CompSci 101
Introduction to Computer Science

April 4, 2017

Prof. Rodger

["ant", 5), ("bat", 4), ("cat", 5), ("dog", 4)]
["ant", 5), ("cat", 5), ("bat", 4), ("dog", 4)]
Inventor of World Wide Web Receives ACM A.M. Turing Award
Sir Tim Berners-Lee Designed Integrated Architecture and Technologies that Underpin the Web

ACM named Sir Tim Berners-Lee, a Professor at Massachusetts Institute of Technology and the University of Oxford, the recipient of the 2016 ACM A.M. Turing Award. Berners-Lee was cited for inventing the World Wide Web, the first web browser, and the fundamental protocols and algorithms allowing the Web to scale. Considered one of the most influential computing innovations in
Announcements

• Exam 2 one week!
• Assignment 7 due Thursday
• APT 8 and APT Quiz 2 due today
  – Doing extra ones – good practice for exam
• Lab this week!
• Review Session – Mon, April 10 7:15pm, LSRC B101
• Today:
  – Finish notes from last time – Dictionary timings
  – Reviewing for the exam
Snarky Hangman

• Version of Hangman that is hard to win.
• Program keeps changing secret word to make it hard to guess!
• User never knows!
• Once a letter is chosen and shown in a location, program picks from words that only have that letter in that location
• Program smart to pick from largest group of words available
Snarky 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
Snarky 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: bcdghjklnopqrstuvwxyz
  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:

  _a__ 147
  _add_ 1
  _a_d_ 17
  _ad__ 3
  dadd_ 1
  da_d_ 1
  da__ 6

  Choose “no d”, most words, 147
  Only 17 words of this type
  Only 1 word of this type
Everytime guess a letter, build a dictionary based on that letter

• Example: Four letter word, guess o

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;O _ O _&quot;</td>
<td>&quot;OBOE&quot;, &quot;ODOR&quot;</td>
</tr>
<tr>
<td>&quot;_ O O _&quot;</td>
<td>&quot;NOON&quot;, &quot;ROOM&quot;, &quot;HOOP&quot;</td>
</tr>
<tr>
<td>&quot;_ O _O&quot;</td>
<td>&quot;SOLO&quot;, &quot;GOTO&quot;</td>
</tr>
<tr>
<td>&quot;_ _ O&quot;</td>
<td>&quot;TRIO&quot;</td>
</tr>
<tr>
<td>&quot;O _ _ _&quot;</td>
<td>&quot;OATH&quot;, &quot;OXEN&quot;</td>
</tr>
<tr>
<td>&quot;_ _ _ _&quot;</td>
<td>&quot;PICK&quot;, &quot;FRAT&quot;</td>
</tr>
</tbody>
</table>

• Key is string, value is list of strings that fit
Keys can’t be lists

• [“O”,”_”,”O”,”_”] need to convert to a string to be the key representing this list: “O_O_”
Snarky Hangman

• How to start? How to modify assignment 5?
DifferentTimings.py

Problem:

• Start with a large file, a book, hawthorne.txt
• For each word, count how many times the word appears in the file
• Create a list of tuples, for each word:
  – Create a tuple (word, count of word)
• We will look at several different solutions
DifferentTimings.py

Problem: (word,count of word)

• Updating (key,value) pairs in structures
• Three different ways:
  1. Search through unordered list
  2. Search through ordered list
  3. Use dictionary
• Why is searching through ordered list fast?
  – Guess a number from 1 to 1000, first guess?
  – What is $2^{10}$? Why is this relevant? $2^{20}$?
  – Dictionary is faster! But not ordered
Linear search through list o' lists

• Maintain list of [string, count] pairs
  – List of lists, why can't we have list of tuples?

  \[
  \begin{array}{c}
  \left[ \['dog', 2\], ['cat', 1], ['bug', 4], ['ant', 5] \right] \\
  \left[ ['dog', 2], ['cat', 2], ['bug', 4], ['ant', 5] \right] \\
  \left[ ['dog', 2], ['cat', 2], ['bug', 4], ['ant', 5], ['frog', 1] \right]
  \end{array}
  \]

  – If we read string 'cat', search and update

  \[
  \begin{array}{c}
  \left[ ['dog', 2], ['cat', 2], ['bug', 4], ['ant', 5] \right] \\
  \left[ ['dog', 2], ['cat', 2], ['bug', 4], ['ant', 5], ['frog', 1] \right]
  \end{array}
  \]

  – If we read string 'frog', search and update

  \[
  \begin{array}{c}
  \left[ ['dog', 2], ['cat', 2], ['bug', 4], ['ant', 5], ['frog', 1] \right]
  \end{array}
  \]
def linear(words):
    data = []
    for w in words:
        found = False
        for elt in data:
            if elt[0] == w:
                elt[1] += 1
                found = True
                break
        if not found:
            data.append([w, 1])
    return data
Binary Search

Find Narten

How many times divide in half?

