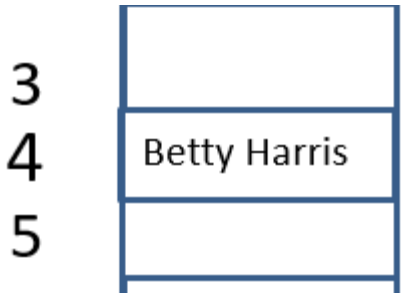
Compsci 101
Sorting and why Dictionaries are so fast



Susan Rodger

November 22, 2022

V is for …

- **Viral Video**
  - Husky Dog sings with iPAD – 18 million views
  - **https://www.youtube.com/watch?v=Mk4bmK-acEM**
- **Virtual Memory**
  - It is and is not there!
- **Virtual Reality**
  - Augmenting IRL
  - http://bit.ly/vr-playlist

danah boyd

Dr. danah boyd is a Principal Researcher at Microsoft Research, … and a Visiting Professor at New York University. Her research is focused on addressing social and cultural inequities by understanding the relationship between technology and society.

"If I have learned one thing from my research, it's this: social media services like Facebook and Twitter are providing teens with new opportunities to participate in public life, and this, more than anything else, is what concerns many anxious adults."

Interested in being a UTA?

- **Enjoy Compsci101?**
- **Would like to help others learn it?**

- **Consider applying to join the team!**
- **https://www.cs.duke.edu/undergrad/uta**

- **Apply soon**

Announcements

- **APT-6 due Tuesday, November 29**

- **Assign 6 – Recommender, due Tuesday, 12/6**
  - Must be turned in by 12/7! No extensions!
  - Start Early!

- **No lab this week! No class Thursday! Enjoy the break**

- **Exam 3 – Thursday, Dec 1**

PFTD

- **Review Sorting**
- **Some Sorting APTs**
- **Why Dictionaries are so fast**

WOTO-1 Review Sorting
http://bit.ly/101f22-1122-1

When and What's in CompSci 101

- **Problem to solve**
  - Use 7 steps
  - Step 5: How do you translate algorithm to code?
    - What do you use to solve it?
    - When do you use it?

What are the “what’s”?

- **Data Structures: list, set, dictionary, tuple**
  - Combined: list of lists, dictionary of key to list
- **Loops : from for to while, index loop**
- **Other:**
  - List comprehensions
  - Parallel lists
  - Lambda
  - If…if…if
  - If…elif…else

Quick When's and What's for 101

- **Whichever makes more sense to you:**
  - Parallel lists vs dictionaries
  - If…if…if vs if…elif…else
  - List comprehension vs for loop
  - Tuples vs Lists
    - If you want to prevent mutation -> tuples
  - Need single line function
    - Lambda vs create normal helper function

APT – Sorted Freqs

APT SortedFreqs

Problem Statement

The frequency with which data occurs is sometimes an important statistic. In this problem you'll determine how frequently strings occur and return a list representing the frequencies of each different/unique string. The list returned contains as many frequencies as there are unique strings. The returned frequencies represent an alphabetic/lexicographic ordering of the unique words, so the first frequency is how many times the alphabetically first word occurs and the last frequency is the number of times the alphabetically last word occurs.

Specification

```
filename: SortedFreqs.py

def freqs(data):
    """
    return list of int values corresponding
    to frequencies of strings in data, a list
    of strings
    """
```

Consider these strings (quotes for clarity, they're not part of the strings).

```
["apple", "pear", "cherry", "apple", "cherry", "pear", "apple", "banana"]
```

The list returned is `[3,1,2,2]` since the alphabetically first word is `"apple"` which occurs 3 times; the second word alphabetically is `"banana"` which occurs once, and the other words each occur twice.

What's the best way to …

- **SortedFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortedfreqs.html

- **Count how many times each string occurs**
  - Create d = {}, iterate over list updating values
- **OR**
  - Use data.count(w) for each w

APT: SortByFreqs

APT SortByFreqs

Problem Statement

Specification

```
filename: SortByFreqs.py

def sort(data):
    """
    return list of strings based on
    the list of strings in parameter data
    """
```

The frequency with which data occurs is sometimes an important statistic. In this problem you are given a list of strings and must determine how frequently the strings occur. Return a list of strings that is sorted (ordered) by frequency. The first element of the returned list is the most frequently occurring string, the last element is the least frequently occurring. Ties are broken by listing strings in lexicographic/alphabetical order. The returned list contains one occurrence of each unique string from the list parameter.

Consider these strings (quotes for clarity, they're not part of the strings).

```
["apple", "pear", "cherry", "apple", "pear", "apple", "banana"]
```

The list returned is:

```
[ "apple", "pear", "banana", "cherry" ]
```

since the most frequently occurring string is `"apple"` which occurs 3 times; the string `"pear"` occurs twice and the other strings each occur once so they are returned in alphabetical order.

Wait, wait, but what's …

- **SortByFreqs**
  - https://www2.cs.duke.edu/csed/pythonapt/sortbyfreqs.html

- **Sort by # occurrences high to low**
  - Tuples with count/lambda and reverse=True?
  - Break ties in alphabetical order: two passes

WOTO-2
http://bit.ly/101f22-1122-2

How do Dictionaries work so fast?

