CompSci 101
Introduction to Computer Science

Nov. 22, 2016
Prof. Rodger
Announcements

• Reading and RQ due next time
• Assignment 7 due next Tuesday
  – Assignment 8 and 9 out soon
• APT 9 out and due in a week and a half

• Today: Solving problems
  – How do change how things are sorted?
    • Other than ordering and re-ordering tuple
    • How do Python .sort and sorted() stack up?
Clever Hangman - Dictionary

• Builds a dictionary of categories
• Start with list of words of correct size
• Repeat
  – User picks a letter
  – Make dictionary of categories based on letter
  – New list of words is largest category
    • Category includes already matched letters
    • List shrinks in size each time
Clever Hangman Example

• Possible scenario after several rounds
  (secret word: calls) # words possible 176
  You guessed a letter
  You have this many guesses left: 4
  Letters not guessed: bcdghijklmnpqrstuvwxyz
  guessed so far: _ a ___ 
  guess a letter or enter + to guess a word: d

• From list of words with a the second letter. From that build a dictionary of list of words with no d and with d in different places:

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>a</em>__</td>
<td>147</td>
</tr>
<tr>
<td><em>add</em></td>
<td>1</td>
</tr>
<tr>
<td><em>a_d</em></td>
<td>17</td>
</tr>
<tr>
<td><em>ad</em>_</td>
<td>3</td>
</tr>
<tr>
<td>dadd_</td>
<td>1</td>
</tr>
<tr>
<td>da_d_</td>
<td>1</td>
</tr>
<tr>
<td>da___</td>
<td>6</td>
</tr>
</tbody>
</table>

  Choose “no d”, most words, 147
  Only 17 words of this type
  Only 1 word of this type
Clever Hangman

• How to start? How to modify assignment 5?
Playing go-fish, spades, or …

• Finding right card?
  – What helps?
  – Issues here?

• Describe algorithm:
  – First do this
  – Then do this
  – Substeps ok
  – When are you done?
Problem Solving with Algorithms

• Top 100 songs of all time, top 2 artists?
  – Most songs in top 100
  – Wrong answers heavily penalized
  – You did this in lab, you could do this with a spreadsheet

• What about top 1,000 songs, top 10 artists?
  – How is this problem the same?
  – How is this problem different
Scale

• As the size of the problem grows …
  – The algorithm continues to work
  – A new algorithm is needed
  – New engineering for old algorithm

• Search
  – Making Google search results work
  – Making SoundHound search results work
  – Making Content ID work on YouTube
import csv, operator

f = open('top1000.csv','rbU')
data = {}
for d in csv.reader(f,delimiter=',',quotechar='"'):    
    artist = d[2]
song = d[1]
if not artist in data:
    data[artist] = 0
    data[artist] += 1

itemlist = data.items()
.dds = sorted(itemlist,key=operator.itemgetter(1),reverse=True)
print dds[:30]
Understanding sorting API

• How API works for `sorted()` or `.sort()`
  
  - Alternative to changing order in tuples and then changing back

```
x = sorted([(t[1],t[0]) for t in dict.items()])
x = [(t[1],t[0]) for t in x]
```

```
x = sorted(dict.items(), key=operator.itemgetter(1))
```

• Sorted argument is key to be sorted on, specify which element of tuple. Must import library operator for this
Sorting from an API/Client perspective

- API is Application Programming Interface, what is this for sorted(..) and .sort() in Python?
  - Sorting algorithm is efficient, stable: part of API?
  - sorted returns a list, doesn't change argument
  - sorted(list, reverse=True), part of API
  - foo.sort() modifies foo, same algorithm, API

- How can you change how sorting works?
  - Change order in tuples being sorted,
    - [(t[1], t[0]) for t in ...]
  - Alternatively: key=operator.itemgetter(1)
Beyond the API, how do you sort?

• Beyond the API, how do you sort in practice?
  – Leveraging the stable part of API specification?
  – If you want to sort by number first, largest first, breaking ties alphabetically, how can you do that?

• Idiom:
  – Sort by two criteria: use a two-pass sort, first is secondary criteria (e.g., break ties)

[('ant', 5), ('bat', 4), ('cat', 5), ('dog', 4)]