$log_2(N)$ for N element list
Binary search through list o' lists

- Maintain list of [string,count] pairs in order

- If we read string 'cat', search and update
  
  \[
  \text{[ [‘ant’, 4], [‘frog’, 2] ]}
  \]

- If we read string ‘dog’ twice, search and update
  
  \[
  \text{[ [‘ant’, 4], [‘cat’, 1], [‘frog’, 2] ]}
  \]

- If we read string ‘dog’ twice, search and update
  
  \[
  \text{[ [‘ant’, 4], [‘cat’, 1], [‘dog’, 1], [‘frog’, 2] ]}
  \]

- If we read string ‘dog’ twice, search and update
  
  \[
  \text{[ [‘ant’, 4], [‘cat’, 1], [‘dog’, 2], [‘frog’, 2] ]}
  \]
def binary(words):
    data = []
    for w in words:
        elt = [w,1]
        index = bisect.bisect_left(data, elt)
        if index == len(data):
            data.append(elt)
        elif data[index][0] != w:
            data.insert(index,elt)
        else:
            data[index][1] += 1
    return data
Search via Dictionary

• In linear search we looked through all pairs
• In binary search we looked at log pairs
  – But have to shift lots if new element!!
• In dictionary search we look at one pair
  – Compare: one billion, 30, 1, for example
  – Note that $2^{10} = 1024$, $2^{20} = \text{million}$, $2^{30} = \text{billion}$

• Dictionary converts key to number, finds it
  – Need far more locations than keys
  – Lots of details to get good performance
def dictionary(words):
    d = {}
    for w in words:
        if w not in d:
            d[w] = 1
        else:
            d[w] += 1
    return [[w, d[w]] for w in d]
### Running times @ $10^9$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00001</td>
</tr>
<tr>
<td>$10^3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^{12}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a real focus in Compsci 201
linear is $N^2$, binary search is $N \log N$, dictionary $N$
Running times @ $10^9$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00001</td>
</tr>
<tr>
<td>$10^3$</td>
<td>0.0</td>
<td>0.0000001</td>
<td>0.00001</td>
<td>0.001</td>
</tr>
<tr>
<td>$10^6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^{12}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a real focus in Compsci 201
linear is $N^2$, binary search is $N \log N$, dictionary $N$
## Running times @ $10^9$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00001</td>
</tr>
<tr>
<td>$10^3$</td>
<td>0.0</td>
<td>0.0000001</td>
<td>0.00001</td>
<td>0.001</td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.0</td>
<td>0.001</td>
<td>0.02</td>
<td>16.7 min</td>
</tr>
<tr>
<td>$10^9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10^{12}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**This is a real focus in Compsci 201**

linear is $N^2$, binary search is $N \log N$, dictionary $N$
Running times @ $10^9$ instructions/sec

This is a real focus in Compsci 201

linear is $N^2$, binary search is $N \log N$, dictionary $N$
Running times @ $10^9$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^2$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00001</td>
</tr>
<tr>
<td>$10^3$</td>
<td>0.0</td>
<td>0.0000001</td>
<td>0.00001</td>
<td>0.001</td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.0</td>
<td>0.001</td>
<td>0.02</td>
<td>16.7 min</td>
</tr>
<tr>
<td>$10^9$</td>
<td>0.0</td>
<td>1.0</td>
<td>29.9</td>
<td>31.7 years</td>
</tr>
<tr>
<td>$10^{12}$</td>
<td>9.9 secs</td>
<td>16.7 min</td>
<td>11.07 hr</td>
<td>31.7 million years</td>
</tr>
</tbody>
</table>

This is a real focus in Compsci 201

linear is $N^2$, binary search is $N \log N$, dictionary $N$
What's the best and worst case?

Bit.ly/101s17-0404-2

• If every word is the same ….
  – Does linear differ from dictionary? Why?
• If every word is different in alphabetical order…
  – Does binary differ from linear? Why?
• When would dictionary be bad?
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
    
    ```python
    x = sorted([(t[1], t[0]) for t in dict.items()])
    x = [(t[1], t[0]) for t in x]
    ```

    ```python
    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)

```python
[('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

\[
a_0 = \text{sorted}(\text{data, key=operator.itemgetter}(0))
\]
\[
a_1 = \text{sorted}(a_0, \text{key=operator.itemgetter}(2))
\]
\[
a_2 = \text{sorted}(a_1, \text{key=operator.itemgetter}(1))
\]

\[
a_0
\]
\[
[('a', 2, 0), ('b', 3, 0), ('c', 2, 5), ('d', 2, 4), ('e', 1, 4), ('f', 2, 0)]
\]

\[
a_1
\]
\[
[('a', 2, 0), ('b', 3, 0), ('f', 2, 0), ('d', 2, 4), ('e', 1, 4), ('c', 2, 5)]
\]

\[
a_2
\]
\[
[('e', 1, 4), ('a', 2, 0), ('f', 2, 0), ('d', 2, 4), ('c', 2, 5), ('b', 3, 0)]
\]
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(…)

cps101 spring 2017