- **How are they implemented?**

Review: Problem – which word occurs the most frequently in a file

- **Need to count how many times each word occurs**
  - slowcount – for each word in a set, calls count
  - fastcount – counting dictionary

slowcount function
Short Code and Long Time

- **See module WordFrequencies.py**
  - Find # times each word in a list of words occurs
  - We have tuple/pair: word and word-frequency

```
37  def slowcount(words):
38      pairs = [(w,words.count(w)) for w in set(words)]
39      return sorted(pairs)
```

- **Think: How many times is `words.count(w)` called?**
  - Why is **`set(words)`** used in list comprehension?

Using fastcount

- **Update count if we've seen word before**
  - Otherwise it's the first time, occurs once

```
28  def fastcount(words):
29      d = {}
30      for w in words:
31          if w in d:
32              d[w] += 1
33          else:
34              d[w] = 1
35      return sorted(d.items())
```

Let’s run them and compare them!

- **Run with Melville and observe time**
  - slowcount about 0.76 seconds
  - fastcount about 0.00 seconds

- **Run with Hawthorne and observe time**
  - slowcount about 14.6 seconds
  - fastcount about 0.03 seconds

Here is the idea behind how dictionaries work

Simple Example

Want a mapping of Soc Sec Num to Names

- **Duke's CS Student Union wants to be able to quickly find out info about its members. Also add, delete and update members. Doesn't need members sorted.**

  267-89-5431 John Smith

  703-25-6141 Ademola Olayinka

  319-86-2115 Betty Harris

  476-82-5120 Rose Black

- **Dictionary d – SSN to names**
  - d['267-89-5431'] = 'John Smith'
  - How does it find 'John Smith' so fast?

Dictionary d(SSN) = (SSN, name)

- **We actually would map the SSN to the tuple of (SSN, name).**
- **That is a lot to display on a slide, so we will just show SSN to name**
- **But remember name is really a tuple of (SSN,name)**

Simple Example

Let's look under the hood.
How are dictionaries implemented?

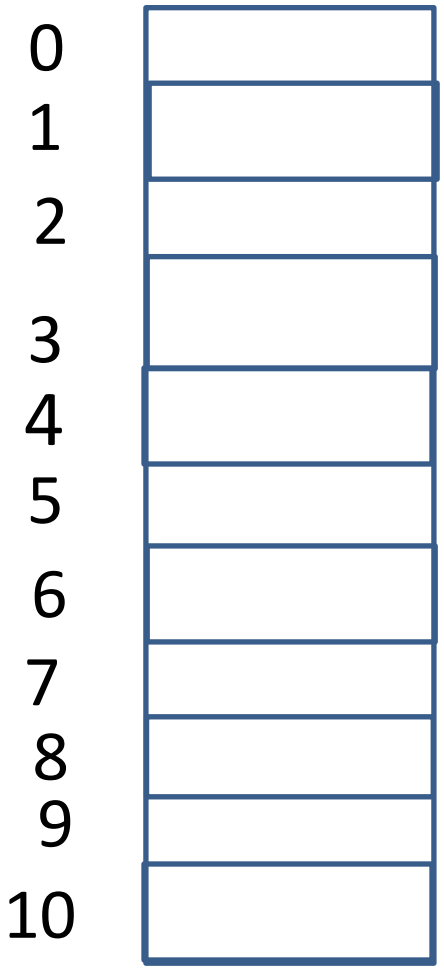
- **Dictionaries implemented with a list, in a clever way**
- **How do we put something into the list fast?**
- **How do we find it in the list quickly?**
  - **d['267-89-5431'] = 'John Smith'**
- **List size is 11 – from 0 to 10**
- **d['267-89-5431'] calculates index location in list of where to put this tuple (SSN,name)**
- **Use a function to calculate where to store 'John Smith**
  - H(ssn) = (last 2 digits of ssn) mod 11
  - Called a Hash function

Have a list of size 11 from 0 to 10

- **Insert these into the list**
- **Insert as (key, value) tuple**

(267-89-5431, John Smith)

(in example, only showing name)



WOTO-3 How Dictionaries Work
http://bit.ly/101f22-1122-3

APT WordPlay

APT: WordPlay

Problem Statement

Given a phrase of words, your task is to return a string of the unique words from the phrase, with the words sorted using the following rules.

1. First the unique words should be sorted in reverse order based on their length (number of characters in the word)
2. For words the same length, they should be sorted in alphabetical order based on only the first letter of each such word
3. If there are ties after 1) and 2) criteria, then sort those words in reverse alphabetical order based on the last letter of each such word
4. If there are ties after 1), 2) and 3) criteria, then sort those words in alphabetical order based on the sub-word between the first and last letter of each such word.

APT WordPlay example

"mouse elephant moth zebra mole tiger moose moth mule"

Returns:

"elephant moose mouse tiger zebra moth mole mule"

WOTO-4 APT WordPlay
http://bit.ly/101f22-1122-4

APT: WordPlay

Problem Statement

Given a phrase of words, your task is to return a string of the unique words from the phrase, with the words sorted using the following rules.

1. First the unique words should be sorted in reverse order based on their length (number of characters in the word)
2. For words the same length, they should be sorted in alphabetical order based on only the first letter of each such word
3. If there are ties after 1) and 2) criteria, then sort those words in reverse alphabetical order based on the last letter of each such word
4. If there are ties after 1), 2) and 3) criteria, then sort those words in alphabetical order based on the sub-word between the first and last letter of each such word.

WOTO-4 APT WordPlay
http://bit.ly/101f22-1122-4