Hashing

Introduction to Computer Science
Iris Howley
TODAY’S LESSON

Hashing

(Arranging dictionary keys to find values quickly)
Dictionary Keys

```python
>>> d = dict()
>>> d[['a',1]] = 'testing'
TypeError: unhashable type: 'list'
>>> d[('a',1)] = 'testing'
```

What’s the difference?
Dictionary Keys

Dictionary keys must be immutable types
	int, float, string, bool, tuple, frozenset

Why?
Mutable Types as Dictionary Keys (*No!*)

- Lists are mutable

- When you `append()` to a list, it changes that list object

- If you used a list object as a key in a dictionary, you wouldn’t be able to find it again, after it’s been changed

*We’re going to see why!*
Mutable Types as Dictionary Keys (*No!*)

If you used a list object as a key in a dictionary, you wouldn’t be able to find it again, after it’s been changed

```python
mylist = ['a', 'b']
mydict = dict()
mydict[mylist] = 'throws an error'
mylist.append('c')
print(mydict[mylist])
# Now mylist is no longer findable in the dict!
```

We’re going to see why!
Dictionary Keys

• Dictionaries index their items by a hash
• A hash is a fixed sized integer that identifies a particular value.
• Each value needs to have its own hash
  ▪ For the same value you will get the same hash even if it's not the same object.

Why not just index items based on their value?
Hashing
Hashing
Hashing

FIND:
Hashing

FIND:
Hashing

Why not just index items based on their value?

• We could organize all words in memory by the letter they start with...

• But words that start with ‘A’ could be numerous

• Compared to words that start with ‘Z’
  ▪ ...Sort of like arranging clothes by color

• Hashing is a different way of mapping items to make them easier to find
Hashing

• Other concerns
  ▪ Bad hashing function for your data, resulting in clustering
  ▪ Running out of space in the pile you’ve assigned
  ▪ Placing shirts in the wrong pile

• Goal: store in the order that makes it easiest to look them up
QUESTIONS?

Please contact me!
Prior to this lecture...

Complete:
1. POGIL: Hashing
   • Glow > Modules
TODAY’S LESSON
Hashing – How

(How we arrange dictionary keys to find values quickly)
Python Hash Function

```python
hash(obj)
```

- It calls special method: `obj.__hash__(self)`
- Used for dictionary keys and sets
- Calculates an int for `obj` that ideally results in:
  - Minimal clustering (i.e., even distribution)
  - Same values generate the same hash value
hash(obj)

• >>> s = 'hello world'
• >>> s2 = 'hello world'
• >>> hash(s) → 4963799451833479185
• >>> hash(s2) → 4963799451833479185
• >>> s is s2 → False

If the 2 strings are the same, they’ll get the same hash
...even if they're different objects!
hash(obj)

>>> s = 'hello world'
>>> hash(s) → 4963799451833479185
>>> exit()
-> python3
>>> s = 'hello world'
>>> hash(s) → 4686556288558268365

You cannot assume that the same values will get the same hash values across different sessions of python!
hash(obj)

- $s = 'hello world'$
- $t = s + '!'$

- $\text{hash}(s) \rightarrow 4963799451833479185$
- $\text{hash}(t) \rightarrow -8774050965770600213$
- $\text{hash}(t[:\text{-}1]) \rightarrow 4960501519247167238$

If the 2 strings are different, they *might* get a different hash. (an even distribution of objects may result in some overlap)
Some hash codes are expensive (million-long tuple)

- hash(1) → 1
- hash(2) → 2
- hash(1000000000000000000) → 10000000000000000000
- hash(10000000000000000000) → 776627963145224196

At some length, it starts treating the numbers like a string
If the hash codes are the same, the values *might* be the same
Hash Tables

• Python's dictionary is an implementation of a more widely known data structure called a Hash Table

• Let's walk through an example with this dictionary:
  
d = {'tally': 'bananas', 'linus': 'everything', 'pixel': 'cheese', 'wally': 'carrots'}

• (dog names mapped to their favorite foods)
# Hash Tables

## How to access mydict[‘wally’]?

<table>
<thead>
<tr>
<th>Keys</th>
<th>Hashes</th>
<th>Buckets</th>
<th>Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>'pixel'</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'tally'</td>
<td>1</td>
<td>tally, bananas</td>
<td></td>
</tr>
<tr>
<td>'wally'</td>
<td>2</td>
<td>linus, everything</td>
<td></td>
</tr>
<tr>
<td>'linus'</td>
<td>3</td>
<td>pixel, cheese</td>
<td></td>
</tr>
</tbody>
</table>

### Collision!
- What to do with Wally?
  - Could re-hash into new table and increase # buckets...
  - ...or...

- 'wally' hash results in a collision with 'pixel'.
Immutable Objects

• Have no way to set/change the attributes, without creating a new object
  ▪ Like int, string, etc.
  ▪ User-defined types: __slots__ = []

• Can be used as keys for dictionaries
  ▪ If the class has __hash__() and __eq__() methods defined!

https://docs.python.org/3/reference/datamodel.html#object.__hash__
Immutable Objects

• Have no way to set/change the attributes, without creating a new object
  ▪ Like int, string, etc.
  ▪ __slots__ = []

• Can be used in sets
  ▪ i.e., you cannot have a set of lists

```python
>>> s = {[1,2,3], [1], [2,3]}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```
Thought Question

How would we implement a good hash function for a user-defined class?

```python
>>> class Flower:
...     slots = ['sepals', 'petals']
...     def __hash__(self):
...         return self.petals + self.sepals

>>> rose = Flower()
>>> rose.petals = 10
>>> rose.sepals = 5
>>> hash(rose)
15
```

Would this be evenly distributed? How to improve?!
Thought Question

How would we implement a good hash function for a user-defined class?

What about for the Scotus class?

What about for Plaintext class?

def __hash__(self):
    return '???'

QUESTIONS?

Please contact me!
Leftover Slides
The diagram illustrates a hash table with keys and their associated values. The keys are 'tally', 'linus', and 'pixel', each pointing to a bucket that contains a list of values.

- For 'tally', the hash value is 2458, and the bucket contains 'tally' and 'bananas'.
- For 'linus', the hash value is 3083, and the bucket contains 'tally' and 'bananas'.
- For 'pixel', the hash value is 3998, and the bucket contains 'pixel' and 'cheese'.
- For 'linus', the hash value is 4360, and the bucket is empty.
- For 'pixel', the hash value is 4360, and the bucket is empty.
- For 'linus', the hash value is 7104, and the bucket contains 'linus' and 'everything'.
Keys
'tally'
'linus'
'pixel'

Hash (Index)
2458
3083
3998
4360
7104

Buckets

'tally' bananas
pixel cheese
linus everything

Overflow
x wally carrots