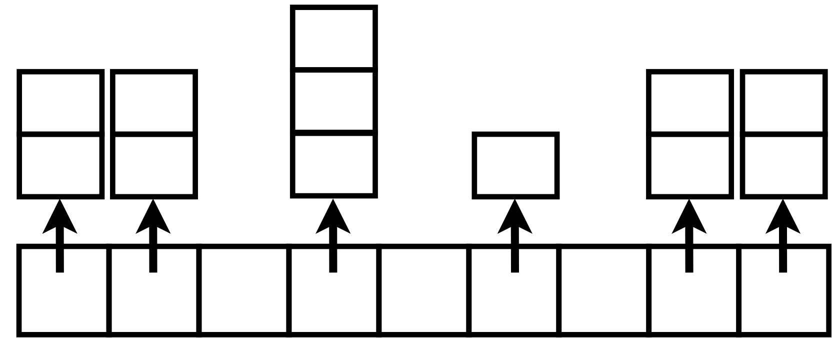
```
a = [None]*5
a[0] = "slkj2"
a[1] = "4@4s"
a[2] = "as43"
a[3] = "9idj"
a[4] = "9idj"
```

List

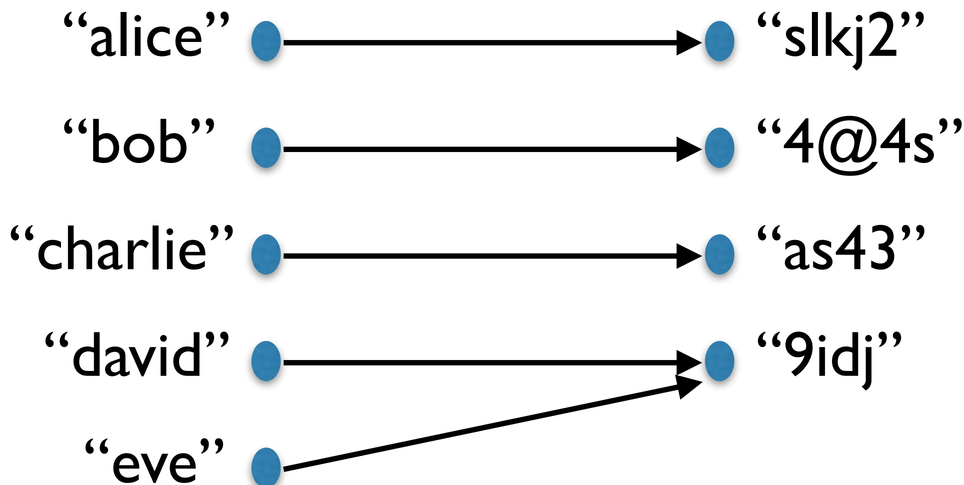


Dictionaries / maps

d = dict()
d[“alice”] = “slkj2”
d[“bob”] = “4@4s”
d[“charlie”] = “as43”
d[“david”] = “9idj”
d[“eve”] = “9idj”



- hash using the key
- store (key, value) pair



Properties:

- unordered
- values are mutable
- keys form a set (immutable, no repetition)

Dictionaries / maps

Creating dictionaries

```
users = dict()

users[“alice”] = “sl@3”

users[“bob”] = “#$ks”

users[“charlie”] = “slk92”
```

```
users = {“alice”: “sl@3”, “bob”: “#$ks”, “charlie”: “slk92”}
```

```
users = [(“alice”, “sl@3”), (“bob”, “#$ks”), (“charlie”, “#242”)]

users = dict(users)
```

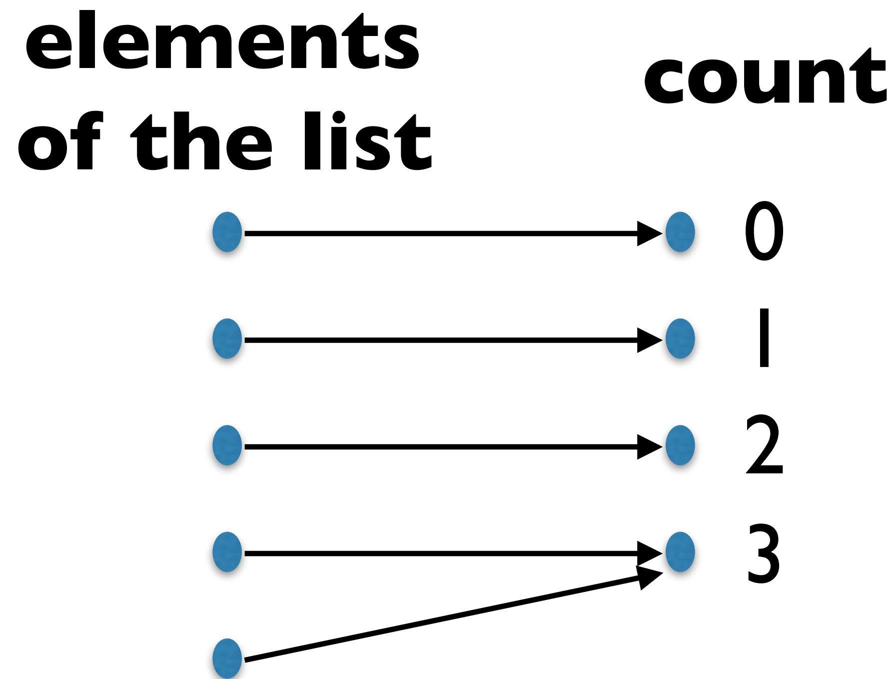
Dictionaries / maps

```
users = {“alice”: “sl@3”, “bob”: “#$ks”, “charlie”: “slk92”}


for key in users:
    print(key, d[key])


print(users[“frank”])              Error

print(users.get(“frank”))          # prints None

print(users.get(“frank”, 0))       # prints 0
```

Example: Find most frequent element

Input: a list of integers

Output: the most frequent element in the list



Exercise: Write the code.