[('ant', 5), ('cat', 5), ('bat', 4), ('dog', 4)]
Two-pass (or more) sorting

- Because sort is stable sort first on tie-breaker, then that order is fixed since stable

```python
a0 = sorted(data, key=operator.itemgetter(0))
a1 = sorted(a0, key=operator.itemgetter(2))
a2 = sorted(a1, key=operator.itemgetter(1))
data
[('f', 2, 0), ('c', 2, 5), ('b', 3, 0),
 ('e', 1, 4), ('a', 2, 0), ('d', 2, 4)]
a0
[('a', 2, 0), ('b', 3, 0), ('c', 2, 5),
 ('d', 2, 4), ('e', 1, 4), ('f', 2, 0)]
```
Two-pass (or more) sorting

```python
a0 = sorted(data, key=operator.itemgetter(0))
a1 = sorted(a0, key=operator.itemgetter(2))
a2 = sorted(a1, key=operator.itemgetter(1))
a0
[('a', 2, 0), ('b', 3, 0), ('c', 2, 5), ('d', 2, 4), ('e', 1, 4), ('f', 2, 0)]
a1
[('a', 2, 0), ('b', 3, 0), ('f', 2, 0), ('d', 2, 4), ('e', 1, 4), ('c', 2, 5)]
a2
[('e', 1, 4), ('a', 2, 0), ('f', 2, 0), ('d', 2, 4), ('c', 2, 5), ('b', 3, 0)]
```
SortByFreqs APT
Sort items by their frequency, then sorted in frequencies.

data = ["apple", "pear", "cherry", "apple", "pear", "apple", "banana"]

Returns: ["apple", "pear", "banana", "cherry"]
MedalTable APT

Sort items by their frequency, then sorted in frequencies.

["ITA JPN AUS", "KOR TPE UKR", "KOR KOR GBR", "KOR CHN TPE"]

Returns:
[ "KOR 3 1 0", "ITA 1 0 0", "TPE 0 1 1", "CHN 0 1 0", "JPN 0 1 0", "AUS 0 0 1", "GBR 0 0 1", "UKR 0 0 1" ]
Timingsorts.py, what sort to call?

- Simple to understand, hard to do fast and at-scale
  - Scaling is what makes computer science …
    - Efficient algorithms don't matter on lists of 100 or 1000
  - Named algorithms in 201 and other courses
    - bubble sort, selection sort, merge, quick, …
    - See next slide and TimingSorts.py

- Basics of algorithm analysis: theory and practice
  - We can look at empirical results, would also like to be able to look at code and analyze mathematically! How does algorithm scale?
New sorting algorithms happen …

• timsort is standard on…
  – Python as of version 2.3, Android, Java 7
  – According to http://en.wikipedia.org/wiki/Timsort
    • Adaptive, stable, natural mergesort with supernatural performance

• What is mergesort? Fast and Stable
  – What does this mean?
  – Which is most important?
  – Nothing is faster, what does that mean?
  – Quicksort is faster, what does that mean?
## TimingSorts.py

<table>
<thead>
<tr>
<th>size</th>
<th>create</th>
<th>bubble</th>
<th>select</th>
<th>timsort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.026</td>
<td>0.127</td>
<td>0.081</td>
<td>0.002</td>
</tr>
<tr>
<td>2000</td>
<td>0.045</td>
<td>0.537</td>
<td>0.273</td>
<td>0.001</td>
</tr>
<tr>
<td>3000</td>
<td>0.058</td>
<td>1.126</td>
<td>0.646</td>
<td>0.002</td>
</tr>
<tr>
<td>4000</td>
<td>0.082</td>
<td>2.174</td>
<td>1.208</td>
<td>0.003</td>
</tr>
<tr>
<td>5000</td>
<td>0.101</td>
<td>3.521</td>
<td>1.862</td>
<td>0.003</td>
</tr>
<tr>
<td>6000</td>
<td>0.118</td>
<td>4.617</td>
<td>3.005</td>
<td>0.004</td>
</tr>
<tr>
<td>7000</td>
<td>0.168</td>
<td>7.504</td>
<td>4.237</td>
<td>0.005</td>
</tr>
<tr>
<td>8000</td>
<td>0.156</td>
<td>9.074</td>
<td>6.152</td>
<td>0.007</td>
</tr>
<tr>
<td>9000</td>
<td>0.184</td>
<td>11.611</td>
<td>8.089</td>
<td>0.007</td>
</tr>
<tr>
<td>10000</td>
<td>0.212</td>
<td>14.502</td>
<td>9.384</td>
<td>0.008</td>
</tr>
</tbody>
</table>
Stable, Stability

• What does the search query 'stable sort' show us?
  – Image search explained
  – First shape, then color: for equal colors?
Stable sorting: respect re-order

- Women before men ...
  - First sort by height, then sort by gender
How to import: in general and sorting

• We can write: import operator
  – Then use key=operator.itemgetter(…)

• We can write: from operator import itemgetter
  – Then use key=itemgetter(…)

• From math import pow, From cannon import pow
  – Oops, better not to do that, use dot-qualified names like math.sqrt and operator.itemgetter
TimingSorts.py Questions
bit.ly/101f16-1122-3